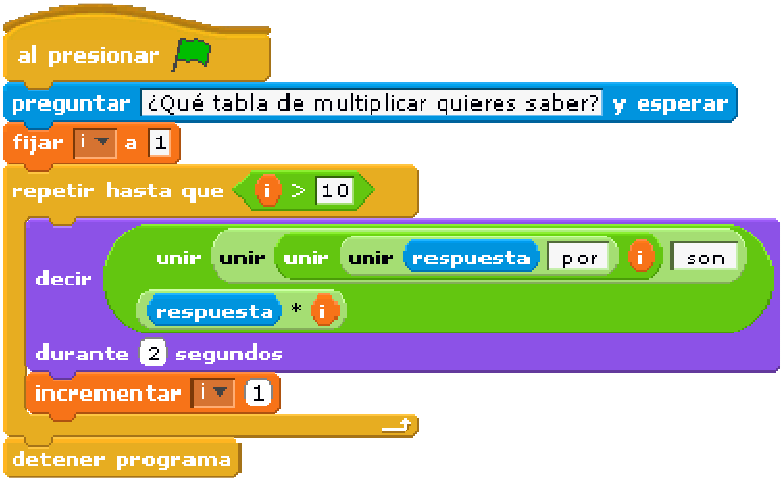
1. PROGRAMA. TIPOS DE INSTRUCCIONES Y DECLARACIONES. OPERACIONES.

Un **programa** es una secuencia de instrucciones y declaraciones que manipulan datos para conseguir un objetivo concreto.



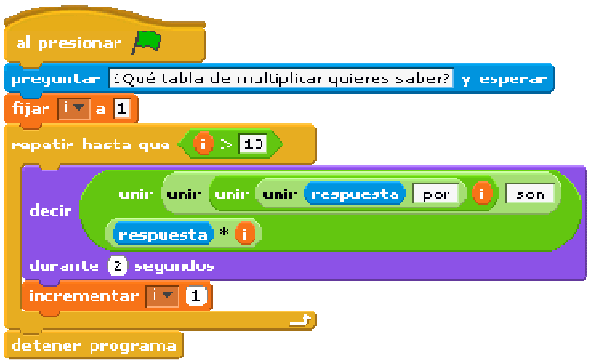
Ejemplo 1: Programa Tabla de Multiplicar

La primera instrucción se llama **inicio** y da comienzo a la ejecución del programa.

La última instrucción se llama **fin** y detiene la ejecución del programa.

Instrucción de inicio

Instrucción de fin



Ejemplo 2: Programa Tabla de Multiplicar. Instrucciones de inicio y fin.

Las instrucciones de **entrada** permiten obtener datos introducidos por el usuario. Las instrucciones de **salida** permiten presentar datos elaborados al usuario.

Instrucción de entrada

Instrucción de salida

Ejemplo 3: Programa Tabla de Multiplicar. Instrucciones de entrada y salida.

Una **declaración** es una instrucción que permite definir variables o funciones.
Una **variable** es una zona de memoria donde almacenar temporalmente datos. Su contenido puede cambiar durante la ejecución del programa.

Ejemplo 4: Programa Tabla de Multiplicar. Declaración de la variable i.

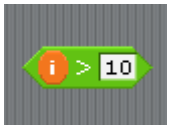
Una **lista** es una tabla de variables de contenido homogéneo que se distinguen la una de la otra por la posición que ocupan.

Una instrucción de **asignación** permite modificar el valor de una variable sustituyendo su contenido o realizando sobre él una operación matemática.

Inst. de asignación

Inst. de asignación

Ejemplo 5: Programa Tabla de Multiplicar. Asignaciones de la variable i.

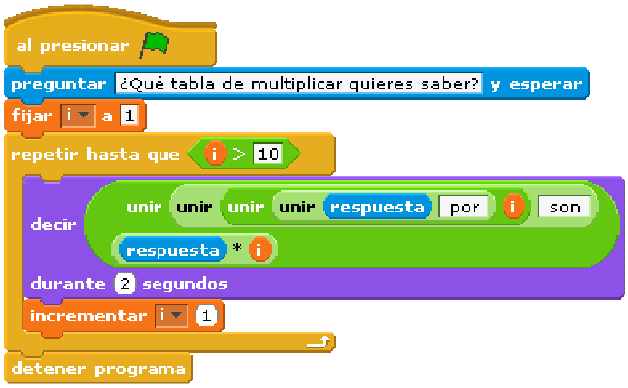


Una **condición** es una expresión que se puede evaluar como cierta o como falsa. Son la parte esencial de las instrucciones de control.

Ejemplo 6: Programa Tabla de Multiplicar. Condición que permite finalizar las iteraciones de la instrucción "repetir hasta que".

Una instrucción de **control** permite regular el flujo de ejecución de un programa *bifurcando* según una condición o *repitiendo* la ejecución de instrucciones (bucle) hasta satisfacerse una condición.

Inst. de control



Ejemplo 7: Programa Tabla de Multiplicar. Instrucción de control para repetir diez veces las instrucciones "decir" e "incrementar".

Una **operación** es un cálculo con constantes, variables o condiciones. Las operaciones pueden ser numéricas (con operadores numéricos) o lógicas (con condiciones).

Scratch tiene definidas las siguientes operaciones:

Operaciones numéricas	**Operaciones lógicas**
Suma (+)	Negación (no)
Resta (-)	Conjunción (y)
Multiplicación (*)	Disyunción (o)
División (/)	***Operadores de las operaciones lógicas (condiciones)***
Residuo (mod)	Menor que (<)
Redondeo (redondear)	Mayor que (>)
Número aleatorio (número al azar entre 1 y 10)	Igual que (=)
Funciones (raíz cuadrada: abs, raíz cuadrada, sin, cos, tan, asin, acos, atan, ln, log, e ^, 10 ^)	**Tablas lógicas**

Tablas lógicas

A	No A
V	F
F	V

A	B	A y B	A o B
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

2. EXPLICANDO EL PROGRAMA TABLA DE MULTIPLICAR.

El programa se construye arrastrando y encajando las piezas que ves en la figura del ejemplo 1.

Los huecos rectangulares anchos son zonas para texto; los estrechos, un texto o un número y los ovalados, un número.

Las instrucciones de inicio/fin están en el menú 

y son:

que inicia la ejecución y 

que la finaliza.

Las instrucciones de entrada/salida están en el menú 

.

La más importante para comenzar a programar es la que nos permite introducir un dato en la variable 

al usar la instrucción 

.

Cuando ejecutemos el programa esta instrucción hará que Calculín nos interrogue y nos muestre una zona de recepción del dato para la variable respuesta (en este caso respuesta=5).

La variable **i** se define en el menú 

.

Nos va a servir para recorrer la tabla de multiplicar desde el 1 hasta el 10.

Por eso inicialmente le asignamos valor 1 con la instrucción 

y en cada iteración la incrementamos en una unidad con la instrucción 

.

La instrucción que hace “cantar” la tabla a Calculín es

y se repite diez veces, desde que i vale 1 hasta que vale 10.

Muestra bocadillos 

como éste:

en el que hemos pillado a Calculín cuando la i vale 3, respuesta vale 5 y respuesta * i vale 5 * 3 = 15. Esta instrucción **decir** es algo complicada porque en ella hemos situado 5 informaciones que hemos tenido que "unir".

La instrucción **unir** se halla en el menú 

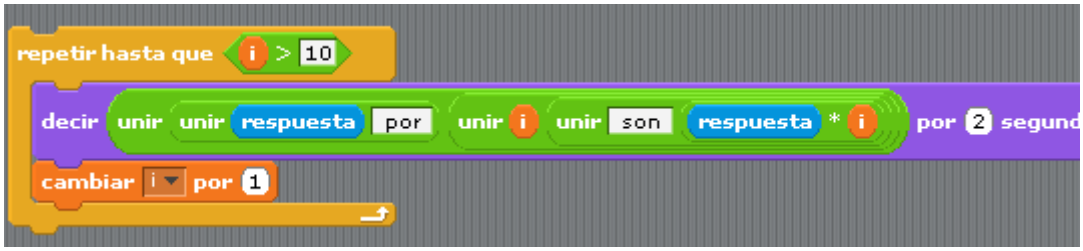
y sirve para juntar **dos** expresiones numéricas o alfabéticas: 

.

Como en nuestro caso tenemos que juntar el valor de respuesta, el literal " por ", el valor de la i, el literal " son " y el valor de respuesta * i hemos necesitado 4 instrucciones unir.

La instrucción que hace repetirse a Calculín está en el menú 

y es

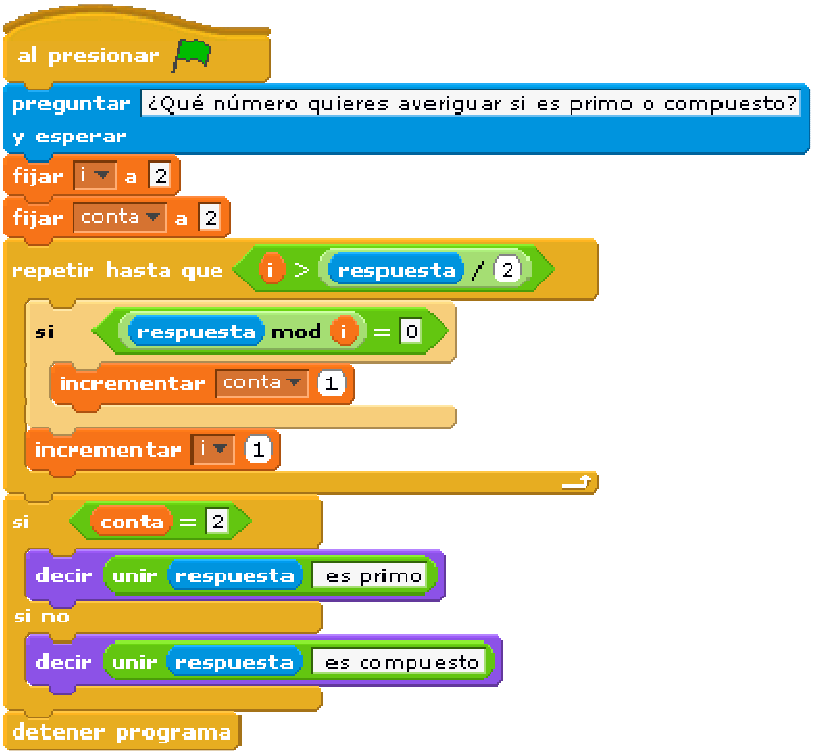


Con ella se REPITEN todas las instrucciones que abraza hasta que se hace cierta la condición 

. Estas instrucciones se llaman **bucles**.

Observa que es de vital importancia que la variable **i** vaya incrementando su valor, dentro del bucle, pues de otra forma no se cumpliría nunca la condición.

3. PRESENTANDO EL PROGRAMA PRIMO O COMPUESTO.



Ejemplo 8: Programa ¿Primo o Compuesto?.

El objetivo de este programa es que Calculín nos informe si un número introducido por nosotros es primo o compuesto. Como recordarás un número es primo si solo tiene 2 divisores: la unidad y él mismo. El planteamiento del programa se basa en esta definición.

Usaremos la variable **conta,** inicialmente con valor 2, para contar los números que dividen exactamente al número ingresado, almacenado en la variable **respuesta**.

La variable **i** recorrerá todos los posibles divisores naturales desde 2 hasta sobrepasar la mitad del número (¿por qué?).

Usamos el operador **mod (**  **)** para comprobar si el resto de la división **respuesta/i** es cero.

En el programa aparecen dos nuevas instrucciones de control:

nos sirve para realizar las instrucciones que abraza si se cumple la condición.

En nuestro caso si **respuesta** es múltiplo de **i** se suma 1 al número de divisores **conta.**

nos sirve para realizar o bien las instrucciones que hay bajo el brazo **si** o bien las que hay bajo el brazo **si no,** dependiendo de si se cumple o no la condición**.**

En nuestro caso si **conta** es igual a 2 Calculín informa que el número es primo y en caso contrario, si **conta** no es igual a 2, el número es compuesto**.**

Para comprobar el contenido de la variable **conta** en diferentes ejecuciones basta con invocar la instrucción **mostrar** o marcar la variable para que se visualice**.**

Ejemplo 9: Dos ejecuciones del programa ¿Primo o Compuesto? señalando el número de divisores encontrados.

4. MEJORANDO EL PROGRAMA PRIMO O COMPUESTO.

Una mejora obvia de nuestro programa es que presente los divisores no triviales de un número si éste es compuesto.

Para ello crearemos mediante la opción de menú **Archivo/Guardar como** una versión inicial del nuevo programa que llamaremos *Primo o compuesto v2.*

La modificación que vamos a realizar consistirá en guardar en una lista los números que dividen al número dado. Esta lista mostrará su contenido en caso de que el número sea compuesto.

La lista **divisores** se define con el botón 

del menú 

.

Cuando detectamos que **i** divide a **respuesta** usamos la instrucción **insertar** un elemento en la última posición de la lista 

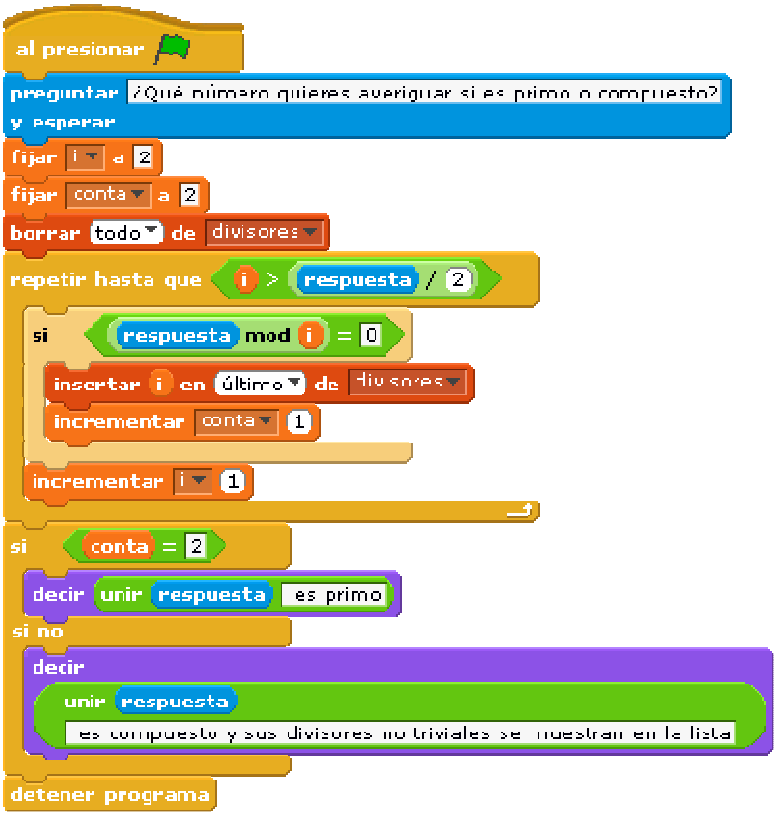
.

La otra instrucción que añadimos es la que asegura que la lista estará vacía al inicio de cada ejecución. Esta instrucción y todas las de inicialización deben figurar al comienzo y son claves para que cada ejecución del programa limpie los datos de la memoria utilizada anteriormente.

Ello se consigue con la instrucción **borrar** 

.

El programa en su nueva versión es:



Ejemplo 10: Programa ¿Primo o Compuesto? mejorado mostrando los divisores de los números compuestos en una lista.

Figura 11: Dos ejecuciones del programa ¿Primo o Compuesto? mostrando los divisores de los números compuestos en una lista.

5. TRUCOS PARA VALIDAR O MANEJAR UNA ENTRADA DE DATOS.

Scratch es un lenguaje débilmente tipado. Eso significa que da poca importancia a cómo son los datos (numéricos, alfabéticos, enteros, etc.) aunque sabe operar con ellos aplicando el sentido común. Veamos algunos trucos para asegurar el tipo de datos:

1) 

Convierte en numérico cualquier entrada de datos que comience con números.
Si este producto vale cero, o bien no hay números en cabecera o bien se trata del cero.

2) 

En la entrada de datos hay algo más que números.

3) 

La entrada de datos no es un entero.

4) La operación 

, aplicada a la respuesta utilizando un bucle con una variable que recorra todos sus dígitos 

, permite recorrer y manipular cada dígito de la entrada. Fíjate en el ejemplo:

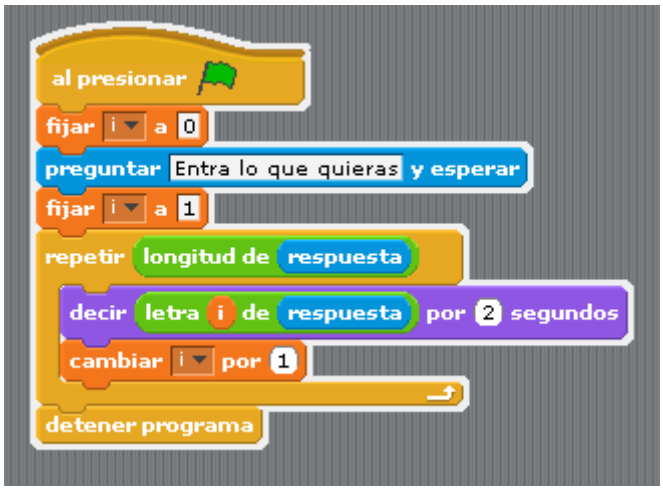


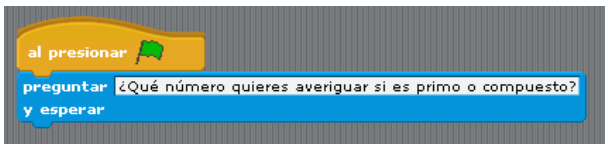
Figura 12: Subrutina que hace decir a Calculín, uno por uno, los dígitos almacenados en respuesta. Lo hemos sorprendido diciendo el cuarto.

6. MODULARIDAD Y RECURSIVIDAD.

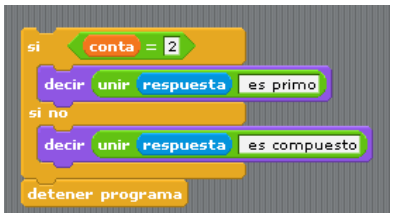
Uno de los lemas fundamentales del buen programador es "Divide y vencerás". Significa que conviene desmenuzar el problema en partes pequeñas y más fáciles de solventar para, a continuación, juntar las partes.

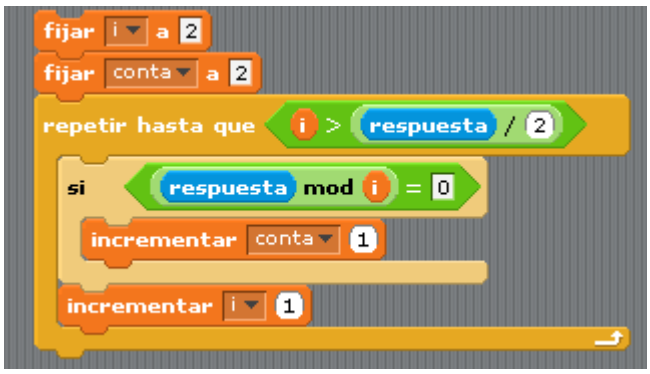
Por ejemplo todos los problemas tienen:

- una parte de **interfaz con el usuario**, donde se resuelve la entrada y la salida de información,
- y una parte **algorítmica**, donde se resuelve el problema central.



Parte del interfaz de usuario





Parte algorítmica

Figura 13: Parte del interfaz de usuario y parte algorítmica del programa Primo o compuesto?. En la primera se pregunta e informa al usuario; en la segunda se calcula la primalidad y los divisores del número dado.

Modularidad es la propiedad que tienen los lenguajes de programación para que subconjuntos de instrucciones que forman una unidad, llamados **funciones, procedimientos, subprogramas o subrutinas**, puedan ser invocados, es decir, llamados para que cumplan su objetivo, recibiendo el control de la ejecución y devolviéndolo al finalizar.

Recursividad es la propiedad de una subrutina de poder llamarse a sí misma. La subrutina ha de tener un caso básico que hace que finalice el proceso recursivo.

Scratch no tiene implementado el uso formal de subrutinas pero **BYOB** (*Build your own blocks*), su versión mejorada, sí.

BYOB permite construir bloques capaces de poner en comunicación fragmentos de programa.

A uno de ellos lo llamaremos **programa principal o llamante** y al otro **subprograma o llamado**. El programa principal detendrá su ejecución y esperará a que el subprograma haya terminado.

BYOB utiliza las instrucciones ejecutar, lanzar o llamar:

Ejecutar (un procedimiento) no informa respuesta alguna, **lanzar** (una función) devuelve un valor y **llamar** (a una función booleana) devuelve cierto o falso.

Es obligatorio ubicar la instrucción 

en el cuerpo de la subrutina lanzada o llamada para depositar y hacer accesible el resultado.

El nombre del nuevo bloque aludirá a la funcionalidad que ha de realizar la subrutina. Ejemplo: EsPrimo.
Cuando el subprograma finalice el programa llamante continuará su ejecución en la instrucción siguiente.
Para coordinar ambos programas usaremos variables o parámetros de **entrada** (el llamante depositará datos que usará el llamado) y de **salida** (donde el llamado depositará sus resultados).
El número de variables de entrada dependerá del objetivo de la subrutina.

Dentro de una función una variable del programa se convierte en una lista con la asignación 

y se asegura que está vacía con la instrucción 

.

No utilices nunca variables ni listas globales en una función. Producen resultados impredecibles.

Se pueden asignar **valores por defecto** a los parámetros de entrada en el momento de su definición haciendo suceder el nombre de la variable del signo igual y del valor. 

Los parámetros de salida deben limpiarse al principio de la ejecución de la subrutina a fin de borrar posibles valores anteriores.

6.1 EJEMPLO DE RECURSIVIDAD. PROGRAMA FACTORIAL.

BYOB también permite la recursividad.

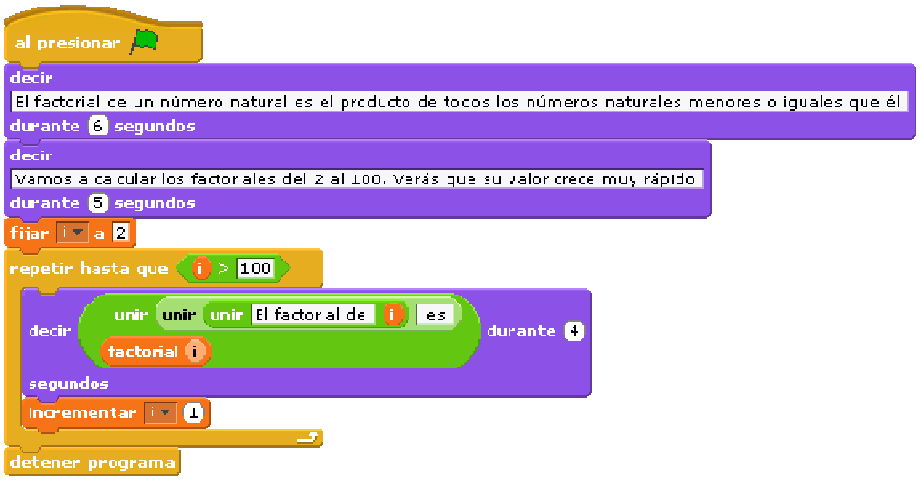


Figura 14.A: Parte del interfaz de usuario del programa Factorial.

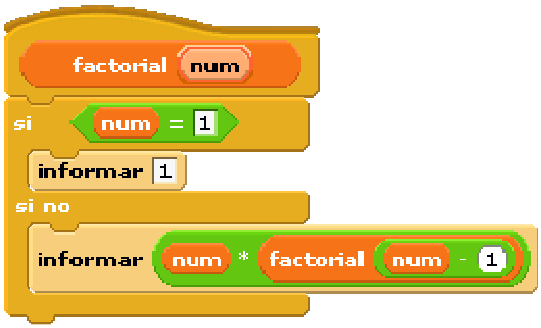


Figura 14.B: Parte algorítmica y recursiva del programa Factorial.
El caso básico es num = 1. Observa como la subrutina se llama a si misma cuando el número no es 1, informando el producto de num y factorial de num-1.
El factorial de 88 tiene 135 cifras y es imposible de obtener en tu calculadora.

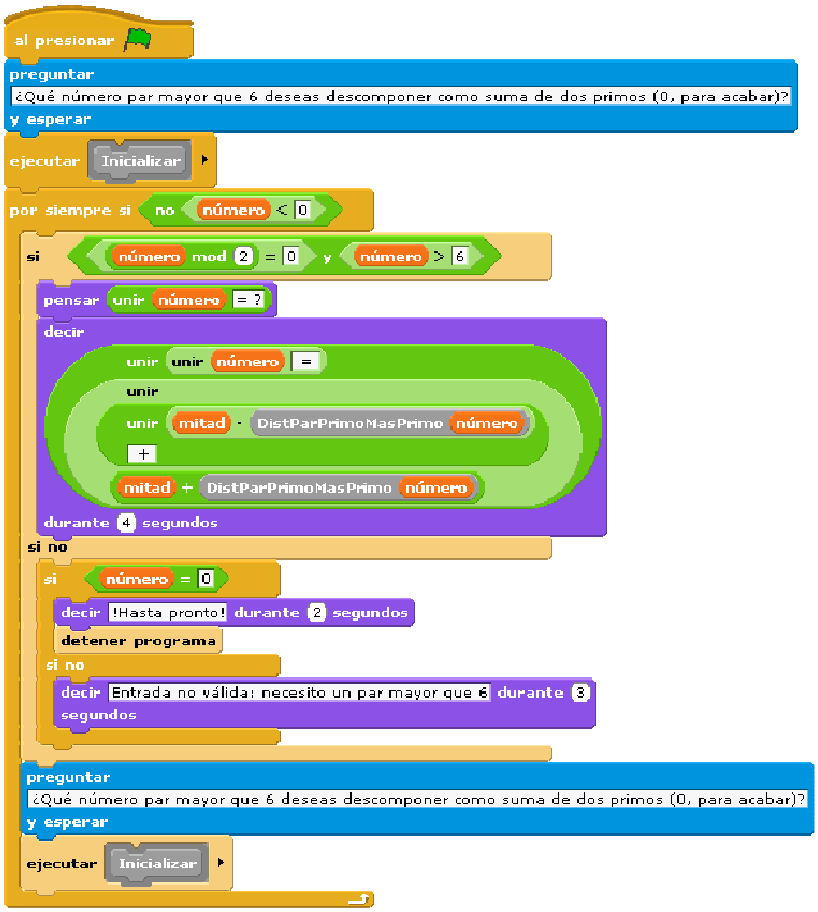
6.2 EJEMPLO DE MODULARIDAD. PROGRAMA QUE DESCOMPONE UN NÚMERO ENTERO PAR MAYOR QUE 8 EN SUMA DE DOS NÚMEROS PRIMOS EQUIDISTANTES DE SU MITAD.

Ejemplifiquemos el propósito de nuestro programa.
Dado el número par 36, su mitad es 18 (compuesto).
Restando y sumando uno obtenemos 17 (primo) y 19 (primo).
La solución buscada, tras un intento, es 36 = 17 + 19.

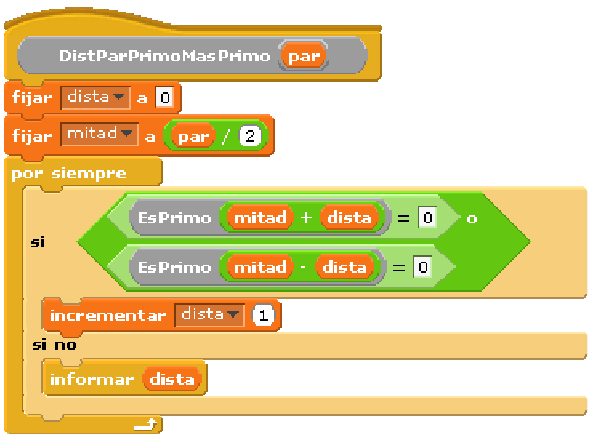
Otro ejemplo: 44. Su mitad es 22 (compuesto).
Restando y sumando uno obtenemos 21 (compuesto) y 23 (primo).
Restando y sumando dos obtenemos 20 (compuesto) y 24 (compuesto).
Restando y sumando tres obtenemos 19 (primo) y 25 (compuesto).
Restando y sumando cuatro obtenemos 18 (compuesto) y 26 (compuesto).

Restando y sumando cinco obtenemos 17 (primo) y 27 (compuesto).
Restando y sumando seis obtenemos 16 (compuesto) y 28 (compuesto).
Restando y sumando siete obtenemos 15 (compuesto) y 29 (primo).
Restando y sumando ocho obtenemos 14 (compuesto) y 30 (compuesto).
Restando y sumando nueve obtenemos 13 (primo) y 31 (primo).
La solución buscada, tras nueve intentos, es 44 = 13 + 31.

El diseño de nuestro programa tendrá tres partes:
1) Interfaz. Pedimos y validamos el número par que desea descomponer el usuario. Invocamos la parte algorítmica. Presentamos el resultado y preguntamos si desea repetir la prueba con otro número.
2) Algoritmo Inicializar. Pone los parámetros a su valor inicial.
3) Algoritmo PrimoParMasPar. Recibimos en par el número. Hallamos su mitad. Fijamos la distancia inicial a cero. Comprobamos si mitad+distancia y mitad-distancia son primos. Si ambos lo son ya hemos obtenido la descomposición buscada. Si alguno no es primo incrementamos en 1 la distancia y volvemos a comprobar la primalidad. Este algoritmo, a su vez, se apoya en otro más simple *Algoritmo EsPrimo.* Recibimos en num el número. Lo dividimos sucesivamente por 2, 3, 4,… hasta llegar a la raíz cuadrada entera del número. Si alguna división del número entre un entero del intervalo [2, √número] es exacta devolvemos 0 en res (no es primo); en caso contrario, devolvemos 1 en res (si es primo).

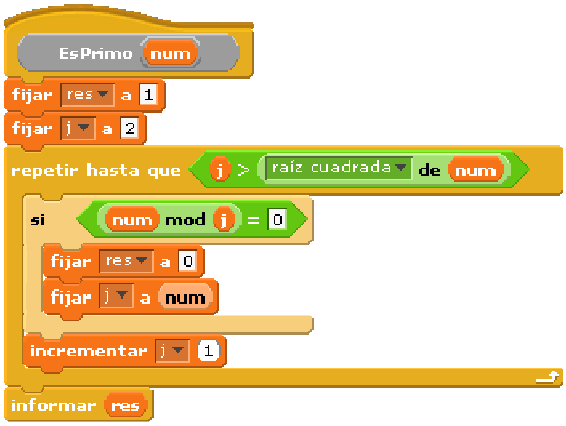


Dialoga con el usuario y controla que el número es entero par mayor que 6



Calcula la distancia desde la mitad de par para su descomposición equidistante y deposita el resultado en dista

Inicializar variables



Testa si un número es primo
Deposita el resultado en res (res=1 es primo; res=0 es compuesto)

Figura 15: Parte del interfaz de usuario y partes algorítmicas del programa Par Igual Primo Más Primo V2.

7. BIBLIOTECA DE FUNCIONES.

BYOB permite crear bloques de funciones y no asignarlas a ningún objeto concreto.
De esta manera puedes crear un proyecto, BIBLIO, que contenga todas las funciones necesarias para ser utilizadas en otros. Bastará importar el proyecto BIBLIO donde sea necesario y se añadirán todas las funciones que contiene a tu nuevo proyecto.
A veces la carga de una biblioteca tarda en exceso. Una alternativa es importar proyectos ya puestos a punto en cualquier momento.
Para los ejercicios a realizar se recomienda crear cuatro bibliotecas de funciones según su objetivo:

- BIB_Numeros
- BIB_Geometria
- BIB_Trigonometria
- BIB_Calculo

8. DISEÑO DEL INTERFAZ.

La interfaz de los programas que realizaremos es siempre del mismo tipo: se muestran varias opciones para que el usuario elija, una vez seleccionada se le piden datos y después del cómputo se le muestran los resultados, pudiendo elegir de nuevo otra opción hasta que decide finalizar.
Usaremos dos objetos: **Calculín** que pedirá los datos y mostrará los resultados y otro objeto, que llamaremos **Menú**, que mostrará las opciones y enviará mensajes a Calculín.
Para el diseño de Menú usaremos una tabla HTML de botones de colores cuya leyenda expresará la opción que se activará al presionar la tecla del carácter que aparezca subrayado.

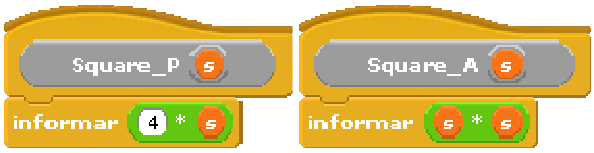
Figura 16. Menú del programa Fórmulas de Geometría

En realidad Menú es una imagen capturada (mediante Impr pant) y recortada (usando Irfanview) de la página web que genera el código que encontrarás en el archivo menu.html.

Para diseñar tu propio Menú basta que modifiques el número y contenido de las leyendas de este archivo con un editor de texto.

Asociado a Menú debes escribir un programa principal que detecte la tecla presionada y envíe el mensaje a Calculín como muestran los fragmentos:

detalle



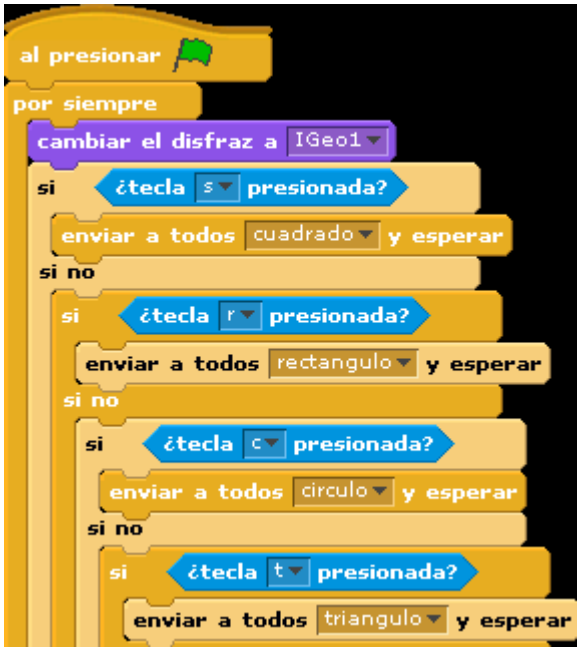


Figura 17. Menú detecta la tecla "s" presionada y envía el mensaje "cuadrado" que recibe y ejecuta Calculín, quien a su vez usa las funciones Square_P y Square_A para informarnos del perímetro y área del cuadrado de lado s.

8.1 POSIBILIDADES DEL INTERFAZ DE SCRATCH.

En el menú 

tenemos una sencilla herramienta de dibujo en pantalla.

La instrucción 

limpia la pantalla. Conviene invocarla al iniciar un dibujo nuevo.

La instrucción 

permite comenzar a pintar el pixel de la pantalla donde se halla el centro del objeto que la tiene asociada.

La instrucción 

inhibe la pintura.

Se puede elegir el color, grosor e intensidad del lápiz:

Propiedad	Instrucción	Opciones
Color		
Tamaño		En número de pixeles
Intensidad		De 0 (muy oscuro: negro) a 100 (muy claro: blanco)

Importante: Se pinta, si el lápiz está bajado, el movimiento o posicionamiento de un objeto.

Según nos convenga o no mostrar el objeto utilizaremos las instrucciones 

y 

del menú 

. Cada objeto puede tener múltiples disfraces asociados.

Para elegir cuál de ellos se muestra usaremos la instrucción 

.

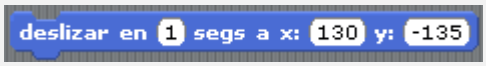
El tamaño del objeto disfrazado se establece con la instrucción 

.

En el menú 

tenemos una potente herramienta para dinamizar los objetos. Existen tres tipos de instrucciones: las que sitúan el objeto, las que fijan la dirección y las que producen el movimiento.

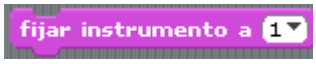
Tipo instrucción	Instrucción	Opciones
Situación	 Establecer las coordenadas	El centro de la pantalla es (0,0) La pantalla tiene un tamaño de 480 pixeles de ancho y 360 de alto. Se muestra desde x=-240 a x=240

	 Modificar las coordenadas	Se muestra desde y=-180 a y=180
Dirección	 Giro horario  Giro antihorario  Dirección puntos cardinales	De 0º a 360º De 0º a 360º 90, este; 0, norte; -90, oeste; 180, sur
Movimiento	 Ir a un punto determinado  Ir a un punto determinado indicando el tiempo a emplear  Desplazarse el número de pixeles indicado en la dirección actual	(x, y) debe estar dentro de la pantalla cuyos vértices visibles son II (-240, 180) I (240, 180) III (-240, -240) IV (240, -180) Permite visualizar el recorrido Asegúrese que la dirección está previamente establecida.

Por último podemos hacer música y reproducir sonidos con el menú  .

Podemos fijar  y modificar  el tempo en pulsos por minuto.

Podemos fijar  y modificar  el volumen.

Podemos establecer qué instrumento sonará  de entre 128 diferentes que se muestran en un desplegable.

Podemos tocar una nota  o un silencio  el número de pulsos indicado.

Podemos reproducir cualquier sonido  previamente grabado.

Hasta aquí un repaso no exhaustivo de posibilidades de Scratch/BYOB.

9. TAREAS DEL PROGRAMADOR.

La tarea más atractiva es la de **diseño**, donde se esboza la solución del problema. En ella se utiliza pseudocódigo, se dibujan diagramas de flujo y se describen tablas de casos con las acciones a realizar. Para establecer los casos se utiliza la combinatoria y las tablas de verdad lógicas.
A continuación viene la tarea de **codificación** donde se escribe la solución en el lenguaje de programación escogido.
El diseño y la codificación con BYOB son la misma tarea. El programa BYOB es a la vez pseudocódigo (en castellano), un diagrama de flujo y un programa sin errores de codificación.
Hasta aquí el 20% del trabajo, la parte más agradable.
La tarea más ardua es la **prueba** o puesta a punto del programa. Mediante diferentes juegos de ensayo (datos de prueba, incluso ilógicos) se verifica el correcto funcionamiento.

BYOB dispone de una herramienta para la puesta a punto: 

Depurar permite detener la ejecución del programa en un determinado punto y observar el contenido de las variables en cada momento tras ejecutar las instrucciones una por una.

Figura 18. Parada mediante la instrucción depurar situada dentro del bucle de la función Collatz para observar la evolución del valor de las variables.

Finalmente viene la tarea de **distribución** o puesta en explotación. Se puede compilar (generar un ejecutable) o ubicar en una página web para que el usuario final pueda disfrutar de él. Conviene añadir la información de ayuda o soporte necesaria para su manejo, es decir, la **documentación** del programa.
Un programa es como un ser vivo y necesita **mantenimiento**. En esta fase se subsanan los errores y se mejoran las prestaciones de acuerdo con las sugerencias de los usuarios.

10. PROYECTOS A REALIZAR.

Hasta aquí toda la teoría que necesitas para realizar cualquiera de los programas que te propongo.

Proyectos a realizar

Nombre	Objetivo del programa
P1	Cuadrados y cubos de los números
P2	Raíz entera y resto de un número
P3	Descomposición de un número en factores primos
P4	Máximo común divisor
P5	Mínimo común múltiplo
P6	Cálculo y verificación de la letra del DNI
P7	Cálculo del dígito de control del ISBN de un libro
P8	Cálculo del dígito de control de un código de barras EAN13
P9	Números triangulares
P10	Ser o no ser triángulo
P11	Cifrado y descifrado con un código de rotación
P12	Adivina el número que he pensado
P13	Master mind de números primos
P14	Los cubos de Nicómaco
P15	Aproximaciones del número PI
P16	Evaluar un polinomio
P17	Palíndromos
P18	Cuentaletras
P19	Traductor de Morse
P20	Números de Eudoxo
P21	Número par como suma de dos primos equidistantes
P22	Números amigos
P23	Día de la semana
P24	Cálculo de la raíz cuadrada (Método de Newton)
P25	Conversor de decimal a binario
P26	Conversor de decimal a octal
P27	El taxi de Ramanujan

P28	Números perfectos
P29	Conversor de medidas forales a actuales
P30	Conversor de números romanos
P31	Operaciones con medidas de ángulos
P32	Conversor de decimal a hexadecimal
P33	Cálculo del día de la Pascua
P34	Generador de apuestas aleatorias
P35	Simulación del lanzamiento de dados
P36	Códigos ASCII y UNICODE
P37	Traductor de alfabeto Braille
P38	Algoritmo de exponenciación rápida
P39	Algoritmo de encriptación RSA
P40	Factorial de un número
P41	Hablar con las vocales
P42	Ordenar varios números
P43	Cálculo de parámetros estadísticos
P44	Detector de progresiones aritméticas
P45	Detector de progresiones geométricas
P46	Sucesión de Fibonacci
P47	Resolutor de triángulos rectángulos
P48	Resolutor de triángulos cualesquiera
P49	Primos gemelos
P50	Perímetros, áreas y volúmenes
P51	Multiplicar a la manera de los egipcios
P52	Cuadrado mágico
P53	Cuadrado mágico con números primos
P54	Método de la orla para cuadrados mágicos de orden par
P55	Conjetura de Brocard
P56	Conjetura de Collatz
P57	Conjetura de Grimm
P58	Conjetura de Oppermann
P59	Conjetura de Polignac
P60	Segunda conjetura de Hardy-Littlewood

P61	Aproximación del número e
P62	Algoritmo babilónico de la raíz cuadrada
P63	Estadística bidimensional
P64	Coordenadas celestes
P65	Operadores ternarios
P66	Piedra, papel o tijera
P67	Copo de nieve de Koch
P68	Espiral cuadrada
P69	Alfombra de Sierpinski
P70	Efecto 2000
P71	Simplificar fracciones
P72	Diferencia de días
P73	Interés simple e interés compuesto
P74	Cuotas de capitalización y de amortización
P75	Resolución de la ecuación de 2º grado
P76	Búsqueda binaria en lista ordenada
P77	Devolver cambio con monedas
P78	Devolver cambio con monedas (y 2)
P79	Comprobación probabilística de primalidad
P80	Generar un número grande probablemente primo
P81	Algoritmo de Maurer para certificar que un número es primo de forma probabilística.
P82	Algoritmo de Agrawal, Kayal y Saxena (AKS-2002) para certificar que un número es primo de forma determinista.
P83	Resolución numérica de ecuaciones. Método de bisección.
P84	Resolución numérica de ecuaciones. Métodos de Whittaker y de Newton-Raphson.
P85	Cálculo de integrales definidas. Fórmulas de Newton-Cotes.
P86	Combinatoria
P87	Las torres de Hanoi
P88	El problema de las ocho reinas

P89	Números apocalípticos
P90	Números económicos
P91	Números felices
P92	Números super-d
P93	Algoritmo 196
P94	La hormiga de Langton
P95	Algoritmo de factorización Pollard-ρ
P96	Casando cadenas de caracteres
P97	Estadísticos robustos: la línea de Tukey
P98	Estadísticos robustos: estimador M de Huber

Recursos para Windows y Bibliografía.

Traducción de BYOB al castellano
Software gratuito BYOB versión 3.1.1
Software gratuito Irfanview para recortar y manipular imágenes
Software gratuito Notepad++ para editar textos
Archivos HTML para el diseño del interfaz
http://wiki.scratch.mit.edu (Wiki de Scratch)
http://www.convertalot.com/ (Conversores y calculadores en tiempo real)
OpenCourseWare de la Universidad Politécnica de Madrid sobre Fundamentos de Programación

C. Gregorio Rodríguez, L.F. Llana Díaz, R. Martínez Unanue, P. Palao Gostanza, C. Pareja Flores. **Ejercicios de Programación creativos y recreativos en C++.** Pearson Educación, S.A., Madrid, 2002.

L. Joyanes Aguilar. **Fundamentos de Programación. Algoritmos, estructuras de datos y objetos.** McGraw Hill, Madrid, 2003.

Robert Sedgewick, Kevin Wayne. **Introduction to Programming in Java. An Interdisciplinary Approach.** Princeton University, 2008.

Charles D. Miller, Vern E. Heeren, John Hornsby. **Matemáticas: razonamiento y aplicaciones. 12ª ed.** Pearson Educación de México, S.A. de C.V., México, 2013.

G. Brassard, P. Bratley. **Fundamentos de Algoritmia.** Prentice Hall, Madrid, 1997.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. **Introduction to Algorithms** (Third edition). The MIT Press, Cambridge, Massachusetts, 2009.

Wolfram MathWorld

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

P1	Cuadrados y cubos de los números
	El cuadrado y el cubo de un número dado es el resultado de multiplicarlo por si mismo dos y tres veces respectivamente. Dado el número 5, su cuadrado es 5^2 y su cubo es 5^3 que valen 25 y 125, respectivamente. Recuerda que $5^3 = 5 \cdot 5 \cdot 5$.
	Solicitar un número natural. Mostrar el cuadrado y el cubo de todos los naturales hasta llegar al del número pedido. V2. Escribir los resultados en sendas listas de Cuadrados y Cubos
	Utiliza una variable que incrementarás dentro de un bloque iterativo hasta superar el número pedido. ☞ V2. No olvides inicializar las listas al principio.

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

P2	Raíz entera y resto de un número
	Un cuadrado perfecto es un número natural resultado de multiplicar un entero por si mismo. Por ejemplo, 25 es un cuadrado perfecto pues $25 = 5^2$. La secuencia de cuadrados perfectos es 1, 4, 9, 16, 25, 36, 49, … Si te fijas la diferencia entre términos consecutivos es la secuencia 3, 5, 7, 9, 11, 13, … , es decir, los números impares.
OBJETIVO	Solicitar un número natural. Informar si es un cuadrado perfecto. Si no lo es señalar entre qué dos cuadrados perfectos se halla. Ejemplo: $5^2 < 31 < 6^2$
	No puedes usar la operación raíz cuadrada. El número de cifras de la raíz cuadrada entera de un número es la mitad del número de cifras del número o la mitad más uno. Recuerda el algoritmo de la raíz que separa las cifras del número de dos en dos comenzando por la derecha y tantea la raíz de la primera pareja con los productos 1x1, 2x2, 3x3, 4x4, 5x5, 6x6, 7x7, 8x8, 9x9
SOLUTIONS	

P3	Descomposición de un número en factores primos
	Todo número natural se puede descomponer de manera única como producto de factores primos. Cuando los números son muy grandes no se conoce ningún algoritmo que resuelva eficientemente este problema; un reciente intento de factorizar un número de 200 dígitos tardó 18 meses y consumió más de medio siglo de tiempo de cálculo en un ordenador.
OBJETIVO	Solicitar un número natural. Guardar en una lista los factores primos que lo dividen de forma exacta. Mostrar la descomposición. V1. Limitar el número hasta 1000000. V2. Escribir en una sola línea la descomposición. Ej: 2000 = 2·2·2·2·5·5·5
	Dividir por la secuencia de naturales 2, 3, 4, 5, … comenzando por el 2 hasta la raíz cuadrada del número. Si la división es exacta seguir probando con el cociente y el mismo número. Si no lo es pasar al siguiente natural de la lista.  ¿Por qué obtendremos sólo divisiones exactas con los números primos aunque los probemos todos?  V1. La longitud de la respuesta ha de ser < 7. Prueba a descomponer 499483, 499523, 988027.

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

P4	Máximo común divisor
	En 1º de ESO has aprendido a calcular el máximo común divisor de varios números. Para ello se descomponen en factores primos y se construye el producto de los factores *comunes* elevados al *mínimo* exponente. El **algoritmo de Euclides** es un procedimiento antiguo y eficaz para calcular el m.c.d. de dos números. Se divide el número mayor entre el menor. Si la división es exacta, el **divisor** es el m.c.d. Si la división no es exacta, dividimos el divisor entre el resto obtenido y se continúa de esta forma hasta obtener una división exacta, siendo **el último divisor** el m.c.d. Ej: el m.c.d.(72, 16) = 8 pues 72 | 16 — 8 | 4 — 16 | 8 — 0 | 2
	Solicitar dos números naturales. Calcular su máximo común divisor. V2. Calcular el máximo común divisor de 3 ó más números.
	Compara los números introducidos para averiguar cuál es el mayor. El mayor será el dividendo y el menor será el divisor. Para conocer el resto de la división entera dispones de la operación  .
	Pseudocódigo **función** Euclides (m, n) r←m MOD n **mientras** r>0 **hacer** m←n n←r r←m MOD n **fin_mientras** **devolver** n

P5	Mínimo común múltiplo
	En 1º de ESO has aprendido a calcular el mínimo común múltiplo de varios números. Para ello se descomponen en factores primos y se construye el producto de los factores *comunes y no comunes* elevados al *máximo* exponente. En el proyecto anterior aprendimos el algoritmo de Euclides para calcular el m.c.d. Pues bien el m.c.d. y el m.c.m. de varios números tienen una curiosa propiedad: **su multiplicación es el producto de los números.** Ej: m.c.d.(72, 16) = 8. Luego m.c.m.(72, 16)= 72·16/8 = 144. En efecto: $72=2^3\cdot3^2$, $16=2^4$ y $m.c.m(72,16)=2^4\cdot3^2=144$
OBJETIVO	Solicitar dos números naturales. Calcular su mínimo común múltiplo, calculando primero su máximo común divisor y aplicando la propiedad enunciada. V2. Calcular el mínimo común múltiplo de 3 ó más números.
	☞ Reutiliza el proyecto P4 Máximo común divisor.

P6	Cálculo y verificación de la letra del DNI

Para evitar errores frecuentes en la grabación de datos, como alterar el orden de dos números consecutivos ("baile de números"), se utilizan los procedimientos de los dígitos de control.
El DNI español tiene una letra al final, que sirve de dígito de control, y recibe el nombre de NIF (número de identificación fiscal).
Para calcular esta letra se divide el número del DNI entre 23 obteniéndose un resto entre 0 y 22. La letra se asigna de acuerdo con la siguiente tabla:

0	1	2	3	4	5	6	7	8	9	10
T	R	W	A	G	M	Y	F	P	D	X

11	12	13	14	15	16	17	18	19	20	21	22
B	N	J	Z	S	Q	V	H	L	C	K	E

Ej: el DNI 19.186.441 tiene la letra W porque 19.186.441 = 834.193 · 23 + 2

OBJETIVO

Solicitar un número de DNI.
Calcular la letra introducida.
Informar el NIF.
V2. Permitir que la entrada pueda contener ya la letra como último dígito. En ese caso verificar que es correcta.

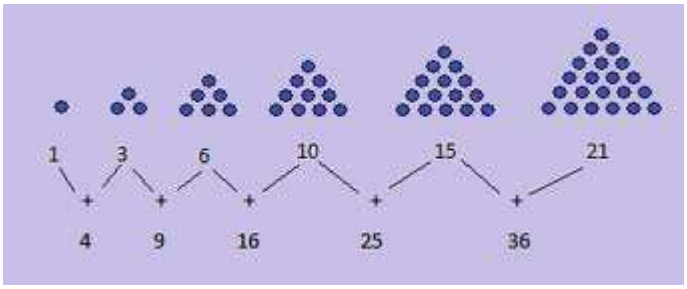
Conocido el resto de la división localiza en una lista a qué letra corresponde. Para ello dispones de la instrucción de lista **ítem**

P7	Cálculo del dígito de control del ISBN de un libro
	El ISBN (International Standard Book Number) es como el DNI de un libro. Cada libro tiene el suyo. Actualmente tiene 13 dígitos (antes tenía 10) que informan el código de país o lengua de origen, el editor, el número del artículo y un dígito de control (el último).Para calcular esta dígito se multiplica el primero de los 12 números iniciales por 1, el segundo por 3, el tercero por 1, el cuarto por 3, y así sucesivamente hasta llegar al número 12. El dígito de control es el valor que se debe añadir a la suma de todos estos productos para hacerla divisible por 10. Ejemplos: si la suma es 97, el dígito de control es 3, porque 97 + 3 = 100, que es divisible por 10; si la suma es 74, el dígito de control será 6; si suman 120, será 0.
	Solicitar un número de ISBN sin dígito de control. Comprobar que tiene 12 dígitos numéricos. Calcular el dígito de control. Informar el ISBN completo. V2. Permitir que la entrada pueda contener ya el dígito de control (longitud 13). En ese caso verificar que es correcto. V3. Programa el cálculo del dígito de control del ISBN de 10 dígitos: Se multiplica cada uno de los nueve primeros dígitos por la posición que ocupan en la secuencia numérica, es decir, el primero por 1, el segundo por dos y así sucesivamente hasta el noveno que se multiplica por 9. Luego se suman estas multiplicaciones y el resultado se divide entre 11. Dicha división dejará un resto entre 0 y 10. Si el resto está entre 0 y 9, este mismo valor es el del dígito de control. Pero si el resto es 10, entonces se establece como dígito de control la letra X.
	Utiliza el truco del bucle que permite manejar cada dígito de la respuesta (punto 5.4)

P8	Cálculo del dígito de control de un código de barras EAN13
	EAN (European Article Number) es un sistema de código de barras presente en la mayoría de artículos de consumo. El código EAN más usual es EAN13, constituido por 13 dígitos y con una estructura dividida en cuatro partes: **código del país** en donde radica la empresa, compuesto por 3 dígitos; **código de empresa**, compuesto por 4 o 5 dígitos, que identifica al propietario de la marca y que es asignado por la asociación de fabricantes y distribuidores y **código de producto**, que completa los 12 primeros dígitos. El 13º dígito es el **dígito de control (DC)**. Es frecuente utilizar el símbolo > al final para indicar zonas en blanco que permiten el adecuado funcionamiento de los escáneres. Para comprobar el dígito de control se suman los dígitos de las posiciones impares, numeradas de derecha a izquierda, y el resultado se multiplica por 3, y se le suman los dígitos de las posiciones pares. Se busca la decena inmediatamente superior y se le resta la suma obtenida. El resultado final es el dígito de control. Ejemplo:  9 789688 874103 Impares: 0+4+8+8+9+7 = 36; Pares: 1+7+8+6+8+9 = 39 36·3+39 = 108+39 = 147. El dígito será 150-147 = 3.
OBJETIVO	Solicitar un código de barras EAN13 sin DC. Comprobar que tiene 12 dígitos numéricos. Calcular el DC. Informar el EAN13 completo. V2. Permitir que la entrada pueda contener ya el DC (13 caracteres). Verificar que es correcto.
	☞ Reutiliza y adapta el programa del ISBN.

P9	Números triangulares

El dibujo describe qué es un número triangular, cómo se forman y por qué se llaman así.



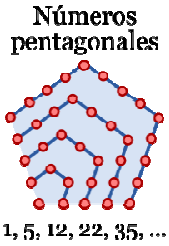
El siguiente de la secuencia se formaría añadiendo la séptima fila de puntos (que tendrá 7 puntos) totalizando 28 puntos. De nuevo comprobamos que 21+28=49, el siguiente cuadrado perfecto 7^2

Una definición más rigurosa nos dice que un número t es triangular si somos capaces de encontrar cierto n que verifica $t = 1 + 2 + 3 + \ldots + n$.

Los números triangulares son un caso especial de números poligonales. Otro ejemplo son los números pentagonales.

El dibujo muestra los 6 primeros.

¿Cuál será el siguiente?



OBJETIVO

Solicitar un número. Comprobar si es triangular.

V2. Comprobar si es pentagonal.

Con un *bucle* ve acumulando la suma 1+2+3+… hasta que alcances o superes el número introducido.

Hay una forma más sencilla.  ¿Recuerdas qué es una progresión aritmética y la fórmula de su suma enésima S_n? Te llevará a una ecuación de 2º grado que habrás de resolver.

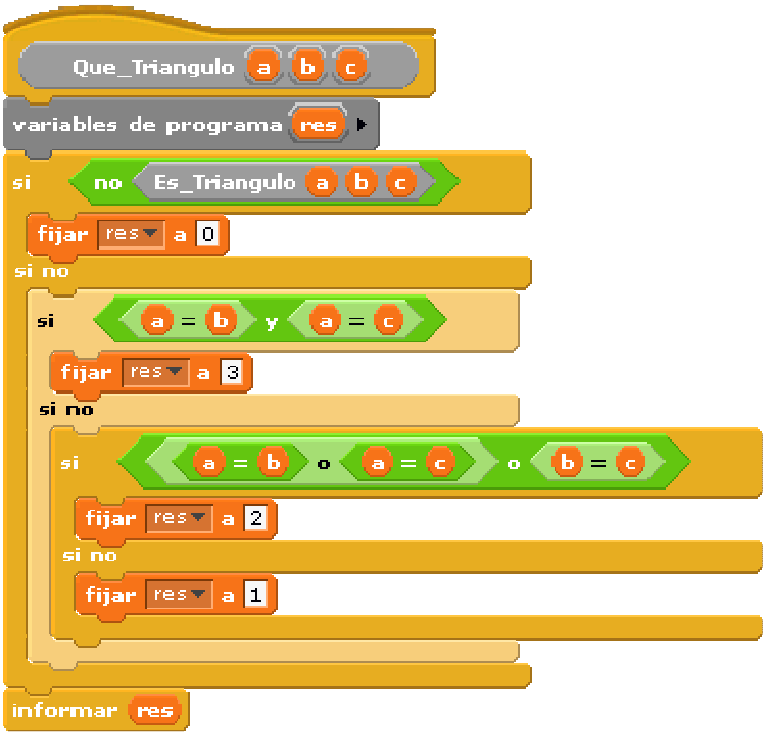

SOLUTIONS


Es triangular n
variables de programa a i t
fijar a a entera abs de n
fijar i a 1
fijar t a 0
repetir hasta que no t < a
incrementar t i
incrementar i 1
si t = a
informar cierto
si no
informar falso


Es poligonal n
variables de programa a i t
fijar a a entera abs de n
fijar i a 1
fijar t a 1
repetir hasta que no t < a
incrementar t 3 * i + 1
incrementar i 1
si t = a
informar cierto
si no
informar falso

P10	Ser o no ser … triángulo
	Las longitudes de los lados de un triángulo en el espacio euclídeo guardan cierta relación: ningún lado puede ser mayor que la suma de los otros dos. Recordemos que un triángulo escaleno tiene los tres lados desiguales, uno isósceles tiene dos lados iguales y el equilátero, como indica su nombre, tiene los tres lados iguales.
OBJETIVO	Solicitar tres números. Comprobar si pueden formar un triángulo. En caso afirmativo, informar de qué tipo es el triángulo (escaleno, isósceles o equilátero).
	Utiliza alguna de las operaciones lógicas de negación  , disyunción  y conjunción  y los operadores de comparación  ,  para la medida de los lados.


Es_Triangulo a b c
si a + b > c y a + c > b y b + c > a
informar cierto
si no
informar falso

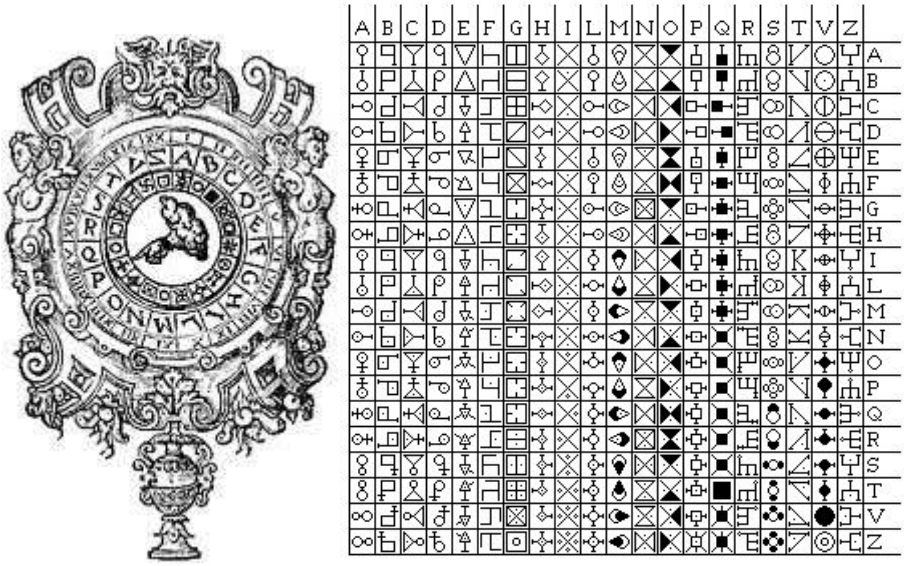

Que_Triangulo a b c
variables de programa res
si no Es_Triangulo a b c
fijar res a 0
si no
si a = b y a = c
fijar res a 3
si no
si a = b o a = c o b = c
fijar res a 2
si no
fijar res a 1
informar res

P11 Cifrado y descifrado con un código de rotación

Julio César utilizaba un código cuando quería mantener en secreto un mensaje. Como el alfabeto latino tiene 21 letras sustituía cada letra por otra de acuerdo con la siguiente tabla:

A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	V
D	E	F	G	H	I	J	K	M	N	O	P	Q	R	S	T	V	A	B	C

Así el mensaje "ALEA IACTA EST" era "DOHD MDFAD HAB" Este cifrado es un código de sustitución aunque en este caso es mejor llamarle de **rotación** pues se mantiene el orden alfabético.



Disco y tabla de cifrado de Giambattista Della Porta

OBJETIVO

Escribe un programa que permita cifrar con un código de rotación, usando el alfabeto castellano y eligiendo la letra que corresponderá a la A.
V2. Una vez elegida la letra asignada a la A, pregunta si el usuario desea cifrar o descifrar.

Utiliza variables de texto. Una de ellas contendrá el alfabeto castellano. La otra contendrá el alfabeto rotado, una vez nos señalen la equivalencia de la A. Construye esta variable con la instrucción unir: desde la letra señalada hasta la Z

unido con desde la A hasta la letra anterior a la señalada. Recorre la frase introducida letra a letra. Localiza la letra en el alfabeto, mira qué posición ocupa, localiza su equivalencia en el alfabeto rotado y sustituye la letra de entrada.

¿Qué hago si la letra escrita no está en el alfabeto o no ha utilizado mayúsculas?

La instrucción 

nos informa el código numérico asignado a un carácter.

La instrucción 

convierte en carácter un código numérico ASCII.

El alfabeto en mayúsculas va del código ASCII 65 (letra A) al 90 (letra Z).

El alfabeto en minúsculas va del código ASCII 97 (letra a) al 122 (letra z).

Averigua el código ASCII de la letra que sirve para cifrar. Supón que es la letra D cuyo código ASCII es 69. ¿Qué operación habrá que hacer para transformar 69 en 65?. Sabiendo que la letra Z tiene código ASCII 90, ¿qué habrá que hacer cuando la resta de un valor inferior a 65?.

¿Qué hago si la letra escrita no está en el alfabeto o no ha utilizado mayúsculas?

P12	Adivina el número que he pensado
	El ordenador pensará un número natural comprendido entre 1 y un número señalado por el jugador. El ordenador solo puede informar si el número sugerido es mayor, menor o igual que el número pensado.
	Acertar el número que ha pensado el ordenador. Solicitar el número máximo e informar si el número sugerido es mayor, menor o igual que el pensado. Llevar un contador de intentos. V2. Sustituir la entrada de un número máximo por la solicitud de grado de dificultad según este criterio: F, fácil, hasta 3 cifras; N, normal, 4 y 5 cifras D, difícil, 6 y 7 cifras. V3. Proporciona una solución automática con el mínimo número de intentos utilizando la técnica de dividir por la mitad el intervalo residual. Ejemplo: el nº máximo es 10000. El ordenador piensa el 5678. La mitad de [0,10000] es 5000. Decimos 5000. El ordenador contesta "Fallaste. El número es mayor". El nuevo intervalo será [5000,10000]. Decimos 7500. El ordenador contesta "Fallaste. El número es menor". El nuevo intervalo será [5000,7500]. Decimos 6250. Y así sucesivamente.
	Utiliza la instrucción que genera un número aleatorio  para simular el número "pensado" por el ordenador.

P13	Master mind de factores primos
	De nuevo un juego de adivinación pero esta vez practicaremos descomposiciones mentales de factores primos. El ordenador pensará un número de 3 cifras que habremos de adivinar. Pero ahora nos ofrecerá pistas. El jugador sugerirá una posible solución y el ordenador informará cuántos y qué factores primos del número sugerido coinciden con los del número pensado. Ejemplo: El ordenador piensa el 236=2·2·59 y el jugador sugiere el 200=2·2·2·5·5. El ordenador informará la coincidencia: 2·2. El jugador deberá seguir sugiriendo números hasta acertar. El número máximo de intentos será 10. Gana aquel que acierta en menor número de intentos.
OBJETIVO	Acertar el número de tres cifras que ha pensado el ordenador. Solicitar la posible solución. Descomponer la entrada e informar los factores coincidentes. Mantener la información de todos los intentos. Controlar el número máximo de intentos (10). V2. Solicitar el grado de dificultad según este criterio: F, fácil, 3 cifras; D, difícil, 4 cifras M, muy difícil, 5 cifras.
	☞ Los factores coincidentes son el m.c.d. de ambos números. ☞ Aloja las coincidencias de cada intento y el número sugerido en una lista para que el jugador pueda razonar cuál puede ser la solución.

P14	Los cubos de Nicómaco
	Nicómaco de Gerasa vivió en Palestina entre los siglos I y II de nuestra era. Escribió Introducción a la Aritmética, primer tratado en que la aritmética se consideraba de forma independiente de la geometría y que se utilizó durante más de mil años como texto básico. Considera la siguiente propiedad descubierta por Nicómaco de Gerasa: Sumando el primar impar se obtiene el primer cubo; sumando los dos siguientes impares, se obtiene el segundo cubo; sumando los tres siguientes, se obtiene el tercer cubo, etc. Comprobémoslo: $1^3 = 1 = 1$ $2^3 = 3 + 5 = 8$ $3^3 = 7 + 9 + 11 = 27$ $4^3 = 13 + 15 + 17 + 19 = 64$
	Generar todos los cubos de Nicómaco. Escríbelos en una lista informando el cubo y el primer y último sumando que lo forman.
	 Puedes detener el programa pulsando el botón rojo. Podrás visualizar cualquiera de los cubos formados con el deslizador hasta ese momento.

P15	Aproximaciones del número π

La razón entre la longitud de una circunferencia y su diámetro es siempre el número π. Ya sabes que π es un número irracional y tiene infinitas cifras decimales que no se repiten.

Hay muchas fórmulas que aproximan el valor de π. Algunas de las más famosas son:

La de François Viète (1540-1603) de 1593:

$$\frac{2}{\pi} = \sqrt{\frac{1}{2}} \cdot \sqrt{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{1}{2}}} \cdot \sqrt{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{1}{2}}}} \cdots$$

La de John Wallis (1616-1703) de 1656:

$$\frac{4}{\pi} = \frac{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \ldots}{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \ldots}$$

La de Gottfried Wilhelm Leibniz (1646-1716) de 1673:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} \cdots$$

Cuatro de las muchas de Leonhard Euler (1707-1783):

$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} \cdots$$

$$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} \cdots$$

$$\frac{\pi^4}{90} = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} \cdots$$

$$\frac{\pi^3}{32} = 1 - \frac{1}{3^2} + \frac{1}{5^2} - \frac{1}{7^2} \cdots$$

La de Borwein de 1987, que converge **muy deprisa** a π:

$$x_0 = \sqrt{2}\,, x_{n+1} = \frac{1}{2}(\sqrt{x_n} + \frac{1}{\sqrt{x_n}})$$

$$y_1 = \sqrt[4]{2}\,, y_{n+1} = \frac{y_n\sqrt{x_n} + \frac{1}{\sqrt{x_n}}}{y_n + 1}$$

$$\pi_0 = 2 + \sqrt{2}$$

$$\pi_n = \pi_{n-1}\frac{x_n + 1}{y_n + 1}$$

A don Manuel Golmayo, ajedrecista español nacido en La Habana en 1883, escribió un poema para recordar las 20 primeras cifras **(3.14159265358979323384)**

Soy y seré a todos definible
mi nombre tengo que daros
cociente diametral siempre inmedible
soy de los redondos aros.

La primera aproximación ofrecida por un ordenador la programó G. W. Reitwiesner en 1949 en un ENIAC y obtuvo 2.037 cifras decimales en 70 horas de proceso.
En 2009 Takahashi halló más de dos billones y medio de decimales con la supercomputadora T2K Tsukuba System, formada por 640 computadoras de alto rendimiento, que juntas consiguen velocidades de procesamiento de 95 teraflops en 73 horas y 36 min.

OBJETIVO

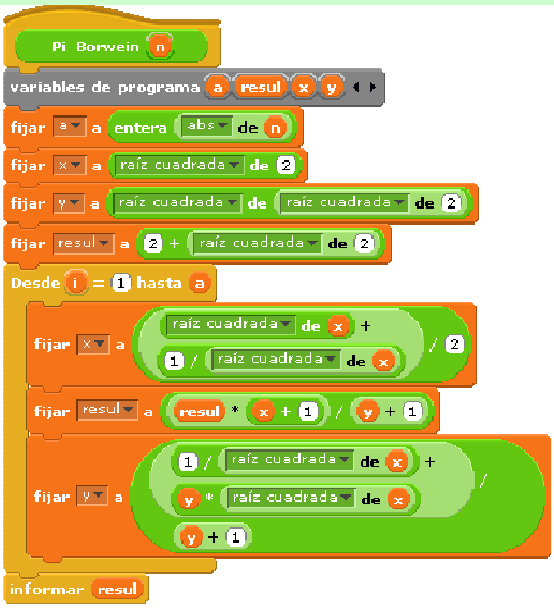
Elige y programa tres de las fórmulas citadas.
Solicita al usuario un valor de n entre 10 y 100.
Calcula el error cometido por cada fórmula.
Presenta los tres valores y sus errores.

La fórmula de Viète también se puede escribir

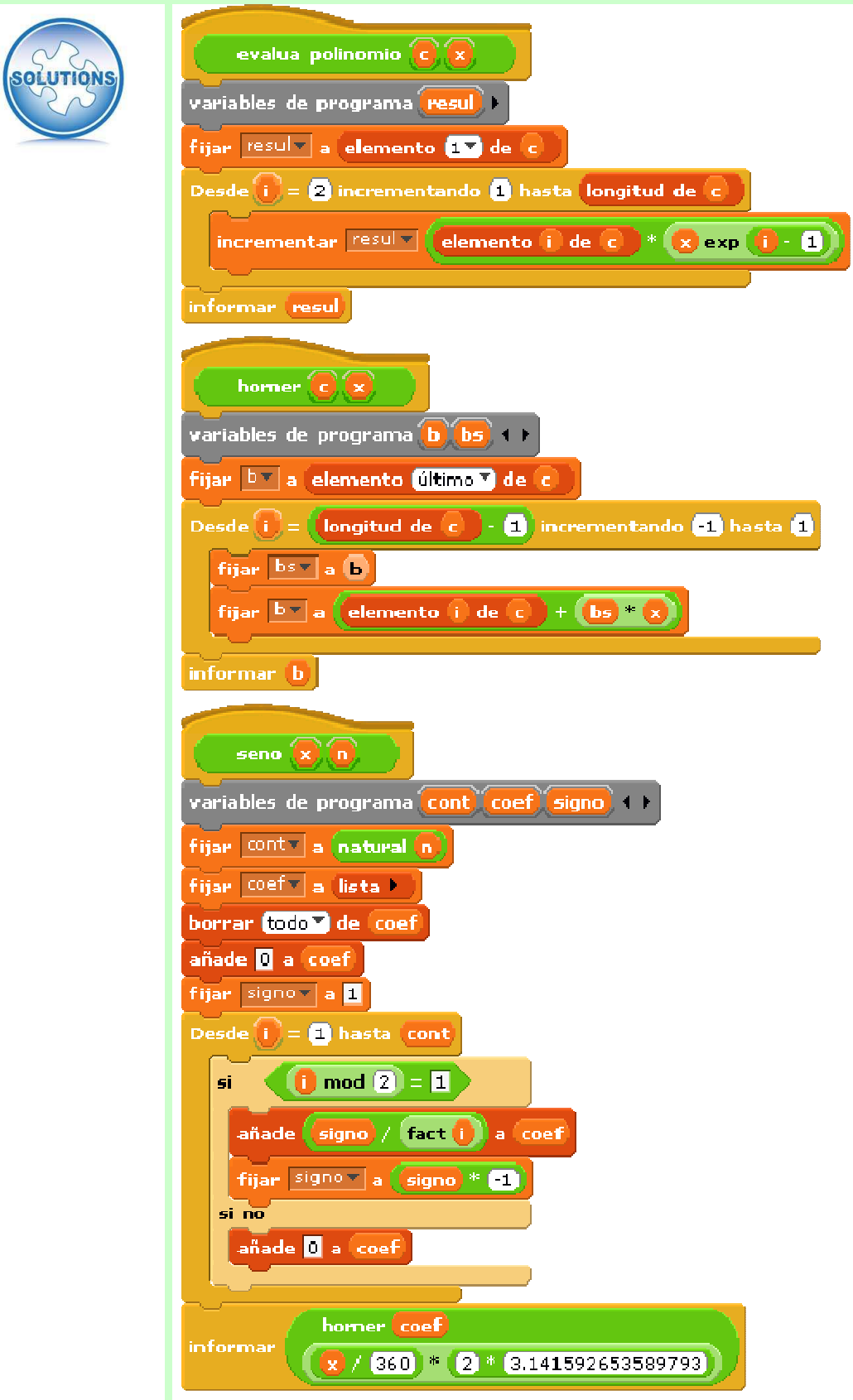
$$\frac{2}{\pi} = \cos\frac{\pi}{4}\cos\frac{\pi}{8}\cos\frac{\pi}{16}\cos\frac{\pi}{32}\ldots$$

El primer coseno del producto es el de 45º, luego el de 45º/2, el de 45º/2^2, …, el de 45º/2^{n-1} en el paso enésimo.

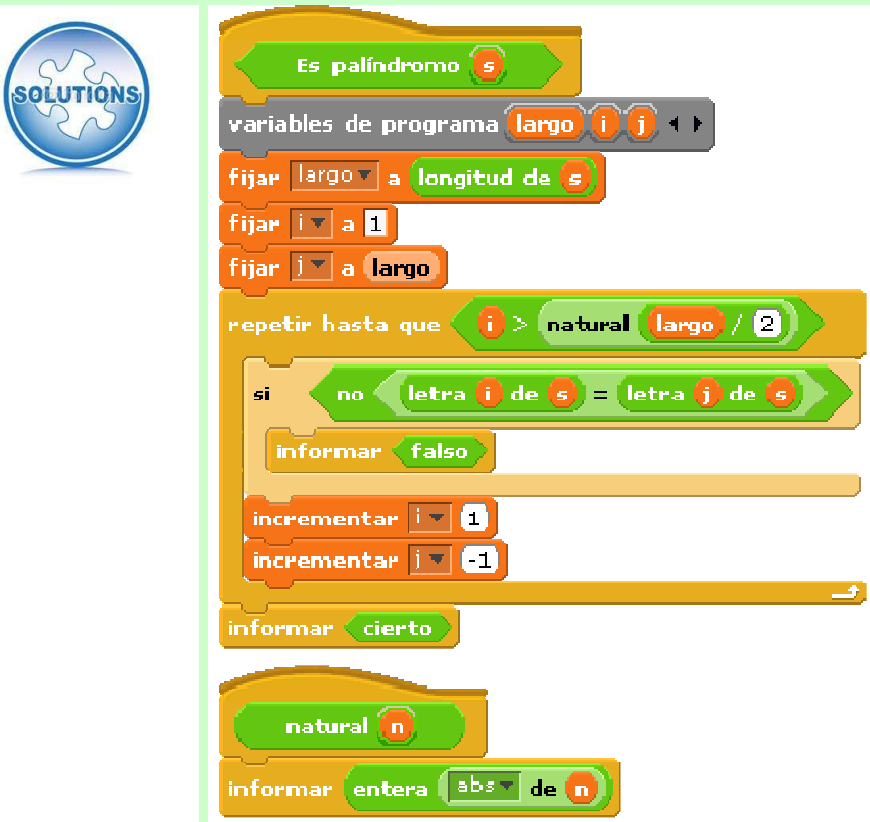
SOLUTIONS



P16	Evaluar un polinomio
	Un polinomio de grado n es una función cuya expresión general es $f(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_nx^n$ siendo a_0, a_1, a_2, a_3, …, a_n coeficientes reales. Evaluar un polinomio es hallar el valor de la función en un punto dado $x=x_0$, es decir, calcular cuánto es $f(x_0)= a_0+a_1x_0+a_2x_0^2+a_3x_0^3+\ldots+a_nx_0^n$ El método de Horner para evaluar polinomios es muy eficiente y se basa en el siguiente uso de los paréntesis $f(x_0)= a_0+x_0(a_1+x_0(a_2+x_0(a_3+\ldots+x_0(a_{n-1}+x_0a_n)\ldots))$. El teorema de Taylor permite aproximar funciones mediante desarrollos polinómicos o series de potencias. Por ejemplo, las funciones trigonométricas $\text{sen } x = x-x^3/3!+x^5/5!+\ldots$ $\cos x = 1-x^2/2!+x^4/4!+\ldots$ y la función exponencial $e^x = 1+x/1!+x^2/2!+x^3/3!+\ldots$ siendo el factorial de n el producto $n!=n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot3\cdot2\cdot1$
	Escribe un programa que evalúe un polinomio de grado n en un punto $x=x_0$ dado. V2. Haz la evaluación por el método de Horner. V3. Calcula el valor del seno, coseno y exponencial mediante los desarrollos polinómicos en series de potencias dados. Aproxima el valor obteniendo de 5 a 20 términos y compáralo con el valor que da el ordenador para la función sen x, cos x y e^x.
	Organiza la entrada de datos: 1º el orden n del polinomio, 2º los n+1 coeficientes y 3º el punto x_0. V2. Define $b_n=a_n$, $b_{n-1}=a_{n-1}+b_nx_0$, … , $b_0=a_0+b_1x_0$. Entonces $b_0=f(x_0)$. V3. Utiliza la rutina "Evaluar un polinomio por el método de Horner" para calcular el seno, coseno y exponencial pedidos.

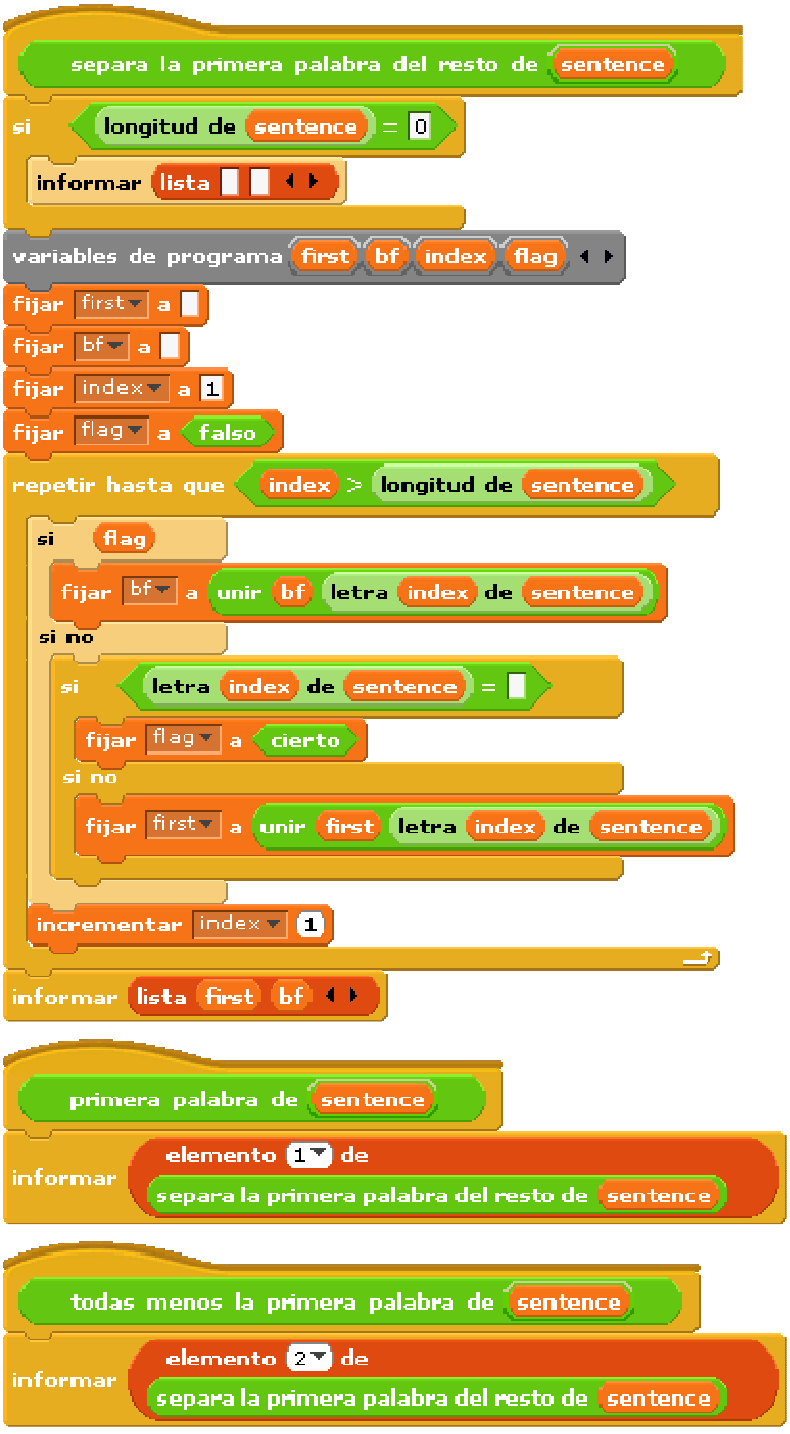

evalua polinomio c x
variables de programa resul
fijar resul a elemento 1 de c
Desde i = 2 incrementando 1 hasta longitud de c
incrementar resul elemento i de c * x exp i - 1
informar resul
horner c x
variables de programa b bs
fijar b a elemento último de c
Desde i = longitud de c - 1 incrementando -1 hasta 1
fijar bs a b
fijar b a elemento i de c + bs * x
informar b
seno x n
variables de programa cont coef signo
fijar cont a natural n
fijar coef a lista
borrar todo de coef
añade 0 a coef
fijar signo a 1
Desde i = 1 hasta cont
si i mod 2 = 1
añade signo / fact i a coef
fijar signo a signo * -1
si no
añade 0 a coef
informar horner coef
x / 360 * 2 * 3.141592653589793

P17	Palíndromos
	Un **palíndromo** es una palabra o frase que se lee igual de izquierda a derecha que de derecha a izquierda, despreciando tildes y espacios en blanco, por ejemplo la palabra "anilina" o la frase "sé verla al revés". En el caso de los números los palíndromos también se llaman **capicúas**, como, el 84148.
	Solicitar una cadena de texto y comprobar si es un palíndromo. Informar del resultado.
	☞ Puedes apoyarte en una subrutina que construya la cadena de texto inversa. A continuación compara cada carácter de la cadena original y la revertida e informa que no es un palíndromo al encontrar una diferencia. Si no encuentras diferencias sí es un palíndromo.


Es palíndromo s
variables de programa largo i j
fijar largo a longitud de s
fijar i a 1
fijar j a largo
repetir hasta que i > natural largo / 2
si no letra i de s = letra j de s
informar falso
incrementar i 1
incrementar j -1
informar cierto
natural n
informar entera abs de n

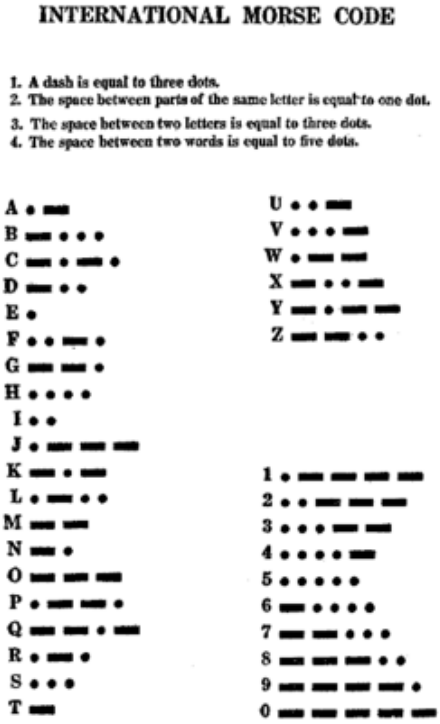
P18	Cuentaletras
	Un **cuentaletras** es un programa que recibe como entrada una cadena de palabras y va contando y escribiendo el número de letras de cada una de ellas. Ejemplo: Al poema de Golmayo *Soy y seré a todos definible* *mi nombre tengo que daros* *cociente diametral siempre inmedible* *soy de los redondos aros* respondería **3141592653589793238 4**
	Solicitar un trozo de texto e informar cuántas letras tiene cada palabra que lo forma.
	 Procesa letra a letra la cadena de entrada y cuando coincida con un espacio en blanco informa el valor del contador de letras (de la palabra que finaliza). No olvides colocar a cero éste contador.


SOLUTIONS


separa la primera palabra del resto de sentence
si longitud de sentence = 0
informar lista
variables de programa first bf index flag
fijar first a
fijar bf a
fijar index a 1
fijar flag a falso
repetir hasta que index > longitud de sentence
si flag
fijar bf a unir bf letra index de sentence
si no
si letra index de sentence =
fijar flag a cierto
si no
fijar first a unir first letra index de sentence
incrementar index 1
informar lista first bf
primera palabra de sentence
informar elemento 1 de separa la primera palabra del resto de sentence
todas menos la primera palabra de sentence
informar elemento 2 de separa la primera palabra del resto de sentence

P19 Traductor de Morse

El alfabeto o código Morse es un sistema de representación de letras y números mediante señales emitidas de forma intermitente en un telégrafo.



Cada letra o número se transmite de forma individual con un código consistente en rayas y puntos, es decir, señales telegráficas que se diferencian en el tiempo de duración de la señal activa. La duración del punto es la mínima posible. Una raya tiene una duración de aproximadamente tres veces la del punto.

Se empleó desde 1830 en las líneas telegráficas mediante los tendidos de cable que se fueron instalando. Más tarde, se utilizó también en las transmisiones por radio, sobre todo en el mar y en el aire, hasta que surgieron las emisoras y los receptores de radiodifusión mediante voz. Tienes un traductor en línea en http://www.otae.com/morse/traduct.htm

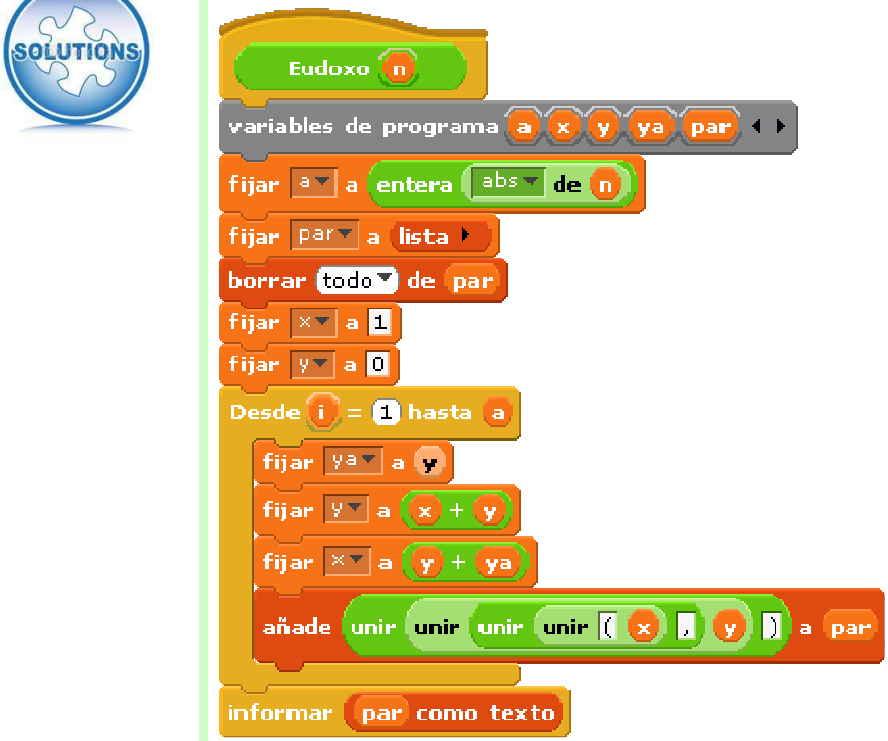
Y otro en http://www.convertalot.com/morse_code_translator.html

OBJETIVO

Solicitar una frase y traducirla a código Morse.
V2. Traductor inverso de Morse a alfabeto clásico.

Procesa letra a letra la cadena de entrada. Utiliza una variable de texto para el alfabeto convencional y una lista para las correspondientes equivalencias de puntos y rayas en código Morse. Ten en cuenta las cuatros reglas que preceden a la tabla de equivalencia de la figura acerca de las separaciones de signos, letras y palabras.

P20	Números de Eudoxo
	**Eudoxo de Cnido** (390 a.C.-337 a.C.) fue un filósofo, astrónomo, matemático y médico griego, el primero en plantear un modelo planetario basado en un modelo matemático, por lo que se le considera el padre de la astronomía matemática. El libro V de los Elementos de Euclides se basa en sus aportaciones sobre las proporciones. Eudoxo elaboró el método de exhausción, antecedente del cálculo integral para calcular áreas y volúmenes, usado por Arquímedes. Los números de Eudoxo se definen así: $x_0 = 1,\ y_0 = 0,\ x_n = y_n + y_{n-1},\ y_n = x_{n-1} + y_{n-1}$ Los primeros pares son (1, 1) (3, 2) (7, 5) y (17, 12) Las yes son conocidos como la secuencia de los números de Pell. Las equis son la secuencia de los múmeros de Pell-Lucas. La razón x_n/y_n tiende al irracional $\sqrt{2}$ hecho que conocía Eudoxo. Los números de Pell se obtienen mediante la fórmula $$P_n = \frac{(1+\sqrt{2})^n - (1-\sqrt{2})^n}{2\sqrt{2}}.$$ y los números de Pell-Lucas con: $$H_n = \frac{(1+\sqrt{2})^n + (1-\sqrt{2})^n}{2}.$$ A $1+\sqrt{2}$ se le conoce como **razón de plata**.
	Solicitar un número natural n. Crear la secuencia de pares (x_i, y_i) para $0<=i<=n$. Mostrar la aproximación de $\sqrt{2}= x_n/y_n$ obtenida.
	Calcula primero las yes y luego las equis.


Eudoxo n
variables de programa a x y ya par
fijar a a entera abs de n
fijar par a lista
borrar todo de par
fijar x a 1
fijar y a 0
Desde i = 1 hasta a
fijar ya a y
fijar y a x + y
fijar x a y + ya
añade unir unir unir unir (x , y) a par
informar par como texto

P21	Número par como suma de dos primos equidistantes
	Una conjetura es una afirmación matemática que se considera cierta pero que no ha sido demostrada. Una de las más famosas, por su simpleza en expresarla, es la de Goldbach (1742): ***Todo número par mayor que 4 se puede expresar como la suma de dos números primos.*** Ejemplos: 18 = 5 + 13 = 7 + 11. 22 = 3 + 19 = 5 + 17 = 11 + 11. Como ves hemos conseguido diversas representaciones como suma de dos números primos. Algunas de ellas (7+11, 11+11) nos da pie a pensar otra conjetura (que se puede demostrar equivalente a la de Goldbach): ***Todo número par se puede expresar como suma de dos números primos equidistantes de su mitad.*** Un ejemplo no trivial: 742.856 = 370.147 + 372.709, donde a la mitad (371.428) hemos sumado y restado 1.281 para obtener los primos 370.147 y 372.709.
	Comprobar la conjetura "par igual a primo más primo equidistante de su mitad" con todos los pares desde 6 hasta 10000.
	Es el ejemplo de modularidad visto en el epígrafe 6.1. 3006 = 1483 + 1523, 6006 = 2969 + 3037 son dos de las descomposiciones que debes obtener.

P22	Números amigos

Dos números a y b son amigos si la suma de los divisores propios de a es igual a b y la suma de los divisores propios de b es igual a a. Los divisores propios de un número incluyen la unidad pero no el propio número. Ejemplo de números amigos son 220 y 284. Los divisores propios de 220 son 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110. Su suma es 284. Los divisores propios de 284 son 1, 2, 4, 71 y 142. Su suma es 220. Por ello, 220 y 284 son amigos. Hacia 850 Tabit ibn Qurra (826-901) descubrió una fórmula general por la cual se podían hallar números amigos:

$p = 3 \times 2^{n-1} - 1$,

$q = 3 \times 2^{n} - 1$,

$r = 9 \times 2^{2n-1} - 1$, donde $n > 1$ es entero y p, q, y r son números primos, entonces $2^n pq$ y $2^n r$ son un par de números amigos. Esta fórmula genera los pares (220, 284), (1184, 1210), (17.296, 18.416) y (9.363.584, 9.437.056).

En occidente, durante muchos siglos, 220 y 284 fueron la única pareja de números amigos conocidos, hasta que en 1636 Euler redescubrió que 17.296 y 18.416 también lo son.

En 1638 Descartes, colega y competidor de Fermat, encontró la tercera pareja: 9.363.584 y 9.437.056.

OBJETIVO

Solicitar dos números naturales, averiguar sus divisores propios y comprobar si son números amigos.

V2. Generar todos los pares de números amigos de 5 cifras. Comprueba tu solución con esta tabla.

a	b
220	284
1 184	1 210
2 620	2 924
5 020	5 564
6 232	6 368
10 744	10 856
12 285	14 595
17 296	18 416
63 020	76 084
66 928	66 992
67 095	71 145
69 615	87 633
79 750	88 730

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

☞ Necesitarás una subrutina que descomponga y sume los divisores propios de cualquier número.

☞ Haz un bucle que descomponga todos los números desde el 10000 hasta el 99999. Si b, suma de los divisores propios de a, tiene 5 cifras comprueba si sus divisores propios sumados dan a.

P23 Día de la semana

Conocemos muchas fechas de nacimiento pero ignoramos en qué día de la semana cayeron.

¿Sabes qué día de la semana naciste?

Las fórmulas que permiten determinar el día de la semana en el calendario gregoriano son:

$a_0 = a-(14-m)/12$

$x = a_0 + a_0/4 - a_0/100 + a_0/400$

$m_0 = m+12\cdot[(14-m)/12]-2$

$d_0 = ((d+x+31\cdot m_0)/12)\%7$

siendo a, año; m, mes; d, día. % es la operación resto módulo.

Ejemplo: 14-2-2000

$a_0 = 2000-(14-2)/12 = 1999$

$x = 1999 + 1999/4 - 1999/100 + 1999/400 = 2483$

$m_0 = 2+12\cdot 1-2=12$

$d_0 = (14+2483+31\cdot 12)/12)\%7=2500\%7=1.$

El 14-2-2000 era lunes.

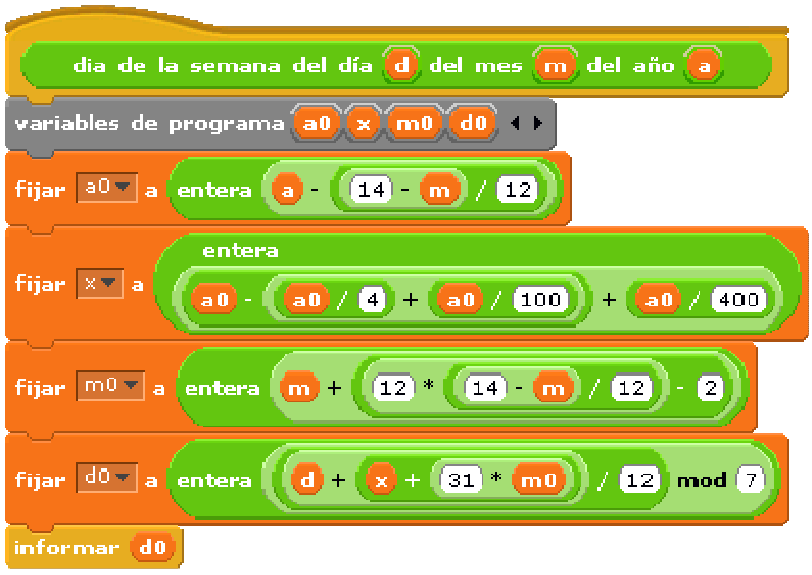
OBJETIVO

Solicita el día, mes y año de una fecha.

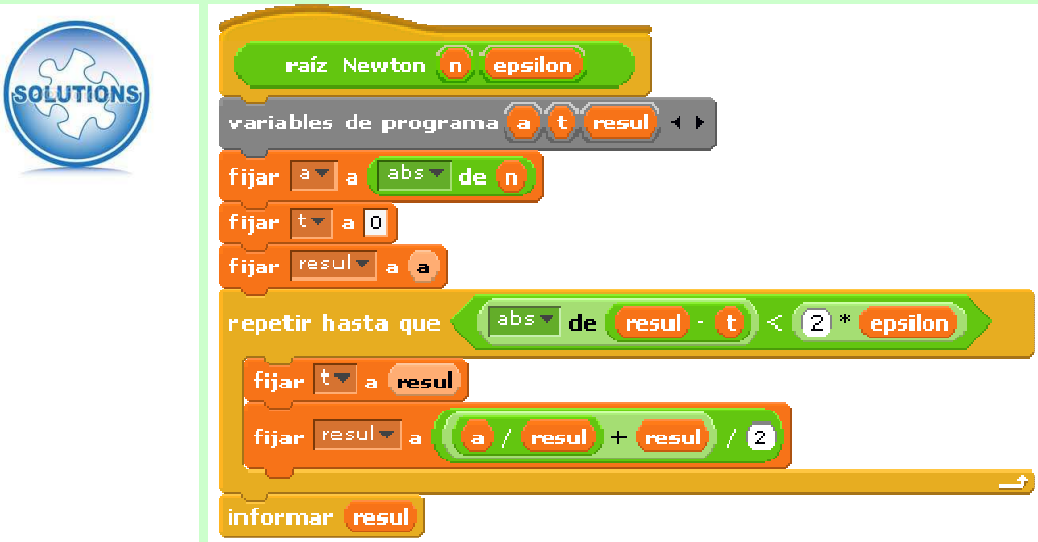
Calcula e informa el día de la semana.

Aplica la fórmulas y contesta según la tabla:

Resto	1	2	3	4	5	6	7
Día	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo



P24	Cálculo de la raíz cuadrada por el método de Newton-Raphson
	El método de Newton-Raphson es un método iterativo y eficiente de encontrar las raíces o ceros de una función f(x). A partir de un valor estimado inicial x_0 el siguiente valor es el punto de corte con el eje 0X de la recta tangente a f(x) que pasa por $(x_0, f(x_0))$. Este nuevo valor estimado lo llamamos x_1 e iteramos el proceso. Obtener la raíz cuadrada de un número c es hallar los ceros de la función $f(x)=x^2 - c$, cuya derivada es $f'(x)= 2x$ por lo que la recta tangente a f(x) será $y-y_0=f'(x_0)(x-x_0)$, es decir, $y-x_0^2-c=2x_0(x-x_0)$. Haciendo y=0 en esta expresión obtenemos $-x_0^2+c = 2x_0x-2x_0^2$ de donde $x_1=(x_0+c/x_0)/2$. Los procesos iterativos deben terminar cuando se alcanza el nivel de precisión deseado, es decir, cuando la diferencia entre dos aproximaciones consecutivas sea menor que una cantidad dada muy pequeña. En este caso, $\|x_i-x_{i-1}\|=\|(x_{i-1}+c/x_{i-1})/2-x_{i-1}\|=$ $=\|(c/x_{i-1}-x_{i-1})/2\|<\epsilon$, es decir, $\|c/x_{i-1}-x_{i-1}\|< 2\epsilon$.
OBJETIVO	Calcular la raíz cuadrada de un número positivo mediante el método de aproximación de Newton-Raphson.
	☞ Solicita el valor de c. Comenzando con $x_0=c$ haz un bucle que compute la fórmula $x_1=(x_0+c/x_0)/2$ mientras la diferencia $\|c/x_{i-1}-x_{i-1}\|$ supere 2ϵ (dos épsilon). ☞ Épsilon es una constante que puedes cambiar. Por ejemplo inicialmente puede ser una cienmilésima: 10^{-5} ☞ Ejemplo de pseudocódigo con el algoritmo de cálculo de la raíz. No usa épsilon de aproximación sino que deja que sea el ordenador quien estime cuando t=r. Fíjate que usa como valor inicial r=x. función $\mathrm{raiz}(x)$ $r \leftarrow x$ $t \leftarrow 0$ mientras $t \neq r$ $t \leftarrow r$ $r \leftarrow \frac{1}{2}\left(\frac{x}{r}+r\right)$ devolver r


raíz Newton n epsilon
variables de programa a t resul
fijar a a abs de n
fijar t a 0
fijar resul a a
repetir hasta que abs de resul - t < 2 * epsilon
fijar t a resul
fijar resul a a / resul + resul / 2
informar resul

P25 Conversor de decimal a binario

La unidad fundamental de memoria en Informática es el bit. Un bit puede valer 0 ó 1. Un byte son 8 bits y con él podemos representar hasta 2^8 = 512 caracteres de información diferentes. Con 2 bytes tendríamos 16 bits disponibles y hasta 2^{16} = 65536 caracteres de información diferentes. Con 4 bytes tendríamos 32 bits disponibles y hasta 2^{32} = 4294967296 caracteres de información diferentes.

Pero centrémonos en los números naturales.

¿Cómo deberíamos escribir 321 usando sólo ceros y unos, es decir, usando el **código binario**?. Resolvamos primero cómo escribir el 2 y el 3. Necesitamos un bit más. Usando dos bits tenemos 4 posibilidades:

0	1	2	3
00	01	10	11

Al igual que en el sistema decimal, como la base es 10, se tiene la siguiente descomposición polinomial

321 = 3 centenas 2 decenas 1 unidad = $3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0$

En el sistema binario se tiene $3 = 1 \cdot 2^1 + 1 \cdot 2^0$. Es decir el valor del 1 depende de la posición que ocupe:

Pos	1	2	3	4	5	6	7	8	9	10
Vale	2^0=1	2^1=2	2^2=4	2^3=8	2^4=16	2^5=32	2^6=64	2^7=128	2^8=256	2^9=512

¿Cómo escribimos 321 en binario?.

Lo dividimos mientras se pueda entre 2:

321:2=160 y resto 1; 160:2=80 y resto 0; 80:2=40 y resto 0;
40:2=20 y resto 0; 20:2=10 y resto 0; 10:2=5 y resto 0;
5:2=2 y resto 1; 2:2=1 y resto 0;

Escribimos los restos en orden inverso y anteponemos el último cociente: 101000001.

En efecto, $101000001 = 1 \cdot 2^8 + 1 \cdot 2^6 + 1 \cdot 2^0 = 256 + 64 + 1 = 321$.

Solicitar un número natural y pasarlo a código binario. V2. Solicitar una cadena de ceros y unos y escribir el número equivalente en sistema decimal.

V1. Guarda en una lista los restos de las sucesivas divisiones entre 2 hasta que el cociente sea menor que 2. Añade un 1 a la lista. Invierte la lista y preséntala como una cadena única.

V2. Comprueba que la respuesta solo contiene ceros y unos. La longitud de la cadena menos 1 es el exponente base 2 del primer término.


SOLUTIONS


binario n
variables de programa resul a
fijar a a entera n
fijar resul a
si a < 2
informar a
repetir hasta que a = 1
si no a mod 2 = 0
fijar resul a unir a mod 2 resul
incrementar a -1 * a mod 2
si no
fijar resul a unir 0 resul
fijar a a a / 2
fijar resul a unir 1 resul
informar resul
n convertir de s a decimal
variables de programa resul i
fijar resul a 0
fijar i a 1
repetir hasta que i > longitud de n
incrementar resul
letra longitud de n - i + 1 de n *
s expn i - 1
incrementar i 1
informar resul

P26 — Conversor de decimal a octal

El sistema octal usa del 0 al 7, tiene 8 caracteres diferentes y necesita tres bits:

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

Es decir el valor del 1 depende de la posición que ocupe:

Posición	1	2	3	4	5	6
Vale	$8^0=1$	$8^1=8$	$8^2=64$	$8^3=512$	$8^4=4096$	$8^5=32768$

Para pasar un número de decimal a octal, de forma similar a lo realizado con el sistema binario, dividiremos entre 8 mientras se pueda y anotaremos los restos en orden inverso anteponiendo el último cociente.

¿Cómo escribimos 321 en octal?.

Lo dividimos mientras se pueda entre 8:

321:8=40 y resto 1; 40:8=5 y resto 0;

Luego $321 = 501_8 = 5\cdot8^2+1\cdot8^0 = 5\cdot64+1=320+1=321$

OBJETIVO

Solicitar un número natural y pasarlo a código octal.

V2. Solicitar una cadena en octal y escribir el número equivalente en sistema decimal.

Reutiliza P26.

V2. Comprueba que la respuesta solo contiene elementos del conjunto {0, 1, 2, 3, 4, 5, 6, 7}.


octal n
variables de programa resul a
fijar a a entera n
fijar resul a
si a < 8
informar a
repetir hasta que a = 1
si no a mod 8 = 0
fijar resul a unir a mod 8 resul
incrementar a -1 * a mod 8
si no
fijar resul a unir 0 resul
fijar a a / 8
fijar resul a unir 1 resul
informar resul

P27	El taxi de Ramanujan

Srinivasa Ramanujan (1887-1920) fue un matemático hindú famoso por su gran intuición con los números. Estando enfermo fue a visitarlo el matemático británico G. H. Hardy (1877-1947) quien comentó que le había traído el taxi 1729, un número bastante insulso. A lo que Srinivasa contestó: "No, Hardy. Es un número muy interesante. Es el más pequeño que se puede expresar como suma de dos cubos de dos maneras diferentes". En efecto, $1729 = 1^3+12^3 = 9^3+10^3$

Otros números que poseen esta propiedad habían sido descubiertos por el matemático francés Bernard Frénicle de Bessy (1602-1675) :

- $2^3 + 16^3 = 9^3 + 15^3 = 4104$
- $10^3 + 27^3 = 19^3 + 24^3 = 20683$
- $2^3 + 34^3 = 15^3 + 33^3 = 39312$
- $9^3 + 34^3 = 16^3 + 33^3 = 40033$

Ramanujan afirmaba que la diosa Namagiri le dictaba sus resultados en sueños. Fue un apasionado del número π del que obtuvo cientos de fórmulas. Una de ellas, demostrada por semejanza de ciertos triángulos construidos sobre una semicircunferencia, es todo un poema geométrico-aritmético:

$$\pi = \sqrt[4]{9^2 + \frac{19^2}{22}} = \sqrt[4]{102 - \frac{2222}{22^2}} = \sqrt[4]{97 + \frac{1}{2} - \frac{1}{11}}$$

OBJETIVO

Escribe un programa que solicite un natural N y escriba todos los naturales menores que él que se pueden expresar como suma de dos cubos de dos maneras diferentes. Es decir, que encuentre 4 naturales distintos a, b, c y d que verifiquen $a^3+b^3=c^3+d^3$.

Utiliza 4 **bucles anidados**.

Un bucle anidado es un bucle dentro de otro bucle.

P28	Números perfectos
	Un número perfecto es un número amigo de sí mismo, es decir, equivale a la suma de sus divisores propios. 6 es un número perfecto porque sus divisores propios son 1, 2 y 3; y 6 = 1 + 2 + 3. Los siguientes números perfectos son 28, 496 y 8128. Considerando la suma de los divisores propios, existen otros tipos de números. • Números *defectivos*: la suma es menor que el número. • Números *abundantes*: la suma es mayor que el número. • Números *semiperfectos*: la suma de algunos de los divisores propios es igual al número. Euclides demostró que la fórmula $2^{n-1}(2^n - 1)$ genera un número perfecto par siempre que $2^n - 1$ es primo. No se conoce la existencia de números perfectos impares.
	Escribe un programa que solicite un número natural y nos diga si es abundante, perfecto o defectivo. V2. En números defectivos el programa debe indicar si son semiperfectos.
	☞ Reutiliza el P22 Números amigos.


SOLUTIONS


Es abundante n
variables de programa a sa
fijar a a entera abs de n
fijar sa a suma divisores propios de a
si a > sa
informar cierto
si no
informar falso
Es defectible n
variables de programa a sa
fijar a a entera abs de n
fijar sa a suma divisores propios de a
si a < sa
informar cierto
si no
informar falso
Es perfecto n
informar n es amigo de n

P29	Conversor de medidas forales a actuales

El sistema de pesos y medidas de época foral valenciana, creado por Jaime I el Conquistador, estuvo vigente desde 1240 hasta mediados del siglo XIX y aún era usado por los labradores valencianos en la primera mitad del siglo XX. No era un sistema decimal. Para las longitudes era antropomórfico. Presentaba variaciones de carácter local y para determinados productos.

Medidas monetarias (*lliura, sou i diner*).
Medidas de longitud (*braça, alna, colze, pam, quart, dit, linea*).
Medidas de peso (*tona, càrrega, quintar, rova grossa i prima, lliura, unça, quart, argenç, gra*).
Medidas de peso para carne (*rova, lliura, unça*).
Medidas de peso para pescado, lino, cáñamo y seda (*càrrega, rova, lliura, unça*).
Medidas de superficie (*jovada, cafissada, fanecada, quartó i braça quadrada*).
Medidas de capacidad para áridos (*cafís, fanega, barcella, almut, quarteró*)
Medidas de capacidad para vino (*tonell, bóta, pipa, càrrega, cànter, quart, mitja*)
Medidas de capacidad para aceite (*càrrega, rova, lliura*)

Equivalencias.

1 lliura = 20 sous; 1 sou = 12 diners

1 braça = 9 pams; **1 alna = 4 pams = 0'906 m.**;
1 colze = 2 pams; 1 pam = 4 quarts
1 quart = 3 dits; 1 dit = 12 línees

1 tona = 8 càrregues;
1 càrrega = 10 roves grosses i 12 primes;
1 quintar = 4 roves primes

1 rova grossa = 36 lliures (para lana, especias y metales)
1 rova prima = 30 lliures (para el resto)
1 rova carnissera = 12 lliures carnisseres
1 rova pescatera = 24 lliures pescateres

1 lliura = 12 unces = 355 g.
1 lliura carnissera= 36 unces = 1065 g.
1 lliura pescatera= 18 unces = 532'5 g.

1 unça = 4 quarts; 1 quart = 4 argenços;
1 argenç = 36 grans

1 tonell = 100 pipes; 1 bóta = 60 cànters;
1 pipa = 40 cànters; 1 càrrega = 15 cànters
1 cànter = 4 quarts = 10'77 l. (vino); 1 quart = 2 mitges

1 càrrega = 12 roves
1 rova = 30 lliures = 11'93 l. (aceite);
1 lliura = 12 unces

1 cafís = 6 fanegues; 1 fanega = 2 barcelles;
1 barcella = 4 almuts = 16'75 l (áridos).

	1 almuts = 4 quarterons; 1 quarteró = 2 mitges. 1 jovada = 6 cafissades; 1 cafissada = 6 fanecades; **1 fanecada = 4 quartons = 831'09 m²** 1 quartó = 50 braces quadrades; 1 braça quadrada = 81 pams quadrats.
OBJETIVO	Escribe un programa que convierta medidas de longitud dadas en "alnes, pams i dits" a metros y medidas de capacidad dadas en "cafissos, fanegues i almuts" a litros. V2. Escribe un programa que convierta medidas de longitud y capacidad del sistema métrico decimal al sistema foral.
	☞ Utiliza en las fórmulas las equivalencias apropiadas.

P30	Conversor de números romanos

El sistema de **numeración romana** es un sistema de numeración no posicional que se desarrolló en la Antigua Roma y se utilizó en todo el Imperio romano.

El inventario de signos que utiliza son de 2 tipos:

tipo 1: I, X, C, M y tipo 5: V, L, D

Sus valores son:

Signo	I	V	X	L	C	D	M
Valor	1	5	10	50	100	500	1000

Como regla general, los símbolos se escriben y leen de izquierda a derecha, de mayor a menor valor.

El valor de un número se obtiene sumando los valores de los símbolos que lo componen, salvo si un símbolo de tipo 1 está a la izquierda inmediata de otro de mayor valor, en cuyo caso se resta al valor del segundo el valor del primero. Ej. IV=4, IX=9.

Los símbolos de tipo 5 siempre suman y no pueden estar a la izquierda de uno de mayor valor.

Se permiten a lo sumo tres repeticiones consecutivas del mismo símbolo de tipo I.

No se permite la repetición de una letra de tipo 5. Su duplicado es una letra de tipo 1: el doble de V es X, el doble de L es C, el doble de D es M.

Si un símbolo de tipo 1 aparece restando, sólo puede aparecer a su derecha un sólo símbolo de mayor valor.

Si un símbolo de tipo 1 que aparece restando se repite, sólo se permite que su repetición esté colocada a su derecha y que no sea adyacente al símbolo que resta.

Sólo se admite la resta de un símbolo de tipo 1 sobre el inmediato mayor de tipo 1 o de tipo 5. Ejemplos:

- el símbolo I sólo puede restar a V y a X.
- el símbolo X sólo resta a L y a C.
- el símbolo C sólo resta a D y a M.

Ej: 2013 en numeración romana es MMXIII

El latín fue lengua vehicular de los negocios notariales en época foral. Es frecuente encontrar en los protocolos notariales medievales fechas y cantidades en numeración romana. En ellos se

infringía alguna regla y aparecen repetidos cuatro veces los signos de tipo 1: IIII en lugar de IV, XXXX en lugar de XL y CCCC en lugar de CD.

Escribir un programa que convierta a numeración romana cualquier natural menor que 3000.

Plantea la siguiente tabla de equivalencia que resuelve el problema hasta el 999:

1	2	3	4	5	6	7	8	9
I	II	III	IV	V	VI	VII	VIII	IX

10	20	30	40	50	60	70	80	90
X	XX	XXX	XL	L	LX	LXX	LXXX	XC

100	200	300	400	500	600	700	800	900
C	CC	CCC	CD	D	DC	DCC	DCCC	CM

Pseudocódigo UPSAM 2.0[1] con la solución es:

```
algoritmo Romanos
//Leer el número y se apoya en el procedimiento
//calcifrarom para convertir cada dígito
//div da el cociente de la división entera
var entero : n,r,digito
inicio
   repetir
   escribir ('Deme número')
   leer(n)
   hasta_que (n>=0) Y (n<=3000)
r←n
digito←r div 1000
r←r MOD 1000
calcifrarom(digito, 'M', ' ', ' ')
digito←r div 100
r←r MOD 100
calcifrarom(digito, 'C', 'D', 'M')
digito←r div 10
r←r MOD 10
calcifrarom(digito, 'X', 'L', 'C')
digito←r
calcifrarom(digito, 'I', 'V', 'X')
fin
```

[1] UPSAM 2.0 es el lenguaje algorítmico de programación creado por los profesores del área de programación de la Universidad Pontificia de Salamanca, campus de Madrid bajo la dirección de Don Luis Joyanes Aguilar.

```
//Observa lo incómodo y confuso que resulta utilizar
//la identación para señalar el principio y el fin

procedimiento calcifrarom(E entero:digito; E
caracter:v1,v2,v3)
var entero: j
inicio
 si   digito=9 entonces escribir (v1,v3)
 si_no
   si digito>4 entonces escribir (v2)
        desde j←1 hasta digito-5 hacer escribir (v1)
        fin_desde
  si_no
      si digito=4 entonces escribir (v1,v2)
      si_no
      desde j←1 hasta digito hacer escribir (v1)
      fin_desde
      fin_si
   fin_si
 fin_si
fin_procedimiento
```

P31	Operaciones con medidas de ángulos
	Tanto los ángulos en Geometría como el tiempo en Cronometría utilizan un sistema de base sexagesimal, sistema posicional en el que 60 unidades de un orden forman una unidad de orden superior. Así, para el tiempo 60 segundos (s) son 1 minuto (min) y 60 minutos son 1 hora (h) y para los ángulos 60 segundos (") son 1 minuto (') y 60 minutos son 1 grado (º). Ejemplo de suma: 44º36'55"+18º29'22"=62º65'77"=62º66'17"=63º6'17" Ejemplo de resta: 44º36'22"-18º29'55"= 44º35'82"-18º29'55"=26º6'27"
	Escribir un programa que solicite dos medidas de tiempo o ángulos y ofrezca el resultado de su suma y de su resta (de la mayor restar la menor). V2. Añadir el producto o la división de una medida entre un número natural.
	☞ Solicita las medidas de tiempo en horas, minutos y segundos. Suma horas con horas, minutos con minutos y segundos con segundos. Haz la división entera del resultado (de segundos y minutos) entre 60. El cociente añádelo a la unidad superior. El resto formará parte del resultado. Para la resta las cantidades del minuendo tienen que ser mayores que las del sustraendo. Si no lo son "presta" (quitando 1) una unidad de orden superior y suma 60.

P32 Conversor de decimal a hexadecimal

Usando 4 bits podemos representar 16 caracteres diferentes. El sistema hexadecimal usa del 0 al 9 y de la A a la F.

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111
8	**9**	**A(10)**	**B(11)**	**C(12)**	**D(13)**	**E(14)**	**F(15)**
1000	1001	1010	1011	1100	1101	1110	1111

El valor del 1 depende de la posición que ocupe:

Posición	1	2	3	4	5
Vale	$16^0=1$	$16^1=16$	$16^2=256$	$16^3=4096$	$16^4=65536$

Para pasar un número de decimal a hexadecimal, de forma similar a lo realizado con el sistema binario, dividiremos entre 16 mientras se pueda y anotaremos los restos en orden inverso anteponiendo el último cociente.

¿Cómo escribimos 321 en hexadecimal?.

Lo dividimos mientras se pueda entre 16:

321:16=20 y resto 1; 20:16=1 y resto 4;

Luego 321 = 141_{16} = $1\cdot16^2+4\cdot16^1+1\cdot16^0$ =256+64+1=321

Solicitar un número natural y pasarlo a código hexadecimal.

V2. Solicitar una cadena en hexadecimal y escribir el número equivalente en sistema decimal.

☞ Reutiliza P26.

☞ V2. Comprueba que la respuesta solo contiene elementos del conjunto {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, F}.

P33	Cálculo del día de la Pascua

El día de la Pascua cristiana es el domingo siguiente a la primera luna llena de la primavera boreal (si esta luna cayese en domingo, la Pascua se traslada al domingo siguiente para evitar la coincidencia con la Pascua judía). Como el equinoccio de primavera tiene lugar el 20 o 21 de marzo la Pascua de Resurrección no puede ser antes del 22 de marzo (en caso de que plenilunio el 21 y sábado) ni tampoco puede ser más tarde del 25 de abril, (suponiendo que el 21 de marzo fuese el día siguiente al plenilunio, habría que esperar una lunación completa de 29 días para llegar al siguiente plenilunio, que sería el 18 de abril, el cual, si cayese en domingo, desplazaría la Pascua una semana para evitar la coincidencia con la pascua judía, quedando 18 + 7 = 25 de abril).

La manera más sencilla de calcular esta fecha es mediante la fórmula desarrollada por el matemático alemán **C.F. Gauss** (1777-1855):

Definamos 5 variables, *a*, *b*, *c*, *d*, y *e*. Además de dos constantes *M* y *N*, que para los años comprendidos entre 1900 y 2100 tomarán los valores 24 y 5 respectivamente. Llamaremos A al año del que queremos calcular la Pascua. Entonces a=A%19, b=A%4, c=A%7, d=(19·a+M)%30, e=(2·b+4·c+6·d+N)%7. Si *d* + *e* < 10, entonces la Pascua caerá en el día (*d* + *e* + 22) de marzo. En caso contrario (*d* + *e* > 9), caerá en el día (*d* + *e* - 9) de abril. Existen dos excepciones a tener en cuenta: Si la fecha obtenida es el 26 de abril, entonces la Pascua caerá en el 19 de abril. Si la fecha obtenida es el 25 de abril, con *d* = 28, *e* = 6 y *a* > 10, entonces la Pascua caerá en el 18 de abril.

La tabla completa de valores de M y N para el calendario gregoriano es:

Años	1583-1699	1700-1799	1800-1899	1900-2099	2100-2199	2200-2299
M	22	23	23	24	24	25
N	2	3	4	5	6	0

Otro procedimiento es el **algoritmo de Butcher**. La ventaja con respecto al de Gauss es que no tiene excepciones y es válido para cualquier año posterior a 1583, la desventaja es que es algo más complejo. Se calcula como sigue:

A = Año%19,
B es el cociente de la división Año/100
C = Año%100,
D es el cociente de la división B/4,
E = B%4,
F es el cociente de la división (B+8)/25,
G es el cociente de la división (B-F+1)/3,
H = (19·A + B - D - G + 15)%30,
I es el cociente de la división C/4,
K = C%4,
L = (32 + 2·E + 2·I - H - K)%7,
M es el cociente de la división (A+11H+22L)/451,
N = H + L - 7·M + 114,
MES es el cociente de la división N/31,
DIA = 1 + N%31 o bien 1 + N - MES·31.

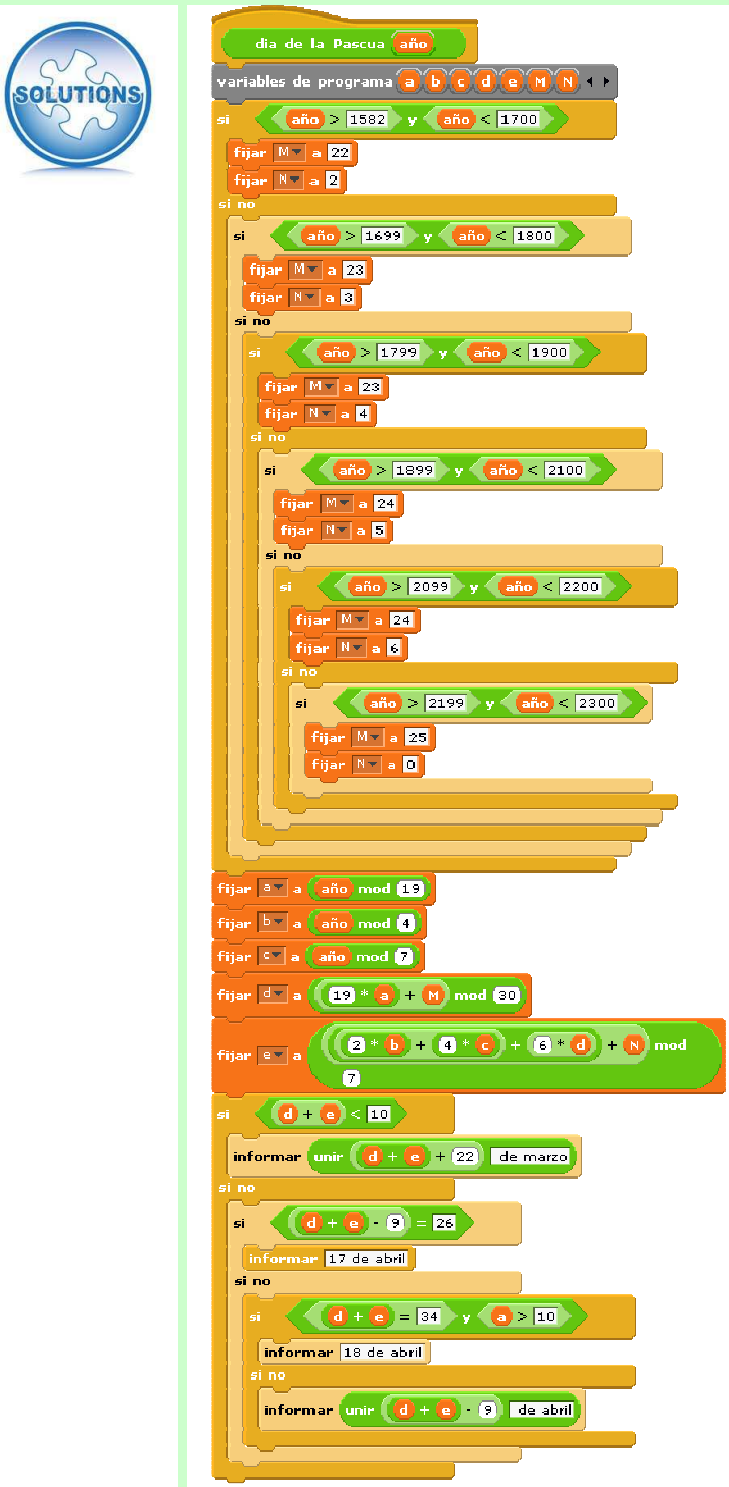
OBJETIVO

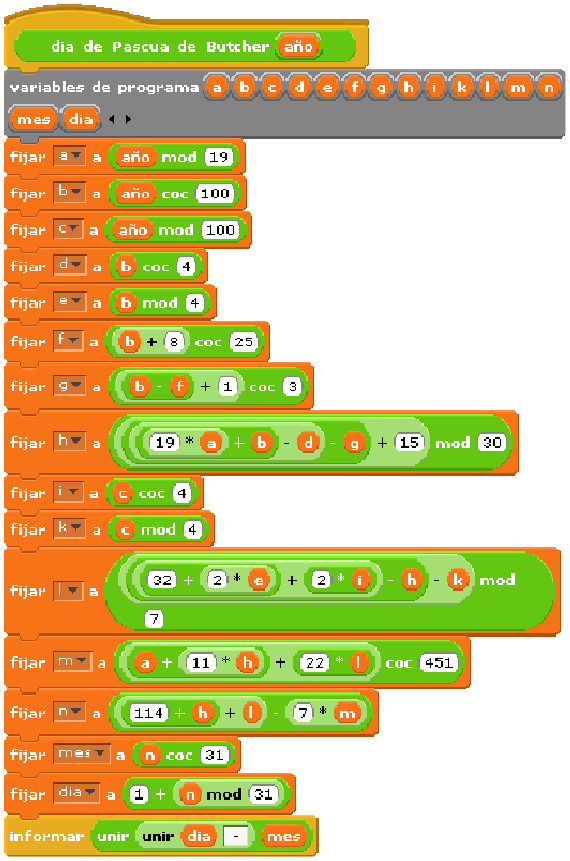
Calcular el día de la Pascua de todos los años de los siglos XX y XXI por el procedimiento de Gauss.
V2. Calcular el día de la Pascua para cualquier año posterior a 1583.
V3. Calcular el día de la Pascua para cualquier año de los siglos XX y XXI por el algoritmo de Butcher y comparar los resultados con los de la V1.

Usa las fórmulas indicadas.

Tabla parcial de resultados para comprobar:

2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
12-4	4-4	24-4	8-4	31-3	20-4	5-4	27-3	16-4	1-4


dia de la Pascua año
variables de programa a b c d e M N
si año > 1582 y año < 1700
fijar M a 22
fijar N a 2
si no
si año > 1699 y año < 1800
fijar M a 23
fijar N a 3
si no
si año > 1799 y año < 1900
fijar M a 23
fijar N a 4
si no
si año > 1899 y año < 2100
fijar M a 24
fijar N a 5
si no
si año > 2099 y año < 2200
fijar M a 24
fijar N a 6
si no
si año > 2199 y año < 2300
fijar M a 25
fijar N a 0
fijar a a año mod 19
fijar b a año mod 4
fijar c a año mod 7
fijar d a 19 * a + M mod 30
fijar e a 2 * b + 4 * c + 6 * d + N mod 7
si d + e < 10
informar unir d + e + 22 de marzo
si no
si d + e - 9 = 26
informar 17 de abril
si no
si d + e = 34 y a > 10
informar 18 de abril
si no
informar unir d + e - 9 de abril


dia de Pascua de Butcher año
variables de programa a b c d e f g h i k l m n
mes dia
fijar a a año mod 19
fijar b a año coc 100
fijar c a año mod 100
fijar d a b coc 4
fijar e a b mod 4
fijar f a b + 8 coc 25
fijar g a b - f + 1 coc 3
fijar h a 19 * a + b - d - g + 15 mod 30
fijar i a c coc 4
fijar k a c mod 4
fijar l a 32 + 2 * e + 2 * i - h - k mod 7
fijar m a a + 11 * h + 22 * l coc 451
fijar n a 114 + h + l - 7 * m
fijar mes a n coc 31
fijar dia a 1 + n mod 31
informar unir unir dia - mes

P34	Generador de apuestas aleatorias
	El organismo nacional de Loterías y Apuestas del Estado gestiona juegos de apuestas basados en el azar. Tres de los más populares son la Lotería Nacional, la Quiniela y la Primitiva. En la lotería nacional las apuestas se concretan en series de billetes de 10 décimos que apuestan a un número de 5 cifras entre el 00000 y el 99999. En la Quiniela el jugador pronostica el resultado de 15 partidos de fútbol asigna 1 a la victoria del equipo de casa, 2 a la victoria del equipo visitante y X al empate. En la Primitiva una apuesta básica es un sexteto de números naturales elegidos del 1 al 49.
	Escribir un programa que una vez elegido un juego (1. Loteria Nacional, 2. Quiniela, 3. Primitiva) genere una apuesta aleatoria para dicho juego: un número del 00000 al 99999, 15 signos 1X2 ó 6 números distintos del 1 al 49. V2. Introduce probabilidades diferentes para el 1X2, por ejemplo: P(1)=0.5, P(2)=0.3, P(X)=0.2.
	 En la combinación de la Primitiva debes controlar que no se repita algún número ya extraído.

P35	Simulación del lanzamiento de dados
	Existen muchos juegos de mesa que basan la jugada en una tirada de 1 o más dados. En este proyecto vamos a utilizar los disfraces de un objeto para presentar el resultado. Un disfraz es una imagen. Cada objeto puede tener asignados varios disfraces que cambian su aspecto. En nuestro caso el objeto será un dado y tendrá 6 disfraces, uno para cada resultado distinto.
	Simular el lanzamiento de un dado. V2. Simular el lanzamiento de dos dados.
	El resultado del lanzamiento lo generaremos con un número aleatorio entre 1 y 6 que almacenaremos en una variable. Según el valor de la variable presentaremos el disfraz que corresponda usando la instrucción  del menú  .

P36 Códigos ASCII y UNICODE

ASCII (*American Standard Code for Information Interchange*) es un código de caracteres basado en el alfabeto inglés, creado en 1963 por el el Instituto Estadounidense de Estándares Nacionales (ANSI) como una evolución de los conjuntos de códigos utilizados en telegrafía.

En 1967 se incluyeron las minúsculas.

Utiliza 7 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional de paridad que se usaba para detectar errores en la transmisión.

Casi todos los sistemas informáticos actuales utilizan el código ASCII o una extensión compatible para representar textos y para el control de dispositivos que manejan texto como el teclado.

El código ASCII reserva los códigos 0 a 31 para caracteres de control de dispositivos (como impresoras). Por ejemplo, el carácter 10 representa la función "nueva línea" (line feed), que hace que una impresora avance el papel, y el carácter 27 representa la tecla "escape" que a menudo se encuentra en la esquina superior izquierda de los teclados comunes. Los códigos del 48 al 57 corresponden con los cifras del 0 al 9, los códigos del 65 al 90 y del 97 al 122 corresponden con el alfabeto inglés en mayúsculas y minúsculas, respectivamente. El resto de códigos son caracteres imprimibles incluyendo el espaciador (código 32).

El código ASCII es el alfabeto básico de ordenación de cadenas de texto almacenadas en un ordenador.

b_7 →				0	0	0	0	1	1	1	1
b_6 →				0	0	1	1	0	0	1	1
b_5 →				0	1	0	1	0	1	0	1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Unicode es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas, además de textos clásicos de lenguas muertas. El término proviene de los tres objetivos que persigue: universalidad, uniformidad y unicidad. Unicode incluye sistemas de escritura modernos como: árabe, braille, copto, cirílico, griego, hanja coreano, hanzi chino y kanji japonés, hebreo y latino; escrituras históricas extintas como cuneiforme, griego antiguo, lineal B micénico, fenicio y rúnico. Entre los caracteres no alfabéticos incluidos se encuentran símbolos musicales y matemáticos, fichas de juegos como el dominó, flechas, iconos etc. Además incluye los signos diacríticos como caracteres independientes que pueden ser combinados con otros caracteres y dispone de versiones predefinidas de la mayoría de letras con símbolos diacríticos en uso en la actualidad, como las vocales acentuadas del español.

La próxima versión Unicode 6.3.0 estará disponible en septiembre de 2013. La web del consorcio donde se pueden encontrar los más de 100.000 caracteres disponibles es http://www.unicode.org.

BYOB dispone de instrucciones para manejar caracteres ASCII: 

y 

. Su versión 4.0 conocida como SNAP maneja además código UNICODE.

OBJETIVO

Diseñar un programa traductor de texto a ASCII, es decir, que informe los códigos numéricos que se almacenarán en el ordenador.
V2. Presentar la tabla de códigos ASCII y su equivalencia.

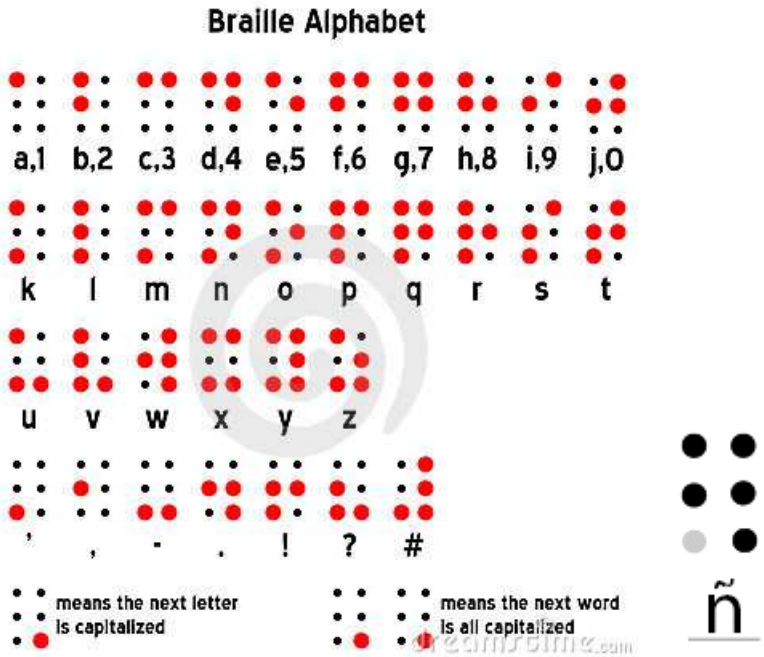
Solicitar una cadena de texto. Procesar letra a letra y convertir con la instrucción 

.

P37	Traductor de alfabeto Braille

El braille es un sistema de lectura y escritura táctil pensado para personas ciegas, fue ideado por el francés Louis Braille a mediados del siglo XIX. El braille resulta interesante también por tratarse de un sistema de numeración binario que precedió a la aparición de la informática. Braille ideó un sistema de 6 puntos en relieve, organizados como una matriz de tres filas por dos columnas, que convencionalmente se numeran de arriba a abajo y de izquierda a derecha, con el que representar las letras, los signos de puntuación, los números, la grafía científica, los símbolos matemáticos, la música, etc. La presencia o ausencia de puntos permite la codificación de los símbolos.

Se obtienen 64 combinaciones diferentes que resultan claramente insuficientes. Por ello se utilizan signos diferenciadores especiales que, antepuestos a una combinación de puntos, convierten una letra en mayúscula, bastardilla, número o nota musical. En el braille español, los códigos de las letras minúsculas, la mayoría de los signos de puntuación, algunos caracteres especiales y algunas palabras se codifican directamente con una celda, pero las mayúsculas y números son representados además con otro símbolo como prefijo.



El alfabeto Braille está representado en los 256 caracteres UNICODE U2800 a U28FF (todas las posibilidades de 2 columnas de 4 puntos).

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

OBJETIVO	Diseñar un programa traductor de texto a Braille.
	☞ Solicitar una cadena de texto y comprobar que en ella solo figuran caracteres del alfabeto básico.

P38	Algoritmo de exponenciación rápida

La exponenciación modular se utiliza muy frecuentemente en criptografía. Vamos a desarrollar un algoritmo eficiente de cálculo de la función exponencial a^b siendo a y b números naturales cualesquiera para, a continuación, resolver el cálculo a^b mod c.

La clave está en expresar el exponente en código binario.

Tomemos la representación binaria de b:

$$b = 2^0 b_0 + 2^1 b_1 + 2^2 b_2 + \ldots + 2^n b_n$$

Expresemos la potencia que vamos a calcular en función de dicha representación:

$$a^b = a^{2^0 b_0 + 2^1 b_1 + 2^2 b_2 + \ldots + 2^n b_n} = \prod_{i=0}^{n} a^{2^i b_i}$$

recordemos que los b_i sólo pueden valer 0 ó 1, por tanto para calcular a^b sólo hemos de multiplicar los a^{2^i} correspondientes a los dígitos binarios de b que valgan 1.

Nótese, además, que $a^{2^i} = (a^{2^{i-1}})^2$, por lo que, partiendo de a, podemos calcular el siguiente valor de esta serie elevando al cuadrado el anterior.

Para el cálculo de a^b mod c haremos uso de la propiedad del cálculo de residuos o aritmética modular:

el residuo del producto es el producto de los residuos.

OBJETIVO

Programar el cálculo de la función exponencial a^b siendo a y b números naturales cualesquiera.

V2. Programar a^b mod c siendo a, b y c números naturales cualesquiera.

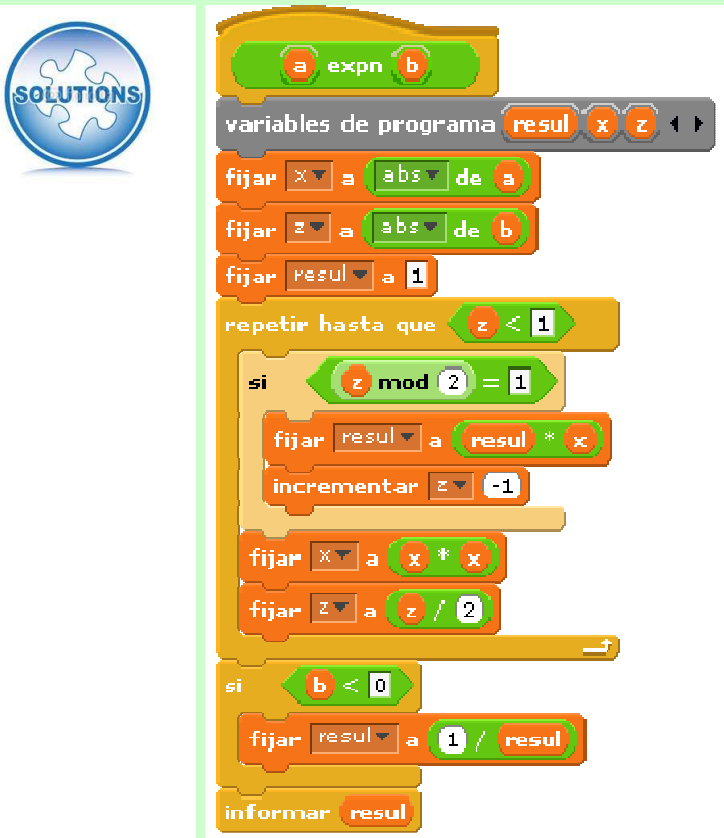
Como pista mostramos la solución en C++.

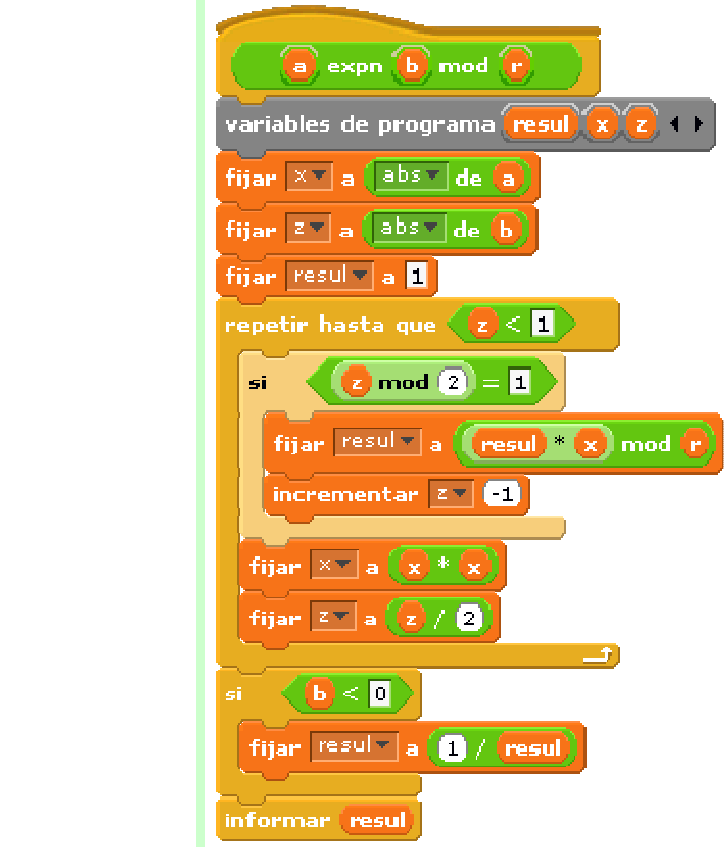
Debes codificar en BYOB esta subrutina como una función con dos parámetros de entrada a y b que deja el resultado en la variable resul.

Como BYOB no utiliza definición de tipos debes asegurarte de que z/2 siempre es entero.

V2. ¿Qué tienes que cambiar además de añadir el parámetro de entrada c?

```
int exp_rapida(int a, int b)
    { int z,x,resul;

      z=b;
      x=a;
      resul=1;
      while (z>0)
        { if (z%2==1)
            resul=resul*x;
          x=x*x;
          z=z/2;

        }
      return(resul);
    }
```


a expn b
variables de programa resul x z
fijar x a abs de a
fijar z a abs de b
fijar resul a 1
repetir hasta que z < 1
si z mod 2 = 1
fijar resul a resul * x
incrementar z -1
fijar x a x * x
fijar z a z / 2
si b < 0
fijar resul a 1 / resul
informar resul


a expn b mod r
variables de programa resul x z
fijar x a abs de a
fijar z a abs de b
fijar resul a 1
repetir hasta que z < 1
si z mod 2 = 1
fijar resul a resul * x mod r
incrementar z -1
fijar x a x * x
fijar z a z / 2
si b < 0
fijar resul a 1 / resul
informar resul

P39	Algoritmo de encriptación RSA

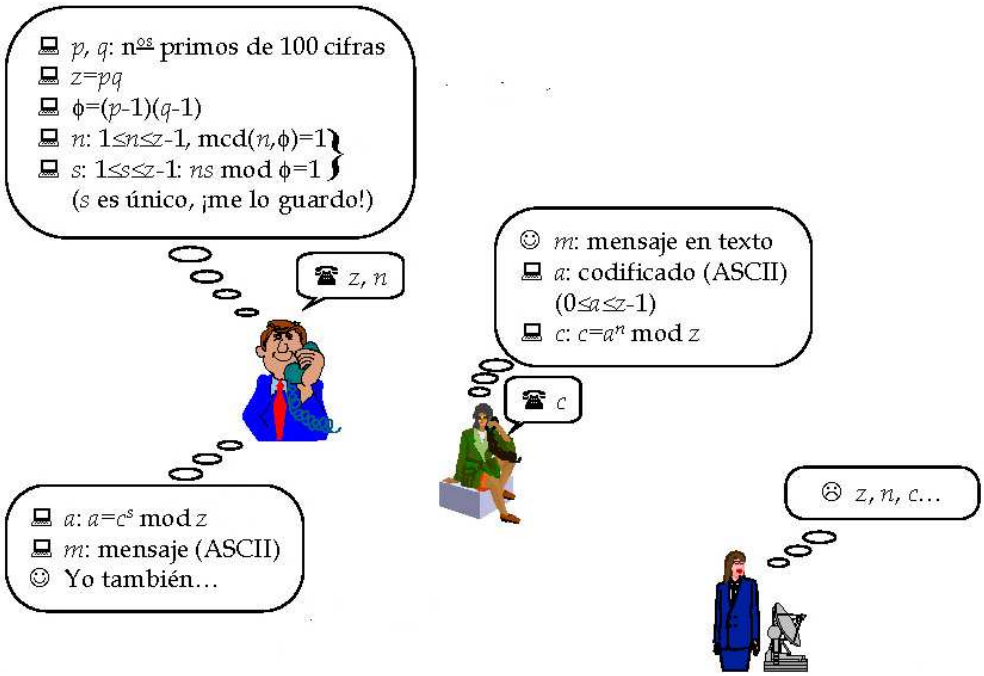
El algoritmo de clave pública RSA fue ideado en 1978 por Ron Rivest, Adi Shamir y Leonard Adleman, del Instituto Tecnológico de Massachusetts (MIT) y es el sistema criptográfico asimétrico más conocido y usado.

La fortaleza del algoritmo RSA se basa en el hecho matemático de la dificultad de factorizar números muy grandes. Basado en la exponenciación modular el sistema RSA crea sus claves de la siguiente forma:

Se buscan dos números primos muy grandes (de entre 100 y 300 dígitos): p y q.
Se obtienen los números z = pq y ø = (p-1)(q-1).
Se busca un número n entre 1 y z-1 de forma que n y ø sean primos entre sí. Esta será la parte pública de la clave.
Se calcula el único s de forma que ns = 1 mod ø, es decir 1 es el resto de la división entera de ns entre ø. Esta será la parte privada de la clave.
Las pistas sobre los números p, q y ø se destruyen.
Dado un carácter a se cifra haciendo $c = a^n \bmod z$
Dado el carácter encriptado c se descifra y se averigua a haciendo $a = c^s \bmod z$.

El cálculo de estas claves se realiza en secreto en la máquina en la que se va a guardar la parte privada de la clave. Se hacen públicos los números z y n, necesarios para alimentar el algoritmo. Es aconsejable el uso de claves de no menos de 1024 bits ya que claves de hasta 512 bits han sido averiguadas, aunque se necesitaron más de 5 meses y casi 300 ordenadores trabajando juntos para hacerlo.

RSA basa su seguridad en que aunque la exponenciación modular es fácil, su operación inversa, la extracción de raíces de módulo ø no es factible, a menos que se conozca la factorización de n, parte privada de la clave.



Esquema de funcionamiento del algoritmo de encriptación RSA.
Un posible intruso puede llegar a conocer z, n y c (cifrado) pero no sabrá reconocer el mensaje original sin cifrar a.

Veamos un ejemplo usando primos muy pequeños.

Sea p=3 y q=11. Entonces z=3·11=33 y ∅=2·10=20

Como m.c.d. (3,20)=1 hacemos n=3 y hallamos s=7 pues 3·7=21=1 mod 20. Codifiquemos el mensaje "CASA".

En ASCII será 67 65 83 65. Restando 64 queda 03 01 19 01.

Letra	a	a^3	$c = a^3 \bmod 33$	c^7	$a=c^7 \bmod 33$
C	03	27	27	10460353203	3
A	01	1	1	1	1
S	19	6859	28	13492928512	19
A	01	1	1	1	1

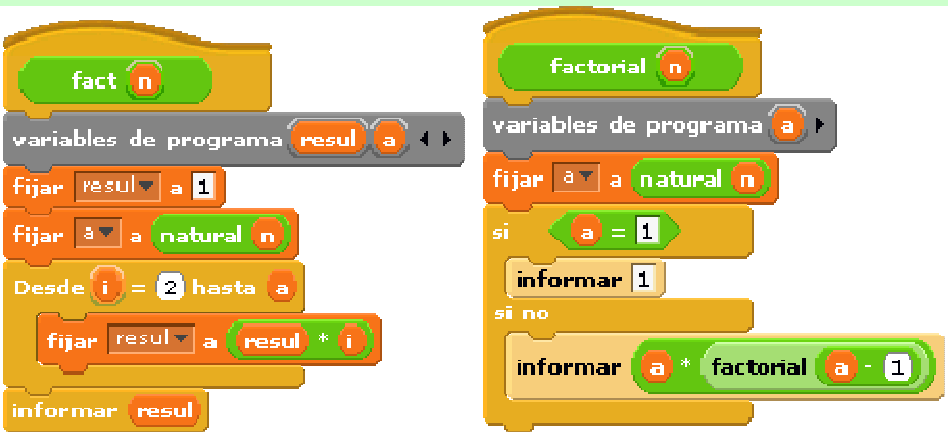
Se transmite 27 01 28 01 en lugar de 03 01 19 01 y solo conociendo que s=7 se puede descifrar.

OBJETIVO

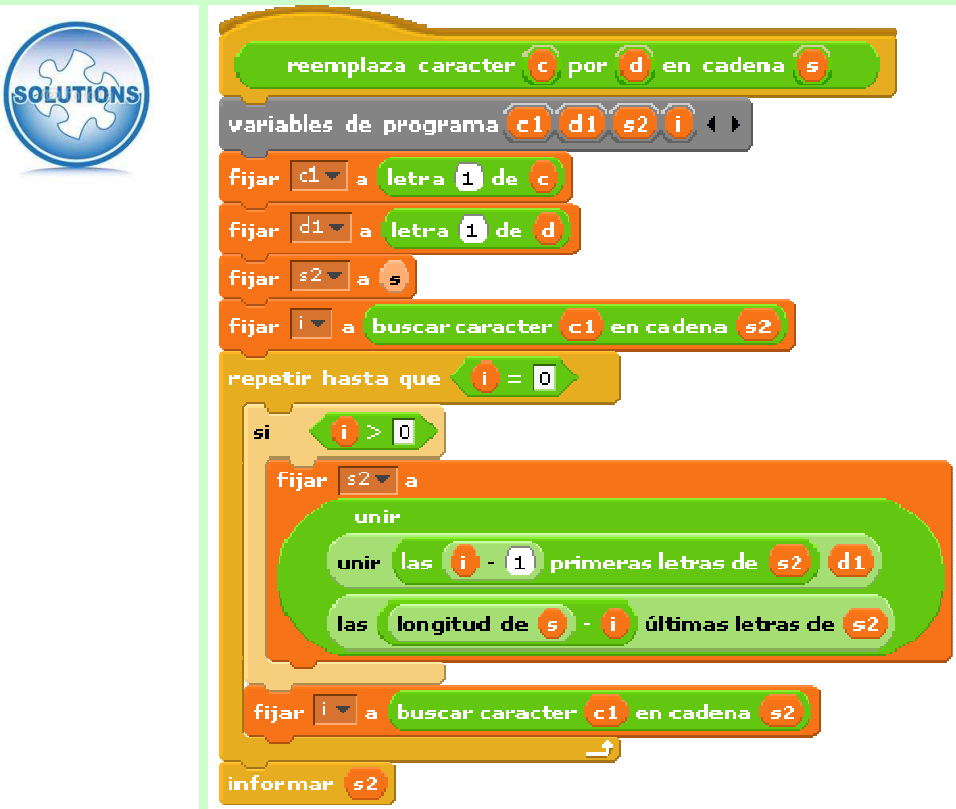
Programar la encriptación de cadenas de texto usando el algoritmo RSA para (p, q)=(3, 11) y (n, s)=(3, 7). Restar 64 para convertir el código ASCII de la A en 1 y de esa manera poder reconocer la A y los 32 caracteres siguientes.

☞ Usar las variables p, q, n y s para, en teoría, poder modificar los valores en cualquier momento.

☞ Utiliza el algoritmo de exponenciación rápido para cifrar y descifrar (cálculo de $c=a^n$ *mod z* y $a=c^s$ *mod z*)

P40	Factorial de un número
	El factorial de un número natural es el producto de todos los números naturales menores o iguales que él. Se escribe n! = n·(n-1)·(n-2)· … ·3·2·1. Ejemplo: 6! = 6·5·4·3·2·1 = 120. El factorial se utiliza en Combinatoria. Hemos visto cómo calcularlo en el ejemplo de recursividad. Ahora lo haremos sin utilizar la recursividad como una función normal a la que se le envía un parámetro de entrada n y devuelve n!. Se llama **primo factorial** al número anterior o posterior de un factorial si es primo. El teorema de Wilson utiliza el factorial para caracterizar los números primos: *n es primo si solo si n divide a (n-1)!+1*
OBJETIVO	Escribe una función que calcule el factorial de un entero positivo. Invócala para que Calculín responda a la entrada de un valor de n solicitado. V2. Escribe en una lista los primos factoriales para valores de n inferiores a 100. Programa una función Wilson(n) que nos corrobore que el número es primo.
	Pseudocódigo **función** Wilson(n) **si** n divide a (n-1)!+1 **devolver** cierto **si_no devolver** falso
SOLUTIONS	

P41	Hablar con las vocales
	Un tradicional juego de niños consiste en hablar con las vocales para no hacerse entender. Existe incluso una canción popular cuya letra dice: *Cuando Fernando séptimo usaba paletó* Con la a: *Caanda Farnanda sáptama asaba palatá* Con la e: *Ceende Fernende sépteme esebe peleté* Con la i: *Ciindi Firnindi síptimi isibi pilití* Con la o: *Coondo Fornondo sóptomo osobo polotó* Con la u: *Cuundu Furnundu súptumu usubu pulutú* **paletó.** (Del fr. *paletot*). **1.** m. Gabán de paño grueso, largo y entallado, pero sin faldas como el levitón. Vamos a elaborar una función que sustituya cualquier vocal {A, E, I, O, U, a, e, i, o, u, á, é, í, ó, ú} por una vocal dada.
	Escribe una función booleana que averigüe si una letra es una vocal. Solicita una vocal para sustituir el resto de vocales por ella. Solicita una cadena de texto y conviertela al habla con esa vocal.
	Debes averiguar el código ASCII de todas las vocales mayúsculas, minúsculas y acentuadas. Procesa la cadena de texto de entrada letra a letra.

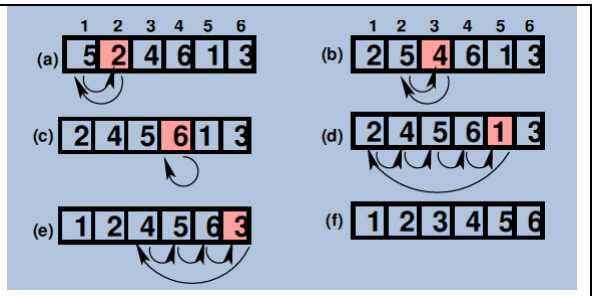

reemplaza caracter c por d en cadena s
variables de programa c1 d1 s2 i
fijar c1 a letra 1 de c
fijar d1 a letra 1 de d
fijar s2 a s
fijar i a buscar caracter c1 en cadena s2
repetir hasta que i = 0
si i > 0
fijar s2 a
unir
unir las i - 1 primeras letras de s2 d1
las longitud de s - i últimas letras de s2
fijar i a buscar caracter c1 en cadena s2
informar s2


buscar caracter c en cadena s
variables de programa i resul
fijar resul a 0
Desde i = 1 hasta longitud de s
si letra i de s = c
fijar resul a i
informar resul


las n primeras letras de s
variables de programa i resul
fijar i a 1
fijar resul a
repetir abs de n
fijar resul a unir resul letra i de s
incrementar i 1
informar resul

P42 Ordenar varios números

Existen numerosos algoritmos de ordenación. Uno de los más sencillos, por su similitud a lo que habitualmente hacemos para ordenar una baza de cartas, es el **ordenamiento por inserción** (***insertion sort*** en inglés).

Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado.	 Después, cuando ya hay *k* elementos ordenados de menor a mayor, se toma el elemento siguiente (*k+1*) y se compara con todos los elementos ordenados. Nos detendremos si se encuentra un elemento mayor siendo este el punto donde se *inserta* el elemento *k+1*. Si no nos detenemos el elemento *k+1* quedará como el mayor del nuevo conjunto ordenado.

Ejemplo de ordenamiento por inserción. Partimos de la lista {5,2,4,6,1,3} y se realizan los cambios que indican las flechas.

Otro algoritmo de ordenación muy sencillo de programar es el llamado **procedimiento de la burbuja**. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

OBJETIVO

Escribe un procedimiento que ordene los números de una lista mediante el algoritmo de ordenamiento por inserción. Escribe un programa que solicite varios números (0, para acabar), los añada a una lista, la ordene usando el procedimiento y la muestre en pantalla. V2. Programa el procedimiento de la burbuja.

Ambos algoritmos ejecutan el ordenamiento de n números en un *tiempo cuadrático*, es decir, proporcional a n^2.

Un pseudocódigo en inglés del procedimiento de ordenamiento por inserción hallado en Internet es:

```
Insertion-Sort (A)
1   for j ← 2 to length [A]
2       do key ← A[j]
           ▷ Insertar A[j] en la secuencia ordenada A[1..j − 1]
3          i ← j − 1
4          while i > 0 and A[i] > key
5                do A[i + 1] ← A[i]
6                   i ← i − 1
7          A[i + 1] ← key
```

El pseudocódigo en español del procedimiento de la burbuja es:

```
algoritmo burbuja( A : lista de n elementos, de A[1] a A[n])
  para i desde 1 hasta n-1 hacer: //las n-1 pasadas
       para j desde 1 hasta n-i hacer: //el recorrido
         si A[j] > A[j+1] entonces //Si no están en orden
            aux  ← A[j] //Se intercambian
            A[j] ← A[j+1]
            A[j+1] ← aux
       fin para
  fin para
fin algoritmo
```


```
coloca aux en lista ordenada lista
variables de programa tam resul i sw
fijar tam a longitud de lista
fijar resul a lista
borrar todo de resul
fijar i a 1
fijar sw a 0
repetir hasta que i > tam
    si aux < elemento i de lista y sw = 0
        añade aux a resul
        añade elemento i de lista a resul
        fijar sw a 1
    si no
        añade elemento i de lista a resul
    incrementar i 1
si sw = 0
    añade aux a resul
informar resul
```

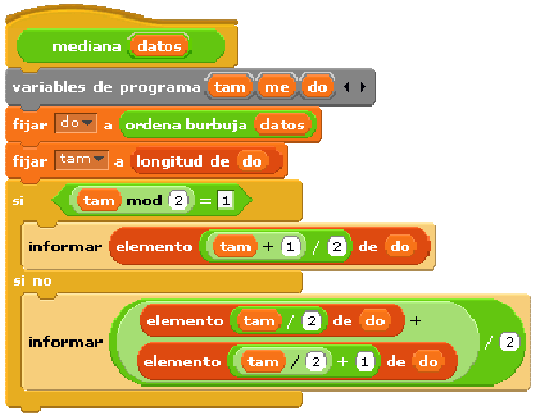
```
ordena burbuja lista
variables de programa tam resul i j aux
fijar tam a longitud de lista
fijar resul a lista
fijar resul a lista
fijar i a 1
repetir hasta que i = tam
    fijar j a 1
    repetir hasta que j = tam - i + 1
        si elemento j de resul > elemento j + 1 de resul
            fijar aux a elemento j de resul
            reemplazar elemento j de resul con elemento j + 1 de resul
            reemplazar elemento j + 1 de resul con aux
        incrementar j 1
    incrementar i 1
informar resul
```

```
ordena insercion lista
variables de programa tam resul k j aux
fijar tam a longitud de lista
fijar resul a lista
borrar todo de resul
añade elemento 1 de lista a resul
fijar j a 2
repetir hasta que j > tam
    fijar aux a elemento j de lista
    fijar resul a coloca aux en lista ordenada resul
    incrementar j 1
informar resul
```

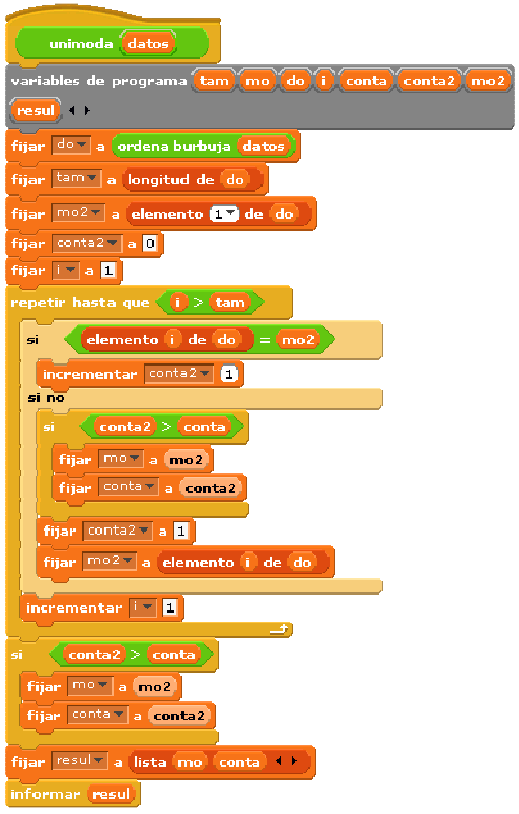
P43	Cálculo de parámetros estadísticos
	La información estadística de un estudio se puede resumir con unos pocos parámetros que recogen los valores centrales y la posición y dispersión de los datos. Entre las medidas de centralización se encuentran la media aritmética, la mediana y la moda. La **media aritmética** es el promedio de los valores que toma la variable y se define como el cociente de la suma de todos ellos entre el número de datos. La **mediana** es el valor central de los datos ordenados. Si el número de datos es par se promedian los dos valores centrales. La **moda** es el valor que más se repite. La medida de posición por excelencia es el percentil. Entre las medidas de dispersión se encuentran el recorrido, la varianza, la desviación típica y el coeficiente de variación. El **percentil i** de unos valores estadísticos ordenados es aquel valor que acumula el i% de los datos, dejando por encima el (100-i)% de los datos. El **recorrido** es la diferencia entre los valores mayor y menor de una distribución. La **varianza** es la media aritmética de los cuadrados de las desviaciones de los datos respecto a su media aritmética. La **desviación típica** es la raíz cuadrada positiva de la varianza. El **coeficiente de variación** es la razón entre la desviación típica y la media aritmética.
	Elabora un programa que solicite datos (F, para finalizar la entrada) y calcule todos los parámetros arriba definidos.
	Ordena los datos usando el proyecto P42 (procedimiento de inserción o burbuja). La fórmula más sencilla de la varianza es $\frac{\sum_{i=1}^{n} x_i^2}{n} - \overline{x}^2$ siendo n el número de datos, x_i los datos y $\overline{x}$ la media aritmética.

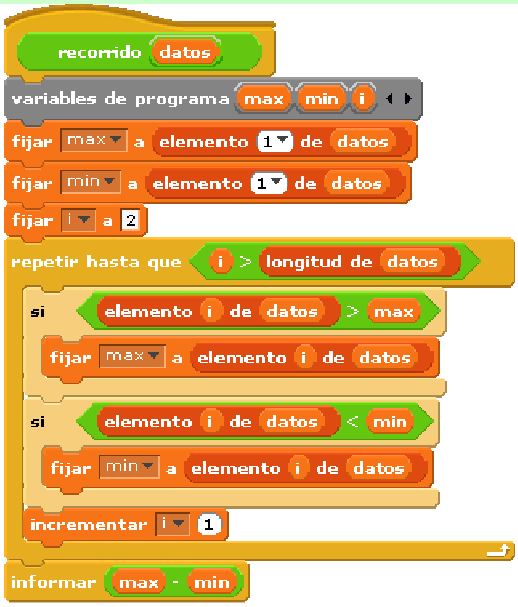

SOLUTIONS

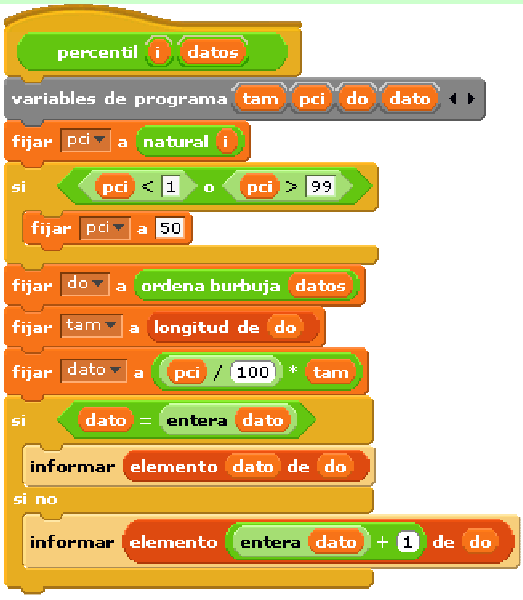

media datos
variables de programa tam mu do i
fijar do a lista
borrar todo de do
fijar do a ordena burbuja datos
fijar tam a longitud de do
fijar mu a 0
fijar i a 1
repetir hasta que i > tam
incrementar mu elemento i de do
incrementar i 1
informar mu / tam


mediana datos
variables de programa tam me do
fijar do a ordena burbuja datos
fijar tam a longitud de do
si tam mod 2 = 1
informar elemento tam + 1 / 2 de do
si no
informar elemento tam / 2 de do + elemento tam / 2 + 1 de do / 2


moda datos
variables de programa mo do conta i conta2 mo2 resul
fijar do a ordena burbuja datos
fijar mo a elemento 1 de unimoda do
fijar conta a elemento 2 de unimoda do
fijar conta2 a 0
fijar mo2 a elemento 1 de do
fijar resul a lista
fijar i a 1
repetir hasta que i > longitud de do
si elemento i de do = mo2
incrementar conta2 1
si no
si conta2 = conta
añade mo2 a resul
fijar conta2 a 1
fijar mo2 a elemento i de do
incrementar i 1
si conta2 = conta
añade mo2 a resul
añade conta a resul
informar resul


unimoda datos
variables de programa tam mo do i conta conta2 mo2
resul
fijar do a ordena burbuja datos
fijar tam a longitud de do
fijar mo2 a elemento 1 de do
fijar conta2 a 0
fijar i a 1
repetir hasta que i > tam
si elemento i de do = mo2
incrementar conta2 1
si no
si conta2 > conta
fijar mo a mo2
fijar conta a conta2
fijar conta2 a 1
fijar mo2 a elemento i de do
incrementar i 1
si conta2 > conta
fijar mo a mo2
fijar conta a conta2
fijar resul a lista mo conta
informar resul


recorrido datos
variables de programa max min i
fijar max a elemento 1 de datos
fijar min a elemento 1 de datos
fijar i a 2
repetir hasta que i > longitud de datos
si elemento i de datos > max
fijar max a elemento i de datos
si elemento i de datos < min
fijar min a elemento i de datos
incrementar i 1
informar max - min


percentil i datos
variables de programa tam pci do dato
fijar pci a natural i
si pci < 1 o pci > 99
fijar pci a 50
fijar do a ordena burbuja datos
fijar tam a longitud de do
fijar dato a pci / 100 * tam
si dato = entera dato
informar elemento dato de do
si no
informar elemento entera dato + 1 de do


varianza datos
variables de programa mu do do2
fijar do a lista
fijar do a ordena burbuja datos
fijar mu a media datos
fijar do2 a lista
fijar do2 a do
Desde i = 1 hasta longitud de do2
reemplazar elemento i de do2 con
elemento i de do2 * elemento i de do2
informar media do2 - mu * mu


desviacion tipica datos
informar raiz cuadrada de varianza datos

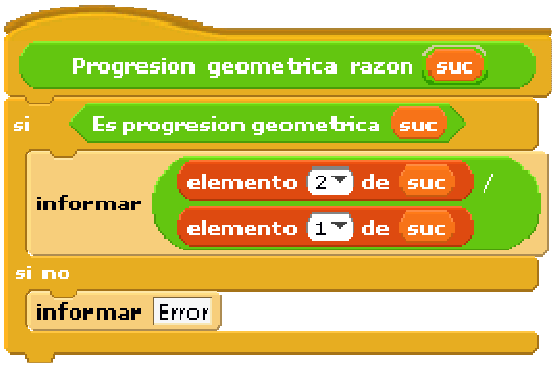

coeficiente de variacion datos
variables de programa a
fijar a a copia de datos
informar desviacion tipica datos / media a

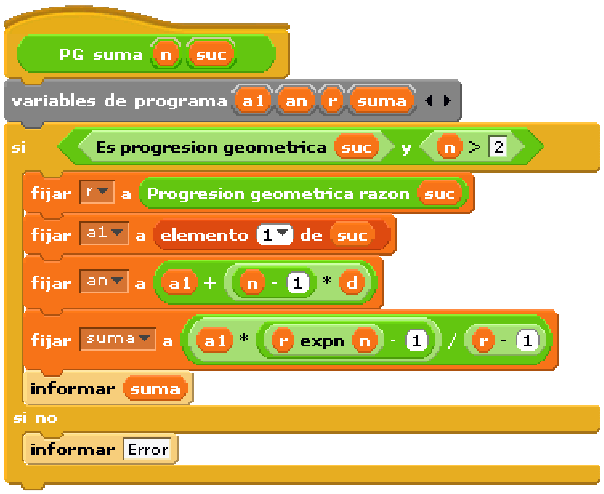
P44	Detector de progresiones aritméticas
	Una **sucesión** es una correspondencia que a cada número natural n asocia un número real a_n. Una sucesión $\{a_n\}$ es una **progresión aritmética** si cada término se obtiene del anterior sumando una cantidad fija d, la diferencia. El **término general** de una progresión aritmética es $a_n = a_1 + (n-1)\cdot d$. La **suma** de los primeros n términos de una progresión aritmética es $S_n = \frac{a_1+a_n}{2}n$. Para detectar una progresión aritmética la resta de cualesquiera dos términos consecutivos debe ser constante.
OBJETIVO	Elabora un programa que solicite datos (F, para finalizar la entrada) y detecte si se trata de una progresión aritmética calculando su diferencia, el término general y la suma de los n primeros términos.
	☞ Aloja los datos en una lista.

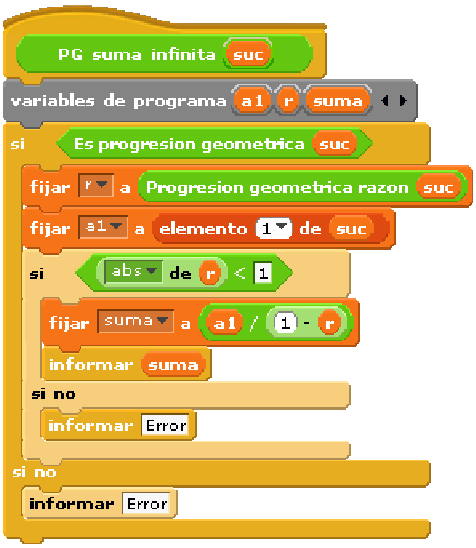

Es progresion aritmetica suc
variables de programa d tam i
fijar tam a longitud de suc
si tam < 2
informar falso
fijar d a elemento 2 de suc - elemento 1 de suc
fijar i a 3
repetir hasta que i > tam
si no d = elemento i de suc - elemento i - 1 de suc
informar falso
incrementar i 1
informar cierto
Progresion aritmetica diferencia suc
si Es progresion aritmetica suc
informar elemento 2 de suc - elemento 1 de suc
si no
informar Error
PA suma n suc
variables de programa a1 an d suma
si Es progresion aritmetica suc y n > 2
fijar d a Progresion aritmetica diferencia suc
fijar a1 a elemento 1 de suc
fijar an a a1 + n - 1 * d
fijar suma a a1 + an / 2 * n
informar suma
si no
informar Error

P45	Detector de progresiones geométricas
	Una sucesión $\{a_n\}$ es una **progresión aritmética** si cada término se obtiene del anterior multiplicándolo por una cantidad fija *r*, la razón. El **término general** de una progresión geométrica es $a_n = a_1 \cdot r^{n-1}$. La **suma** de los primeros n términos de una progresión geométrica es $S_n = \frac{a_n \cdot r - a_1}{r - 1}$. Si |r|<1 la suma infinita de términos es $S_\infty = \frac{a_1}{1 - r}$. Para detectar una progresión geométrica el cociente de cualesquiera dos términos consecutivos debe ser constante.
OBJETIVO	Elabora un programa que solicite datos (F, para finalizar la entrada) y detecte si se trata de una progresión geométrica calculando la razón, su término general y la suma de los n primeros términos.
	☞ Aloja los datos en una lista.


Es progresion geometrica suc
variables de programa r tam i
fijar tam a longitud de suc
si tam < 2
informar falso
fijar r a elemento 2 de suc / elemento 1 de suc
fijar i a 3
repetir hasta que i > tam
si no r = elemento i de suc / elemento i - 1 de suc
informar falso
incrementar i 1
informar cierto


Progresion geometrica razon suc
si Es progresion geometrica suc
informar elemento 2 de suc / elemento 1 de suc
si no
informar Error


PG suma n suc
variables de programa a1 an r suma
si Es progresion geometrica suc y n > 2
fijar r a Progresion geometrica razon suc
fijar a1 a elemento 1 de suc
fijar an a a1 + n - 1 * d
fijar suma a a1 * r expn n - 1 / r - 1
informar suma
si no
informar Error


PG suma infinita suc
variables de programa a1 r suma
si Es progresion geometrica suc
fijar r a Progresion geometrica razon suc
fijar a1 a elemento 1 de suc
si abs de r < 1
fijar suma a a1 / 1 - r
informar suma
si no
informar Error
si no
informar Error

P46	Sucesión de Fibonacci

Una sucesión $\{a_n\}$ es de Fibonacci si a partir del tercer término cada término se obtiene como suma de los dos anteriores, es decir, $a_n=a_{n-1}+a_{n-2}$ cuando n>2.

Si hacemos $a_1=1$, $a_2=1$ obtenemos

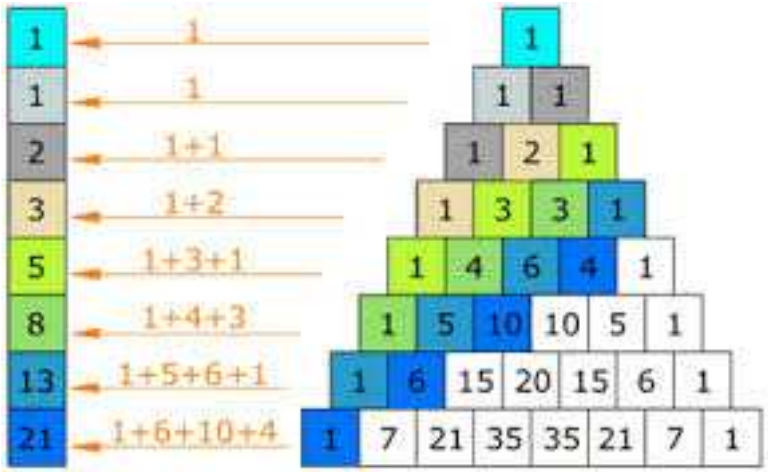
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, …}.

El cociente de dos términos consecutivos tiende a la razón áurea $\phi = \frac{1+\sqrt{5}}{2}$ un número irracional con curiosas propiedades (sus infinitas cifras decimales son las mismas que las de su inverso $\frac{1}{\phi}$ y las de su "anterior" $\phi - 1$).

La **fórmula de Binet** $a_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$ proporciona los términos de la sucesión de Fibonacci.

Alguna propiedades de la sucesión son:

- Cualquier número natural se puede escribir mediante la suma de un número limitado de términos de la sucesión de Fibonacci, cada uno de ellos distinto a los demás.
- El máximo común divisor de dos números de Fibonacci es otro número de Fibonacci.
- Los números de Fibonacci aparecen al sumar las diagonales del triángulo de Pascal.

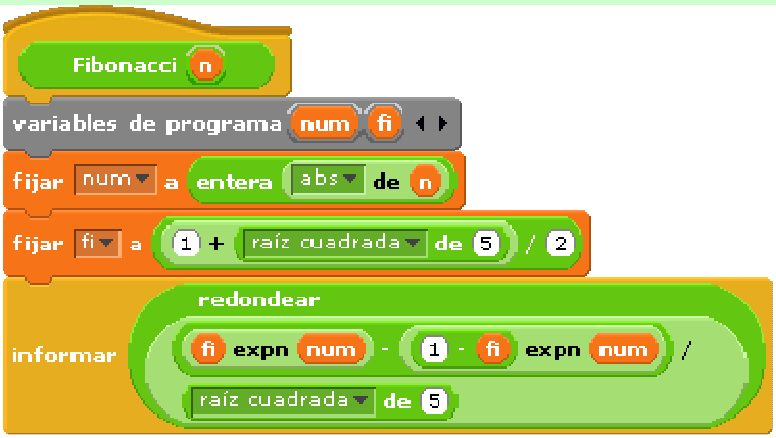


- La suma de diez números de Fibonacci consecutivos es once veces el séptimo número de la serie.
- La suma de los cuadrados de los primeros n números de Fibonacci es igual al producto del enésimo número y del enésimo más uno.
- El cuadrado de un número de Fibonacci y el producto del anterior y posterior de la secuencia difieren en una unidad.

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

Elabora un programa que construya la sucesión de Fibonacci y calcule la razón áurea.
Comprueba la fórmula de Binet redondeando.
V2. Verifica la propiedad de la suma de diez términos consecutivos. Solicita un ordinal que será el primero de los diez, calcula y muestra los diez términos y verifica la ecuación la suma es el séptimo término por once.

Aloja los términos en una lista.

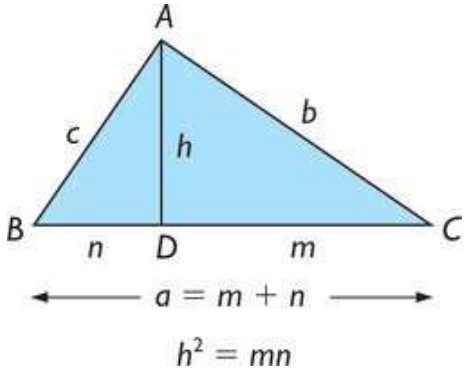


P47	Resolutor de triángulos rectángulos

Resolver un triángulo es hallar los ángulos y lados desconocidos a partir de los datos conocidos.
Para resolver un triángulo rectángulo basta conocer dos lados o un lado y un ángulo agudo.
Para ello nos apoyamos en tres resultados:

Teorema de Pitágoras. Un triángulo es rectángulo si y sólo si la hipotenusa al cuadrado es igual a la suma de los catetos al cuadrado.

ABC es rectángulo en A si y solo si $a^2 = b^2+c^2$



Teorema de la altura. En un triángulo rectángulo la altura es media proporcional de los segmentos en que divide a la base. $h^2 = m \cdot n$

Teorema del cateto. En un triángulo rectángulo el cateto es media proporcional de la hipotenusa y la proyección del cateto sobre la hipotenusa. $b^2 = m \cdot a$ y $c^2 = n \cdot a$

OBJETIVO

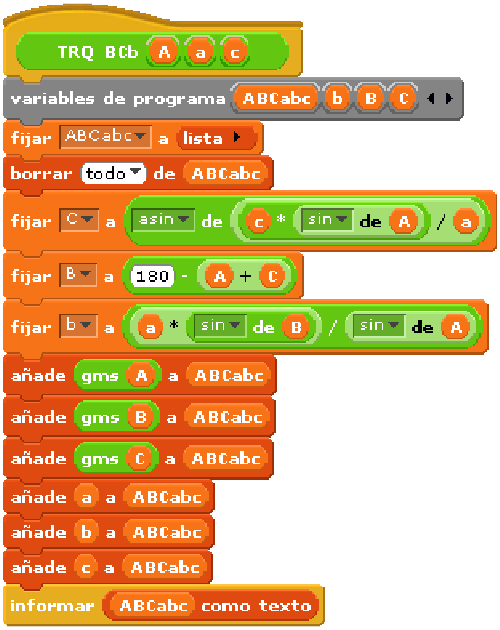
Elabora un programa que

1) Dados tres posibles lados verifique si forman un triángulo rectángulo.
2) Dados dos catetos calcule la hipotenusa y los tres ángulos
3) Dados un cateto y la hipotenusa calcule el otro cateto y los tres ángulos
4) Dada la hipotenusa y un ángulo agudo calcule los catetos y el otro ángulo.
5) Dado un cateto y un ángulo agudo calcule el otro cateto y la hipotenusa y el otro ángulo.

Utiliza los teoremas y programa las fórmulas para cada caso. Por ejemplo en la opción 1) basta detectar el lado mayor y comprobar si $a^2 = b^2+c^2$. Define una función que reciba los tres lados y devuelva la hipotenusa. Esta función puede apoyarse en P10 Ser o no ser triángulo o bien en otra booleana EsTrianguloRecto.

Recuerda que en el triángulo rectángulo de la figura A = 90º, sen B = b/a y sen C = c/a.

NOTA: Las letras en el nombre de las funciones son los parámetros a calcular; los que faltan son los datos (A=90).



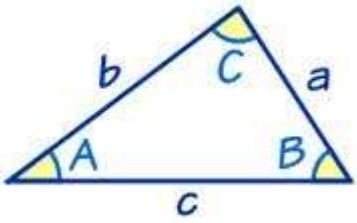
P48 Resolutor de triángulos cualesquiera

Para resolver un triángulo cualquiera hace falta conocer tres datos: dos lados y un ángulo, dos ángulos y un lado o los tres lados.

Para ello nos apoyamos en dos resultados:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

Teorema del seno. En un triángulo cualquiera los lados son proporcionales a los senos de los ángulos que se les oponen.



$$a^2 = b^2 + c^2 - 2bc\cos A$$
$$b^2 = a^2 + c^2 - 2ac\cos B$$
$$c^2 = a^2 + b^2 - 2ab\cos C$$

Teorema del coseno (o de Pitágoras generalizado). En un triángulo cualquiera el cuadrado de un lado excede a la suma de los cuadrados de los otros dos en el doble del producto de dichos lados por el coseno del ángulo que forman.

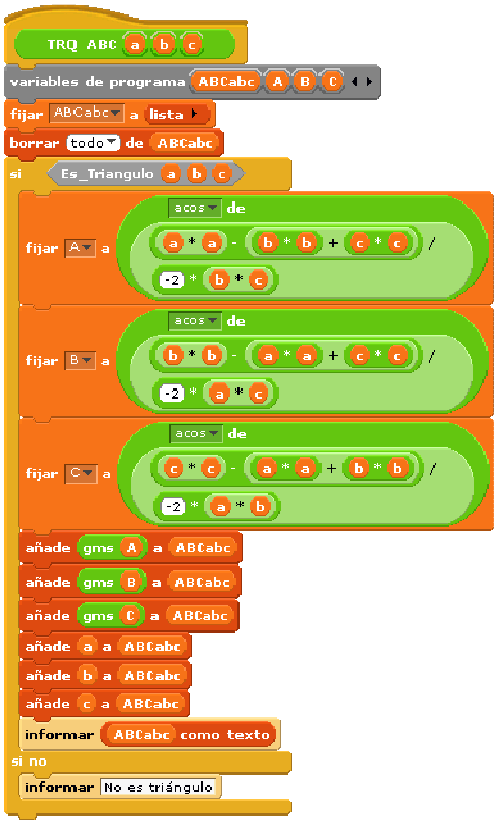
OBJETIVO

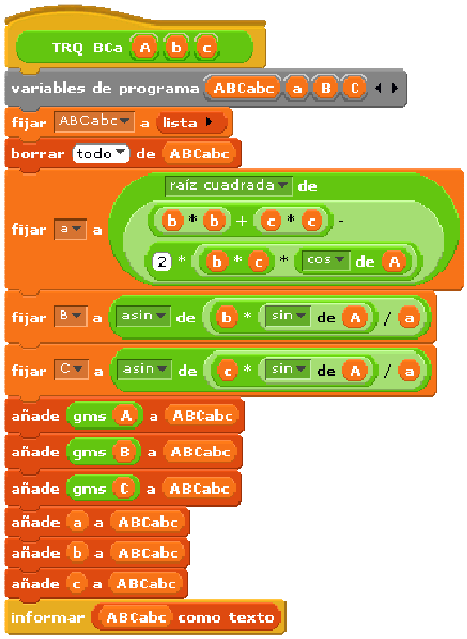
Elabora un programa que

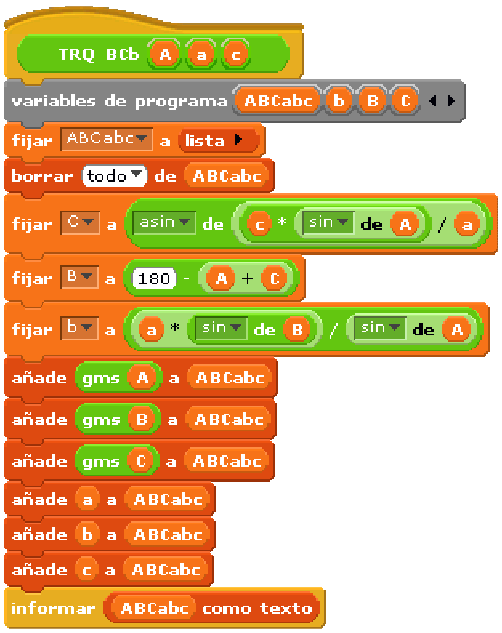
1) Dados tres posibles lados verifique si forman un triángulo y calcule los ángulos.
2) Dados dos lados y el ángulo que comprenden calcule el otro lado y los otros dos ángulos.
3) Dados dos lados y un ángulo no comprendido calcule el otro lado y los otros dos ángulos.
4) Dados dos ángulos y un lado calcule el otro ángulo y los otros dos lados.

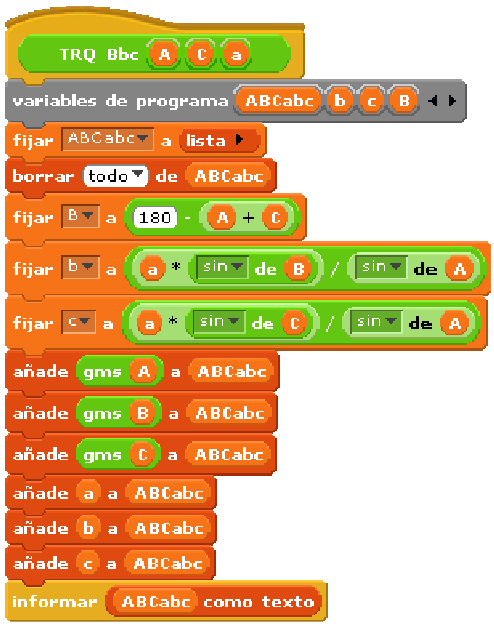
Utiliza los teoremas y programa las fórmulas para cada caso. Por ejemplo en la opción 2) conocidos A, b y c averiguamos a mediante la fórmula del teorema del coseno.
A continuación por el teorema del seno, conocido a/sen A y b averiguamos sen B y por tanto B.

NOTA: Las letras en el nombre de las funciones son los parámetros a calcular; los que faltan son los datos. Se devuelve una lista con los seis datos: los ángulos A, B y C y los lados a, b y c.









P49	Primos gemelos
	Dos números primos p y q (p<q) se dicen **primos gemelos** si q=p+2. Las parejas de primos gemelos menores que 100 son (3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73) Conforme se van considerando primos más grandes la frecuencia de aparición de pares de primos gemelos va disminuyendo, pero se sospecha que es cierta la conjetura que postula la existencia de infinitos pares de primos gemelos. Los números primos gemelos más grandes conocidos son el par $3756801695685 \cdot 2^{666669} - 1$ y $3756801695685 \cdot 2^{666669} + 1$. Fueron descubiertos el 25-12-2011 y tienen 200700 dígitos. Hay $808_{2}675.888_{1}577.436$ primos gemelos menores que un trillón. Se llaman **primos cuádruples** a una pareja de primos gemelos separados por 4 unidades (la distancia más pequeña posible). Los primeros primos cuádruples son (5, 7, 11, 13), (11, 13, 17, 19) y (101, 103, 107, 109). Un primo **aislado** es aquel cuyo número impar anterior y posterior no son primos. Ejemplo: 23 es primo aislado pues 21 y 25 son compuestos. En 1919 Viggo Brun demostró la convergencia de la serie $B_2 = \left(\frac{1}{3}+\frac{1}{5}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+\left(\frac{1}{11}+\frac{1}{13}\right)+\left(\frac{1}{17}+\frac{1}{19}\right)+\left(\frac{1}{29}+\frac{1}{31}\right)+\cdots$ mientras que la suma de los inversos de todos los números diverge. Si la serie fuera divergente, demostraría la infinidad de los primos gemelos. Pero no lo es. La suma de los inversos de todos los primos cuádruples también es convergente: $B_4 = \left(\frac{1}{5}+\frac{1}{7}+\frac{1}{11}+\frac{1}{13}\right)+\left(\frac{1}{11}+\frac{1}{13}+\frac{1}{17}+\frac{1}{19}\right)+\left(\frac{1}{101}+\frac{1}{103}+\frac{1}{107}+\frac{1}{109}\right)+\cdots$ Los valores aproximados de B_2 y B_4 son 1,902160583104 y 0,8705883800
OBJETIVO	Elabora un programa que calcule de manera aproximada los valores de B2 y B4 a partir de todos los primos gemelos y primos cuádruples menores que 10000. V2. Escribe los primos aislados menores que 10000.

☞ Reutiliza la función EsPrimo.

☞ Elabora una función booleana que responda a partir de un número p si p y p+2 (p, p+2, p+4 y p+6) son primos gemelos (cuádruples).

☞ Al detectar una pareja (cuaterna) acumula la suma de sus inversos en B_2 (B_4).

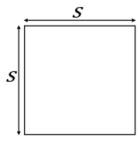
P50 Perímetros, áreas y volúmenes

Un resumen de fórmulas de geometría elemental es

GEOMETRY SHAPES AND SOLIDS

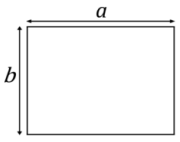
SQUARE

$P = 4s$
$A = s^2$



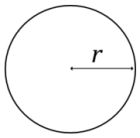
RECTANGLE

$P = 2a + 2b$
$A = ab$



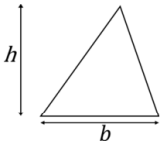
CIRCLE

$P = 2\pi r$
$A = \pi r^2$



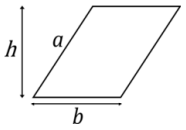
TRIANGLE

$P = a + b + c$
$A = \frac{1}{2}bh$



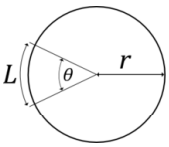
PARALLELOGRAM

$P = 2a + 2b$
$A = bh$



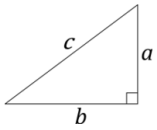
CIRCULAR SECTOR

$L = \pi r \frac{\theta}{180°}$
$A = \pi r^2 \frac{\theta}{360°}$



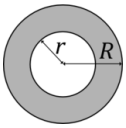
PYTHAGOREAN THEOREM

$a^2 + b^2 = c^2$
$c = \sqrt{a^2 + b^2}$



CIRCULAR RING

$A = \pi(R^2 - r^2)$

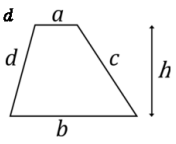


SPHERE

$S = 4\pi r^2$
$V = \frac{4\pi r^3}{3}$

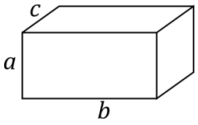
TRAPEZOID

$P = a + b + c + d$
$A = h\frac{a+b}{2}$



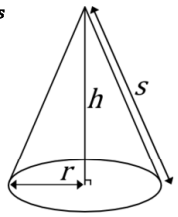
RECTANGULAR BOX

$A = 2ab + 2ac + 2bc$
$V = abc$



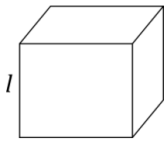
RIGHT CIRCULAR CONE

$A = \pi r^2 + \pi rs$
$s = \sqrt{r^2 + h^2}$
$V = \frac{1}{3}\pi r^2 h$



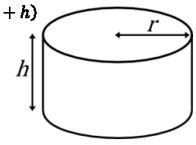
CUBE

$A = 6l^2$
$V = l^3$



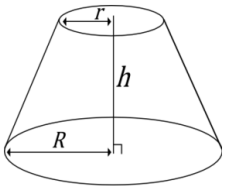
CYLINDER

$A = 2\pi r(r + h)$
$V = \pi r^2 h$



FRUSTUM OF A CONE

$V = \frac{1}{3}\pi h(r^2 + rR + R^2)$



Elabora un programa que selecciones una figura, dependiendo de la figura solicite las medidas que intervienen en las fórmulas y finalmente presente los resultados.

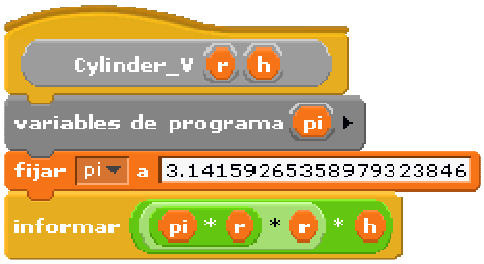
PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

Elabora una función para cada fórmula.

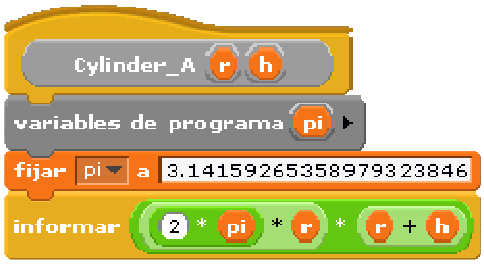
Es muy sencillo. Estas son las del cuadrado y cilindro:

Área del cuadrado de lado s

Perímetro del cuadrado de lado s



Volumen del cilindro de radio r y altura h



Área del cilindro de radio r y altura h

Descarga e importa en tu proyecto la biblioteca de funciones BIB Geometria

P51 Multiplicar a la manera de los egipcios

La multiplicación por duplicación es un antiguo método que no requiere conocer las tablas de multiplicar y solo requiere saber sumar. En el método egipcio se necesita saber duplicar y en el ruso saber dividir entre 2.

Método del Antiguo Egipto.
Ejemplo: 41 × 59.

En la primera columna, comenzando por 1 iremos duplicando el valor sin rebasar 41. En la segunda columna duplicaremos comenzando con 59. Como 41=32+8+1, sumaremos solo las filas que contienen estos valores en la primera columna y obtendremos el resultado.

Factor 1	Factor 2	Se suma
1	**59**	✓
2	118	
4	236	
8	**472**	✓
16	944	
32	**1888**	✓
Total	**2419**	

Método del campesino ruso.
Ejemplo: 41 × 59.

En la primera columna, comenzando por 41 iremos hallando la mitad entera (ignorando el resto) hasta llegar a 1. En la segunda columna duplicaremos comenzando con 59. Sumaremos solo las filas que contienen impares en la primera columna y obtendremos el resultado.

Factor 1	Factor 2	Se suma
41	**59**	✓
20	118	
10	236	
5	**472**	✓
2	944	
1	**1888**	✓
Total	**2419**	

OBJETIVO

Elabora un programa que multiplique según el método del Antiguo Egipto.
V2. Elaborar un programa que multiplique según el método del campesino ruso.

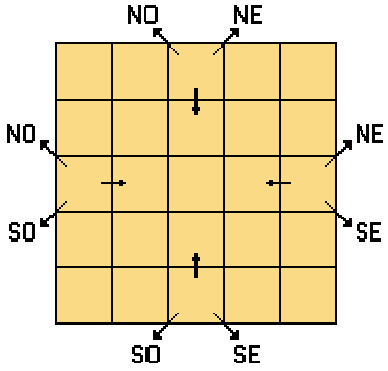
41 en binario es 101001. Bastará una lista con los duplicados del factor 2 y multiplicarla por los dígitos de la representación binaria recorrida a la inversa.

P52 Cuadrado mágico

Un **cuadrado mágico** es una tabla de números, con las mismas filas que columnas, en la que la suma de los números por columnas, filas y diagonales principales es la misma, la **constante mágica**. Usualmente los números empleados para rellenar las casillas son consecutivos, de 1 a n^2, siendo n el número de columnas y filas del cuadrado mágico.

Cuadrados de orden impar. Pueden generarse por el método ya conocido por los astrólogos orientales y publicado en 1691 por Simón de la Loubere. Comenzando en la casilla central de la primera fila con el primer número, se rellena la diagonal quebrada con los siguientes en sentido NO (ó NE). Completada la primera diagonal se desciende una posición y se rellena la segunda en el mismo sentido que la anterior, repitiéndose el paso anterior con el resto de diagonales hasta completar el cuadrado.

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9



Cuadrados mágicos de orden múltiplo de 4. Se construye un cuadrado con los números dispuestos consecutivamente. Las sumas de las diagonales son la constante mágica. Conservando la submatriz central de orden $n/2$ y las cuatro submatrices de esquina de orden $n/4$ los números restantes se giran 180º respecto del centro del cuadrado.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

⇨

1	2	62	61	60	59	7	8
9	10	54	53	52	51	15	16
48	47	19	20	21	22	42	41
40	39	27	28	29	30	34	33
32	31	35	36	37	38	26	25
24	23	43	44	45	46	18	17
49	50	14	13	12	11	55	56
57	58	6	5	4	3	63	64

Observar que el 17 estaba en (3,1) y ha pasado a (8+1-3,8+1-1).

	Para cuadrados mágicos de orden múltiplo de 4 más 2 existe un método algo más complicado denominado **LUX**.
OBJETIVO	Elabora un programa que construya un cuadrado mágico de orden impar dado. V2. Elabora un programa que construya un cuadrado mágico de orden múltiplo de 4 dado.
	Scratch no tiene matrices pero BYOB sí: una lista de listas.  es una lista cuyos elementos (fila1, fila2, fila3, fila4, fila5) son a su vez listas. Fijate que en el ejemplo (k=5) en cada "diagonal quebrada" se colocan k elementos antes de sumar 1 a la fila. La aritmética para recorrer la diagonal en sentido ascendente es restar 1 a la fila y sumar 1 a la columna. Para pasar a otra diagonal se suma 1 a la fila. Cuando el resultado es 6 se pone 1; cuando el resultado es 0 se pone 5.

P53 Cuadrado mágico con números primos

Un ejemplo de cuadrado mágico con números primos consecutivos de orden 4 es:

79	31	47	101
53	71	61	73
89	59	67	43
37	97	83	41

Una fórmula general para construir cuadrados mágicos de orden 3, a partir de 3 números a, b y c es

a+b	a-b-c	a+c
a-b+c	a	a+b-c
a-c	a+b+c	a-b

y su constante mágica es 3a.

Por ejemplo. Si a=59, b=-42 y c=12 tenemos:

17	89	71
113	59	5
47	29	101

cuadrado mágico con números primos de orden 3. Para construirlo hemos partido de un primo (a=59) y hemos buscado dos múltiplos de 6 (b=-42 y c=12) de forma que también sean primos a+b, a-b-c, a+c, a-b+c, a+b-c, a-c, a+b+c y a-b.

Una fórmula general para construir cuadrados mágicos de orden 4, a partir de 8 números a, b, c, d, e, f, g y h es:

a-c+e	a+b-d+g	a+d+f	a+c+h
a+f	a+c+d+h	a-c-d+g	a+b+e
a+g+h	a-c+d+e+f	a+b+c-d	a
a+b+c	a-d	a+d+e+h	a-c+g+f

y su constante mágica es 4a+b+e+f+g+h.

Por ejemplo. Si a=29, b=6, c=18, d=12, e=32, f=18, g=8 y h=24 tenemos (b, c, d y f son múltiplos de 6; e, g y h lo son de 4):

43	31	59	71
47	83	7	67
61	73	41	29
53	17	97	37

	cuadrado mágico con números primos de orden 4.
OBJETIVO	Elabora un programa que construya cuadrados mágicos de orden 3 a partir de un número primo dado, buscando múltiplos de 6 convenientes para los valores de b y c. V2. Elabora un programa que construya cuadrados mágicos de orden 4 a partir de 8 números dados.
	☞ Comprueba que los números candidatos a+b, a-b-c, a+c, a-b+c, a+b-c, a-c, a+b+c y a-b son primos. ☞ V2. Programa las fórmulas de cada casilla.

P54	Método de la orla para cuadrados mágicos de orden par

Los cuadrados mágicos de orden par son, en general, más difíciles de construir. Hemos visto un par de métodos para los múltiplos de 4 y múltiplos de 4 mas 2.

A finales del s. XVII Frénicle de Bessy propuso el **método de la orla** para cualquier orden par, que permite pasar de un cuadrado de orden par n a otro de orden n+2.

Sumemos 2n+2 a todos los elementos del cuadrado mágico de partida y añadamos una orla vacía alrededor (una fila arriba y abajo, una columna a derecha e izquierda).

Quedarán por distribuir en la orla los números del 1 al 2n+2 y del n^2+2n+3 al $(n+2)^2$.

Nombramos los números de la orla de esta manera:

$$\begin{array}{ccccccc} v & b_1 & b_2 & . & b_n & w \\ c_1 & & & & & c_1' \\ c_2 & & & & & c_2' \\ \dots & & & & & \dots \\ c_n & & & & & c_n' \\ w' & b_1' & b_2' & . & b_n' & v' \end{array}$$

La suma de un número y el mismo número con prima debe ser n^2+4n+5, la diferencia de las constantes mágicas de ambos cuadrados, es decir, $b_i + b_i' = c_i + c_i' = n^2 + 4n + 5$

Basta elegir de forma conveniente B={b_i}, C={c_i}, v y w.

Para n=4 elegiremos:

B={33,7,3,1} C={29,27,9,6} v=35 w=32

Para n=6 elegiremos:

B={63,62,59,58,5,8} C={56,54,52,10,12,14} v=1 w=4

Para n múltiplo de 4 mayor que 4:

Si $n = 4m$, $m \geq 2$ podemos tomar $v = n^2 + 3n + 7, \quad w = n^2 + 3n + 4,$

$$B = \{n^2 + 3n + 5, n + 3, n - 1, n - 3\}$$

$$\bigcup_{k=0}^{\frac{n-8}{4}} \{n - 7 - 4k, n - 4 - 4k, n^2 + 3n + 10 + 4k, n^2 + 3n + 11 + 4k\},$$

$$C = \{n^2 + 3n + 1, n^2 + 3n - 1, n + 5, n + 2\}$$

$$\bigcup_{k=0}^{\frac{n-8}{4}} \{n + 7 + 4k, n + 10 + 4k, n^2 + 3n - 3 + 4k, n^2 + 5n - 4 + 4k\}$$

Para n múltiplo de 4 más 2 mayor que 6:

Si $n = 4m + 2$, $m \geq 2$ tomamos $v = n - 5$, $w = n - 2$,

$$B = \{n^2 + 3n + 9, n^2 + 3n + 8, n^2 + 3n + 5, n^2 + 3n + 4, n - 1, n - 2\}$$

$$\bigcup_{k=0}^{\frac{n-10}{4}} \{n - 6 - 4k, n - 9 - 4k, n^2 + 3n + 12 + 4k, n^2 + 3n + 13 + 4k\},$$

$$C = \{n^2 + 3n + 2, n^2 + 3n, n^2 + 3n - 2, n + 4, n + 6, n + 8\}$$

$$\bigcup_{k=0}^{\frac{n-10}{4}} \{n + 9 + 4k, n + 12 + 4k, n^2 + 3n - 5 - 4k, n^2 + 3n - 6 - 4k\}.$$

OBJETIVO

Elabora un programa que a partir de un cuadrado mágico de orden par n construya otra de orden n+2 por el método de la orla.

Puedes partir del cuadrado mágico de orden 4 y aplicar los valores dados para n=4.

7	12	1	14
2	13	8	11
16	3	10	5
9	6	15	4

P55 Conjetura de Brocard

La conjetura de Brocard afirma que el número de primos comprendidos entre el cuadrado de un primo mayor que 2 y el cuadrado del primo siguiente es mayor que 4.
Por ejemplo, dados 5 y 7 sus cuadrados son 25 y 49.
Entonces entre 25 y 49 hay al menos 4 primos.
En efecto, hay 6: 29, 31, 37, 41, 43 y 47.
En teoría de números se nombra π(x)=número de primos menores o iguales que x y esta conjetura se escribe $\pi[(p+1)^2]-\pi(p^2)>4$ si $p>2$.

OBJETIVO

Elabora un programa que compruebe la conjetura de Brocard con los primos menores que 100000.

Para presentar el resultado crea una lista cuyos elementos serían ternas (primo, primo siguiente, número de primos entre sus cuadrados).

Primo	Primo siguiente	Número de primos entre sus cuadrados
3	5	5
5	7	6
7	11	15
11	13	9

P56 Conjetura de Collatz

La conjetura de Collatz afirma que dado cualquier número natural n la reiteración del siguiente procedimiento acaba siempre en 1: si es par lo dividimos entre 2 y si es impar lo triplicamos y le sumamos uno.

Ej: 17. Como es impar lo triplicamos y sumamos 1. Obtenemos 17·3+1=52. Como es par lo dividimos entre 2 y obtenemos 26 (par) y luego 13 (impar). 13·3+1=40 (par), lo dividimos entre 2 y obtenemos 20(par), 10(par) y 5 (impar). 5·3+1=16 (par), lo dividimos entre 2 y obtenemos 8(par), 4(par), 2(par) y finalmente 1.

OBJETIVO

Elabora un programa que compruebe la conjetura de Collatz con cualquier número introducido por el usuario. Muestra las operaciones y su resultado en una lista.

La lista del caso 17 sería (I, inicial; T, triplicar más 1; M, mitad):

{(I,17), (T,52), (M,26), (M,13), (T,40), (M,20), (M,10), (M,5), (T,16), (M,8), (M,4), (M,2), (M,1)}.

P57	Conjetura de Grimm
	La conjetura de Grimm afirma que dados k números compuestos consecutivos existen k números primos **distintos** que los dividen. Ejemplo: Entre 13 y 17 está la tripleta 14,15 y 16 de compuestos consecutivos. 7 divide a 14, 5 divide a 15 y 2 divide a 16. Los tres primos distintos buscados son 7, 5 y 2.
OBJETIVO	Elabora un programa que compruebe la conjetura de Grimm con cualquier número introducido por el usuario.
	Si el número es primo deberá mostrar la secuencia de compuestos entre el número y el siguiente primo. Si el número es compuesto deberá encontrar los compuestos que le anteceden o suceden entre el primo anterior y el siguiente primo. Guarda los compuestos en una lista. Descomponlos usando el programa P3. Guarda las descomposiciones en otra lista.  ¿Cómo seleccionamos los primos diferentes?

P58	Conjetura de Oppermann
	La conjetura de Oppermann afirma que dado un número natural x>1 existe al menos un primo entre x(x-1) y x^2 y al menos otro primo entre x^2 y x(x+1). Ejemplo: Dado x=4, entre 12=4·3 y 16=4·3 tenemos el primo 13 y entre 16 y 20=4·5 tenemos los primos 17 y 19. En teoría de números se nombra π(x)=número de primos menores o iguales que x y esta conjetura se escribe $\pi(x^2-x)<\pi(x^2)<\pi(x^2+x)$.
	Elabora un programa que compruebe la conjetura de Oppermann con cualquier número introducido por el usuario.
	 Construye la lista de los primos entre x(x-1) y x(x+1).

P59	Conjetura de Polignac
	La conjetura de Polignac afirma que dado un número natural n existen infinitos casos de números primos consecutivos que se diferencian en n. Cuando n=2 tenemos la infinitud de los primos gemelos.
OBJETIVO	Elabora un programa que encuentre las tres primeras parejas de primos consecutivos que difieren en cualquier número introducido por el usuario.
	Limita el valor de n a 10000.

P60	Segunda conjetura de Hardy-Littlewood
	La segunda conjetura de Hardy-Littlewood afirma que dados dos naturales x e y el número de primos menores que x+y es menor o igual que la suma del número de primos menor que x más el número de primos menor que y. En teoría de números se nombra π(x)=número de primos menores o iguales que x y esta conjetura se escribe $\pi(x+y) \leq \pi(x)+\pi(y)$.
	Elabora un programa que verifique la certeza de la segunda conjetura de Hardy-Littlewoord a partir de dos números naturales introducidos por el usuario.
	 Comprueba la primalidad de todos los números entre 2 y x+y. En el mismo bucle cuenta π(x) y π(y) averiguando primero si x>y o al contrario x<y.  Indica los tres valores π(x+y), π(x) y π(y).

P61	Aproximación del número e

El número irracional e es una constante fundamental en matemáticas. Es la base de las funciones exponenciales y logarítmicas.

e

$e \approx 2.71828182845904523536028747135 3$

La siguiente serie infinita converge en e:

$$2 + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \ldots$$

Esta serie es el desarrollo en serie de Taylor de la función e^x cuando x=1: e^x = 1+x/1!+x²/2!+x³/3!+x⁴/4!+ …

La aproximación $e \approx (1 + 9^{-4^{7 \cdot 6}})^{3^{2^{85}}}$, tiene según su autor $18_4457.734_3525.360_2901.453_1873.570$ de decimales exactos, y usa todas las cifras del 1 al 9. Esta basada en la definición del número $e = \lim_{n \to \infty} (1 + \frac{1}{n})^n$ pues $9^{-4^{7 \cdot 6}} = \frac{1}{(3^2)^{2 \cdot 42}} = \frac{1}{(3^2)^{84}}$.

Es famosa la ecuación de Euler que relaciona las principales constantes de la aritmética: $e^{i\pi} - 1 = 0$ siendo $i = \sqrt{-1}$. Benjamín Pierce escribía en la pizarra esta versión $i^{-i} = \sqrt{e^{\pi}}$ y decía a sus alumnos: *Gentlemen, we have not the slightest idea what this equation means, but we may be sure that it means something very important.*

OBJETIVO

Elabora un programa que calcule las primeras 120 fracciones del tipo 1/n! y las sume. Obtendrás 200 cifras decimales exactas de e.

V2. Calcula la aproximación de e con P38. Algoritmo de exponenciación rápida.

Observa que $\frac{1}{1 \cdot 2} = \frac{1}{2!}$, $\frac{1}{1 \cdot 2 \cdot 3} = \frac{1}{3!}$, $\frac{1}{1 \cdot 2 \cdot 3 \cdot 4} = \frac{1}{4!}$.

P62 — Algoritmo de la raíz cuadrada

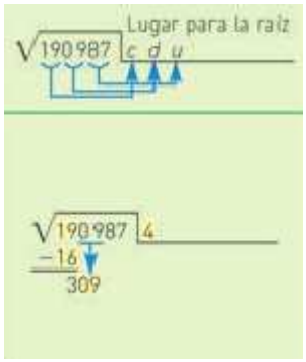
Uno de los algoritmos de cálculo manual más complicados que se muestran en la enseñanza secundaria es el de la extracción de la raíz cuadrada de un número.

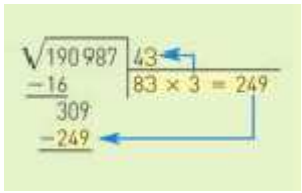
REGLA DE CÁLCULO DE LA RAÍZ CUADRADA.

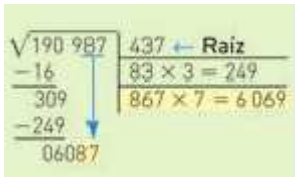
1) Formamos grupos de dos cifras comenzando por las unidades y calculamos la raíz entera del primer grupo que será el primer dígito de la raíz cuadrada resultado.
2) Restamos del primer grupo el cuadrado de la raíz entera hallada, bajamos el grupo siguiente y conformamos con el resto y las dos cifras bajadas un número cuya raíz entera cabe hallar.
3) Doblamos el resultado parcial y ensayamos añadirle una cifra hasta que el producto del número formado y dicha cifra añadida sea la raíz cuadrada entera buscada.
4) Hallamos el resto, bajamos el siguiente grupo y subimos la cifra añadida a la zona de resultados.
5) Nos reiteramos en 3 y 4 hasta que se agoten los grupos.

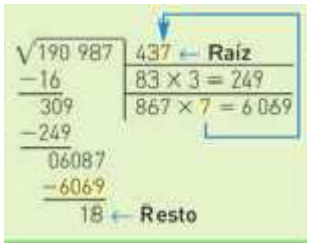
Nota: Para obtener cifras decimales se coloca la coma decimal en el resultado en el momento que se agoten los grupos y bajaremos grupos de 2 ceros a discreción.

Como ejemplo del algoritmo de extracción de la raíz véase el cálculo para $\sqrt{190987}$. Se extrae la raíz entera, sin decimales.









OBJETIVO

Elabora un programa que calcule la raíz cuadrada entera de un entero.

Fijate que la obtención del primer dígito y del primer resto (pasos 1 y 2) requieren un cálculo diferente al del resto de dígitos de la solución (pasos 3 y 4).

P63	Estadística bidimensional

Cuando de una **población** estudiamos ***dos* características o variables cuantitativas, x e y,** con la finalidad de **encontrar alguna relación** entre ellas enfrentamos un problema de estadística bidimensional.

Ejemplo: relación entre el peso y la altura de las personas.

Un primer análisis gráfico lo proporciona la **nube de puntos,** resultante de dibujar los pares en unos ejes cartesianos.

La **covarianza** es el parámetro que mejor resume la interdependencia. Se define como:

$$s_{xy} = \frac{\Sigma_{i=1}^{n}(x_i-\overline{x})\cdot(y_i-\overline{y})}{N} = \frac{\Sigma_{i=1}^{n}x_i\cdot y_i}{N} - \overline{x}\cdot\overline{y}$$

Obsérvese que está totalizando y promediando productos de desviaciones respecto de las medias y tiene gran parecido con la varianza (basta igualar y_i con x_i).

Llamamos **correlación** a la dependencia estadística que existe entre las dos variables. Se mide con el **coeficiente de correlación lineal o de Pearson** que relaciona la covarianza con el producto de las varianzas.

$$r = \frac{s_{xy}}{s_x\cdot s_y}$$

Se llama **recta de regresión** a aquella que mejor ajusta la nube de puntos, minimizando el cuadrado de las distancias. Permite estimar parejas no incluidas en la muestra. Sus ecuaciones son:

$$y-\overline{y} = \frac{s_{xy}}{s_x^2}\cdot(x-\overline{x}) \qquad x-\overline{x} = \frac{s_{xy}}{s_y^2}\cdot(y-\overline{y})$$

Ambas rectas se cortan en el **centro de gravedad** $(\overline{x},\overline{y})$ de la nube de puntos.

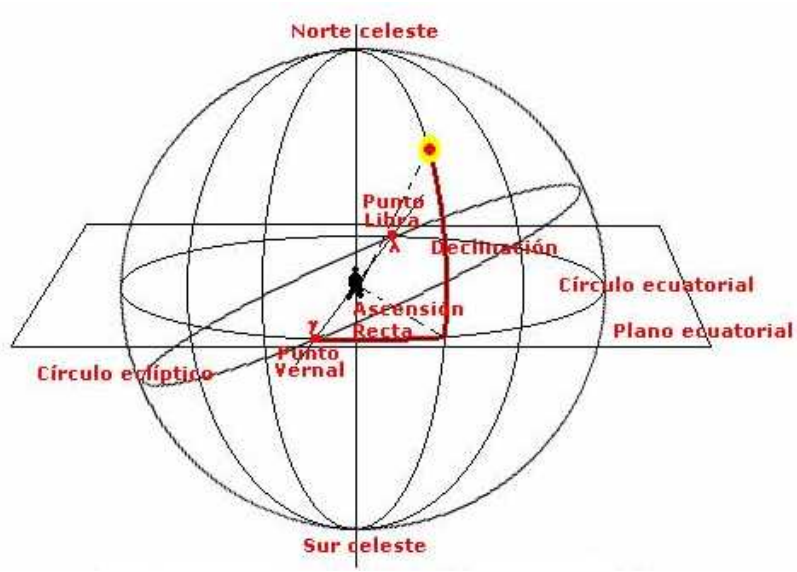
OBJETIVO

Elabora un programa que calcule la covarianza, el coeficiente de correlación de Pearson, las rectas de regresión y el centro de gravedad de dos conjuntos con el mismo número de datos.

Comprueba que ambas listas de datos tienen el mismo tamaño.

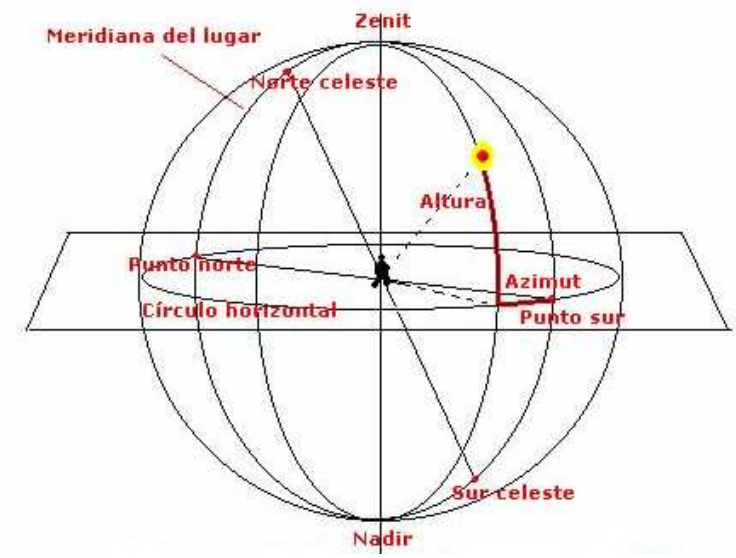
P64	Coordenadas celestes

Los catálogos de objetos estelares suelen utilizar las coordenadas ecuatoriales ascensión recta (AR) y declinación (DEC) para describir la posición en la esfera celeste.



La ascensión recta se mide en horas, minutos y segundos desde el punto vernal o Aries (corte de la eclíptica y el ecuador celeste) en sentido antihorario. La declinación en grados, minutos y segundos, norte o sur, desde el plano ecuatorial al objeto.
Eje celeste y punto Aries no dependen del observador.

La observación del cielo por un aficionado utiliza un sistema de referencia vinculado al observador (sur y cénit) conocido como coordenadas horizontales: azimut (AZ) y altura (ALT).



El azimut varía de 0º a 360º y se mide desde el punto sur de la línea meridiana del lugar (donde culminan todos los astros) en dirección este. La altura se mide sobre el horizonte del observador en grados, minutos y segundos y puede ser norte (por encima) o sur (por debajo).

Las coordenadas ascendentes (AR y AZ) se relacionan a través de la ecuación del tiempo sidéreo en la que interviene la longitud (LON) del observador y el tiempo de la observación (TU).

En concreto, si TSML es el tiempo sidéreo medio local el ángulo horario es AH = TSML - AR donde TSML = TSMG + LON y TSMG es el tiempo sidéreo medio de Greenwich, que es una función polinómica de TU.

Las coordenadas declinantes (DEC y ALT) se relacionan la latitud (LAT) del lugar (desviación del cénit respecto del polo celeste).

Las ecuaciones que permiten transformar AR y DEC en AZ y ALT (una vez hallado el ángulo horario AH) son:

$$\operatorname{sen}(ALT) = \operatorname{sen}(DEC)\cdot\operatorname{sen}(LAT) + \cos(DEC)\cdot\cos(LAT)\cdot\cos(AH)$$

$$\cos(A) = [\operatorname{sen}(DEC) - \operatorname{sen}(ALT)\cdot\operatorname{sen}(LAT)] / [\cos(ALT)\cdot\cos(LAT)]$$

Si sen(AH) es negativo AZ = A, si positivo AZ = 360º - A

Para hallar el AH del lugar y momento de la observación se sigue el siguiente procedimiento:

1) TU (hor, min, seg) debe expresar la hora solar.

   En España a la hora oficial hay que restar 2 horas en horario de verano (desde el último domingo de marzo al último sábado de octubre) y una en horario de invierno.

2) Transformar la fecha (año, mes, dia) en días (DJ) y fracción de siglo (TJ) transcurridos desde 1-1-2000.

```
Si mes de Enero o Febrero restar 1 al año y sumar 12 al mes.
a = entera(año/100)
b = 2 - a + entera(a/4)
c = entera(365.25·año)
d = entera(30.6001·(mes + 1))
DJ = b + c + d - 730550.5 + dia +(hor + min/60 + seg/3600)/24
TJ = DJ/36525
```

3) Aplicar la ecuación polinómica a DJ y TJ

$$TSMG = 280.46061837 + 360.98564736629\cdot DJ + 0.000387933\cdot TJ^2 - TJ^3/38710000$$

4) Hallar la suma incompleja en grados

```
AH = TSMG mod 360 + LON - AR
```

OBJETIVO

Elabora un programa que permita encontrar un objeto estelar a cualquier observador en su lugar de observación.

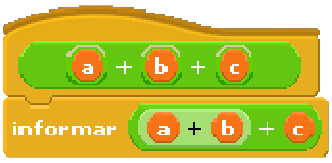
Deberá solicitar los siguientes datos:

1. Del objeto estelar: AR y DEC
2. Del lugar de observación: LON y LAT
3. Del momento de observación: TU

Deberá calcular AZ y ALT.

Datos de prueba

Pleyades M45, AR 03h 47m, DEC +24°07'
observadas desde 42°21' N, 71°04' W
con TU 06/04/2004 21:00:00
AZ = 283.967°, ALT = 21.0656°

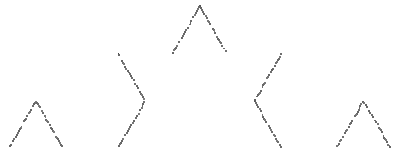
P65	Operadores ternarios
	Los operadores aritméticos y de cadena que proporciona Scratch/BYOB son binarios, es decir, contemplan dos operandos. Una mejora evidente es ampliar su número a tres y que sean ternarios.
	Crea operadores de tres sumandos para sumar, tres factores para multiplicar y tres cadenas para unir.
	Mira la solución para la suma
	Para definir la suma de tres operandos:  cuya apariencia será 

P66	Piedra, papel o tijera
	Piedra, papel o tijera es un juego infantil de manos en el cual existen tres elementos. La piedra que vence a la tijera rompiéndola, la tijera que vence al papel cortándolo y el papel que vence a la piedra envolviéndola. Representa un ciclo que da su esencia al juego. Este juego es muy utilizado para decidir quien de dos personas hará algo, tal y como a veces se hace usando una moneda, o para dirimir algún asunto. Si los jugadores eligen la misma arma o cada uno una diferente (en caso de tres jugadores) es un empate y se juega otra vez. Cada una de estas pequeñas partidas se repite hasta que uno de los jugadores gana dos veces de tres o tres de cinco, siendo entonces el vencedor del juego.
	Utilizando los disfraces de piedra, papel o tijera simular un juego entre dos jugadores A y B que realizan una partida al mejor de cinco.
	☞ Usa la función  para generar las apuestas de cada jugador. ☞ Mostrar los disfraces según el número obtenido:    1) Piedra 2) Papel 3) Tijera ☞ Mostrar quien gana y los marcadores.

P67	Copo de nieve de Koch

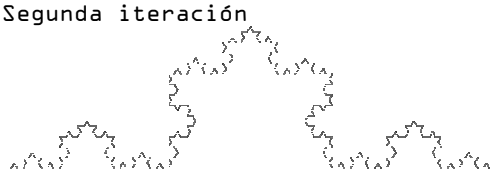
El **copo de nieve de Koch** o **estrella de Koch** es una curva cerrada continua pero no diferenciable en ningún punto descrita por el matemático sueco Helge von Koch en 1904 en un artículo titulado *Acerca de una curva continua que no posee tangentes y obtenida por los métodos de la geometría elemental*. En lenguaje actual diríamos que es una curva fractal. Su construcción más simple se realiza mediante un proceso iterativo: se toma un segmento, se lo divide en tres partes iguales, se remplaza la parte central por dos partes de igual longitud haciendo un ángulo de 60 grados. Luego, con los cuatro segmentos, se procede de la misma manera, lo que da lugar a 16 segmentos más pequeños en la segunda iteración. Y así sucesivamente.

Primera iteración



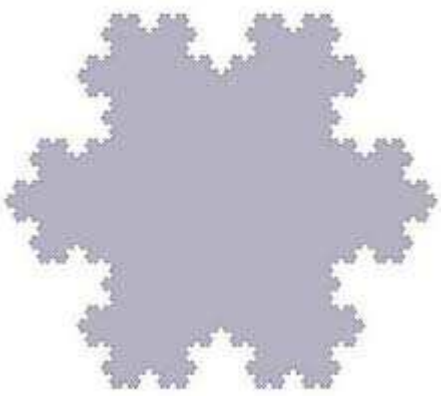
Segunda iteración

Tercera iteración



Cuarta iteración

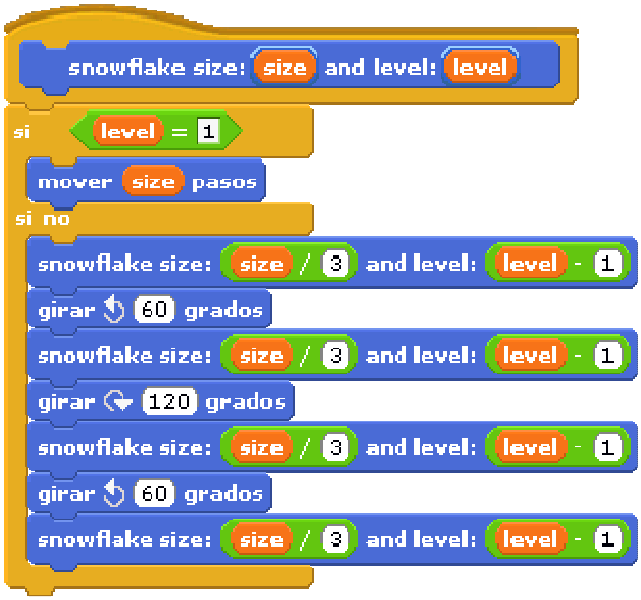
Tres de estas curvas unidas forman el copo de nieve de Koch



OBJETIVO

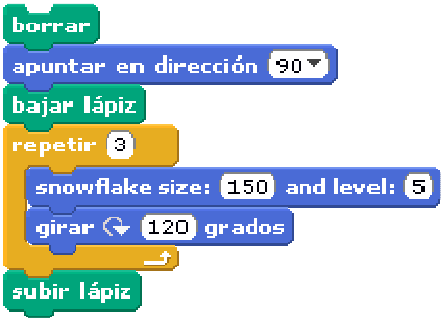
Programar una función recursiva que dibuje el copo de nieve de Koch indicando el tamaño del segmento original y el número de iteraciones.

Usa una función recursiva similar a esta traduciéndola al castellano. Ubícala en el menú Movimiento.

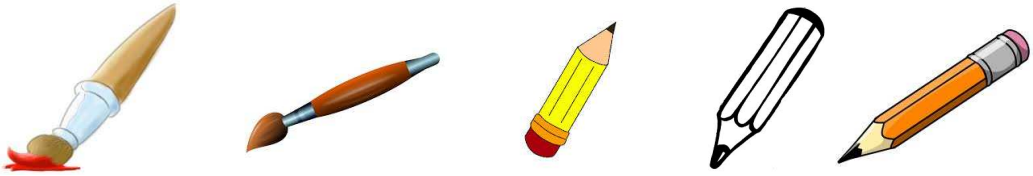


Para pintar las tres curvas de Koch asocia el siguiente código al objeto pintor.

Se muestra un ejemplo de tamaño (*size*) 150 y 5 iteraciones (*level*).



Posibles objetos pintores:



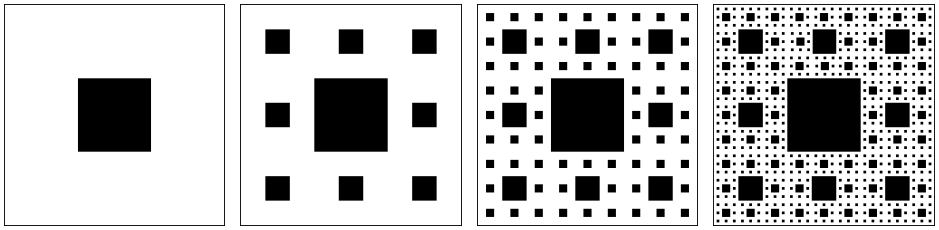
P68	Espiral cuadrada
	Para dibujar una **espiral cuadrada** partiremos de un punto origen y dibujaremos un segmento horizontal hacia el este de longitud l, giraremos a la izquierda 90º (norte) y dibujaremos otro segmento de la misma longitud, giraremos a la izquierda 90º (oeste) y dibujaremos un segmento de longitud l+d, giraremos a la izquierda 90º (sur) y dibujaremos un segmento de longitud l+d. Con estos 4 pasos completamos un giro de 360º. Continuamos con un segmento de longitud l+2d en dirección este y así sucesivamente girando siempre un ángulo recto a la izquierda e incrementando la longitud del segmento cuando corresponde trazarlo horizontalmente (una de cada dos veces). Cuando el origen es (0, 0) y la longitud l y el incremento d valen 1 se recorren todos los puntos de coordenadas enteras del plano: {(0,0), (1,0), (1,1), (0,1), (-1,1), (-1,0), (-1,-1), (0,-1), (1,-1), (2,-1), (2,0), (2,1), (2,2), … }.
	Elaborar un programa que partiendo del centro de la pantalla pinte una espiral cuadrada de longitud l e incremento d y se detenga cuando alcance el borde. V2. Programar una función que dado un número entero n le asigne el punto del plano de la secuencia de la espiral cuadrada que comienza en (0,0) cuando l=d=1. Es decir, si n=0 el punto es (0,0), si n=1 el punto es (1,0), si n=7 el punto es (-1,-1), etc.
	☞ Que el objeto está tocando el borde se detecta con la condición  del menú  . ☞ V2. Observa el valor de n que corresponde a coordenadas iguales y positivas: (1,1) es n=2, (2,2) es n=12, (3,3) es n=30, (4,4) es n=56. En el caso de igual valor absoluto y tercer cuadrante: (-1,1) es n=4, (-2,2) es n=16, (-3,3) es n=36, (-4,4) es n=64. {4, 16, 36, 64, … } es la sucesión $\{2^2, 4^2, 6^2, 8^2, \ldots\}$.

P69	Alfombra de Sierpinski

La **alfombra de Sierpinski** es un conjunto fractal descrito por primera vez por el matemático polaco Waclaw Sierpinski en 1916. La construcción de la alfombra se define de forma recursiva:

1. Comenzamos con un cuadrado.
2. El cuadrado se corta en 9 cuadrados congruentes y se elimina el cuadrado central.
3. El paso anterior vuelve a aplicarse recursivamente a cada uno de los 8 cuadrados restantes.

La alfombra de Sierpinski es el límite de este proceso tras un número infinito de iteraciones. Las figuras de abajo muestran el resultado de 1, 2, 3 y 4 iteraciones.



OBJETIVO

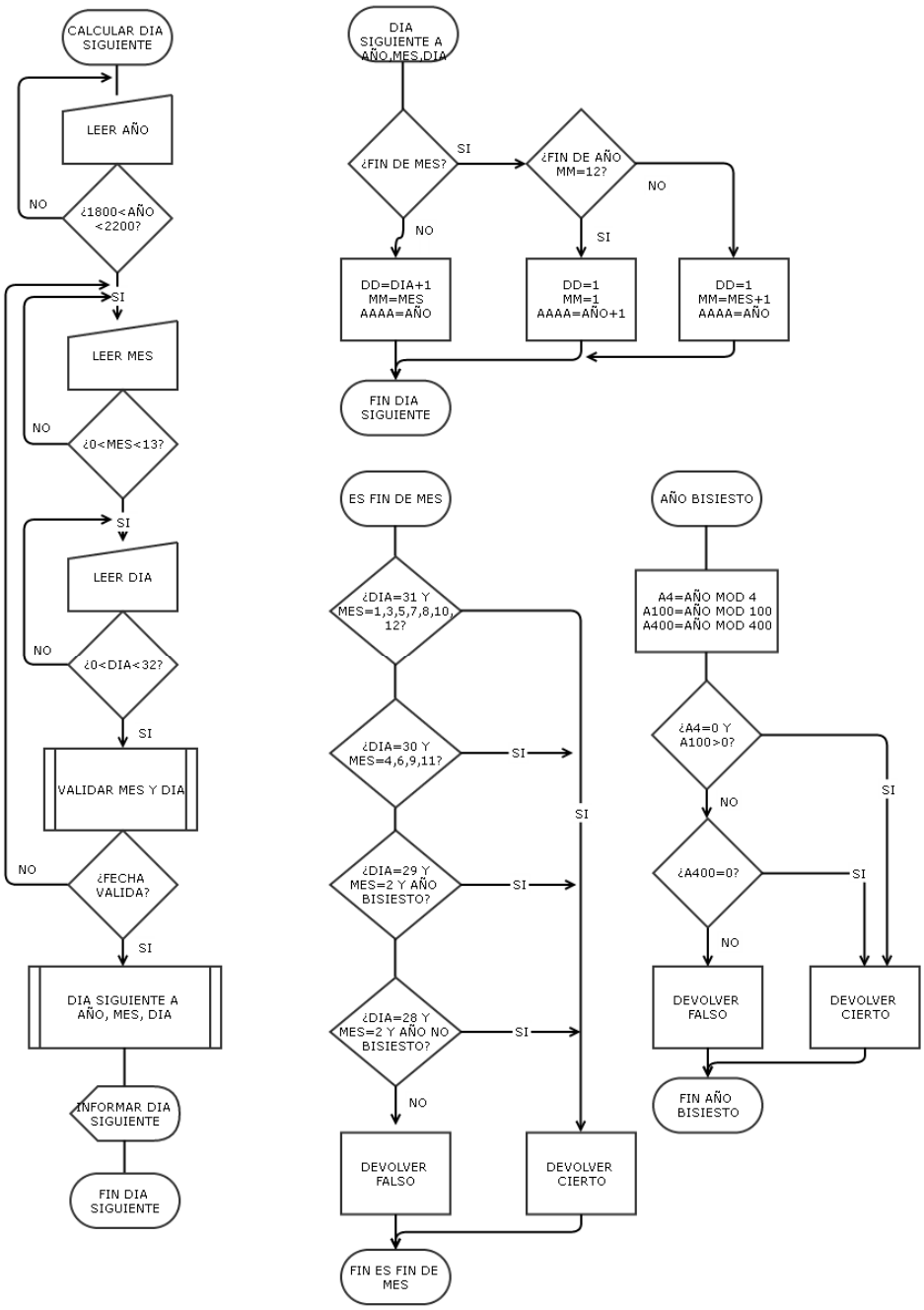
Elaborar un programa que dibuje una alfombra de Sierpinski 200x200 tras 4 iteraciones.

El siguiente código funciona en C++ y Java

```
/**
* Decide si un punto (x, y) debe estar relleno o no.
* @param x,y     Coordenadas x,y del punto testado
* @param ancho, alto Ancho y alto de la alfombra de Sierpinski
* @return 1 si se rellena el punto, 0 si no se rellena
*/
int RellenoSierpinski(int x, int y, int ancho, int alto){
// Caso base 1. Fila arriba, columna izquierda o fuera de límites se rellena
        if ((x <= 0)||(y <= 0)||(x>=ancho)||(y>=alto)) return 1;
// Si hemos dividido en 9 partes, ¿en qué parte (x2,y2) se ubica x,y?
        int x2 = x * 3 / ancho;
        int y2 = y * 3 / alto;
// Caso base 2. Si es cuadrado central no se rellena
        if (x2 == 1 && y2 == 1) return 0;
// Caso general. Preparamos y realizamos la llamada recursiva
        x -= (x2 * ancho+2) / 3;
        y -= (y2 * alto+2) / 3;
        ancho = (ancho +2-x2)/3;
        alto = (alto+2-y2)/3;
        return RellenoSierpinski(x, y, ancho, alto);
}
```

P70	Efecto 2000
	En 1999 dos bulos recorrieron el mundo: o El fin del milenio, que pronosticaba el fin de los tiempos con caída de meteorito incluida. o El efecto 2000, que predecía un posible mal funcionamiento de cualquier aparato electrónico dotado de reloj o calendario. El primer bulo ignoraba que el conteo de los años de la era cristiana comenzó con el año 1 con lo que el primer milenio finalizó el 31-12-1000 y el inicio del tercer milenio era el 1-1-2001 y no el 1-1-2000. El segundo bulo se fundamentaba en la costumbre informática, hasta mediados de los 80, de codificar y almacenar fechas con 6 dígitos numéricos en formato AAMMDD. Este formato ordena de forma creciente y sin problemas las fechas del mismo siglo. Pero al 31-12-1999, almacenado como 991231, sucedería el 01-01-2000, almacenado como 000101, un número menor. La solución es usar el formato AAAAMMDD, que pospone 8000 años el problema.
	Elaborar un programa que valide una fecha introducida en tres campos AAAA (año), MM (mes) y DD (día). Elaborar un programa que calcule el día siguiente a un día dado. Solicitará y validará el día.
	☞ Recuerda que *Trenta días tiene noviembre,* *Con abril, junio y septiembre.* *Los demás, a treinta y uno,* *menos uno (febrero).* ☞ Calcular el día siguiente no tiene ningún problema (basta sumar 1 a DD) **cuando no estamos a fin de mes.** Si estamos a fin de mes el día siguiente es 1 y sumaremos 1 a MM (**cuidado si es fin de año**). ☞ Necesitarás saber cuándo un año es bisiesto: ha de se ser múltiplo de 4, pero si lo es de 100 también ha de serlo de 400. Ejemplos: 1600 y 2000 fueron bisiestos. Otra falacia que corría en 1999 era que no lo era pues 1700, 1800 y 1900 no lo habían sido.

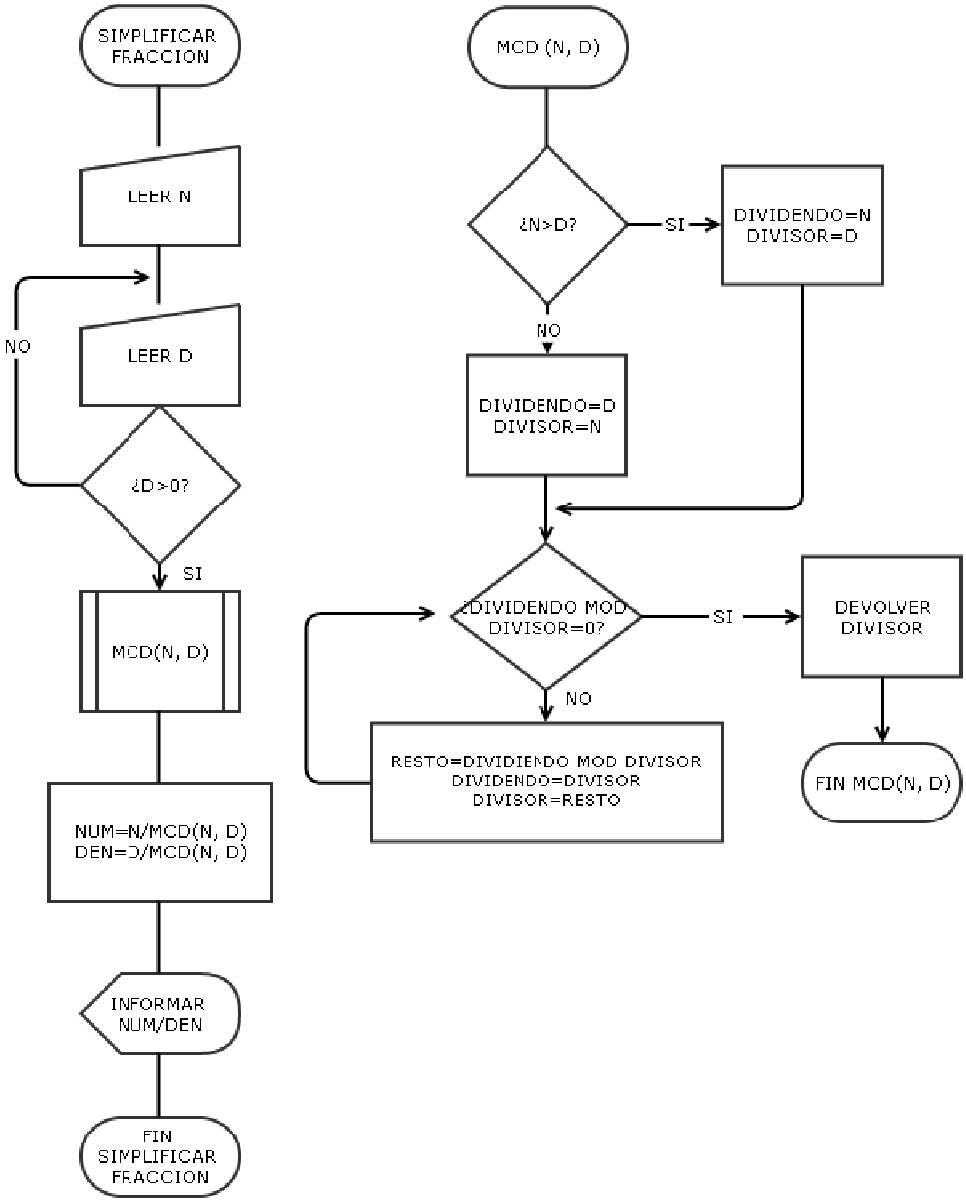
Un diagrama de flujo de la solución, hecho con la herramienta on-line Gliffy, es:



Falta la rutina **Validar mes y dia**. ¿Sabrías construirla fijándote en la rutina **Es fin de mes**?

P71	Simplificar fracciones
	Una de las operaciones aritméticas más odiadas por los alumnos de secundaria es la simplificación de fracciones. El procedimiento más sencillo es probar si el numerador y el denominador son divisibles a la vez por 2, 3, 4, … Si lo son se dividen ambos por el divisor común y comenzamos de nuevo con la nueva fracción reducida. ¿Cuándo podemos parar de probar divisores? Cuando lleguemos a la raíz cuadrada entera del menor de ellos. Otro procedimiento más contundente, que produce la fracción irreducible con una sola división, es dividir numerador y denominador entre el máximo común divisor de ambos. Pero rara vez recordamos cómo se calcula el m.c.d.: *es el producto de los factores comunes elevados al mínimo exponente*. Existe un algoritmo, debido a Euclides, que proporciona el m.c.d de dos números haciendo unas pocas divisiones enteras (véase P4).
	Elaborar un programa que dados dos números enteros, numerador(n) y denominador(d), obtenga la fracción irreducible num/den. V1. Probando a dividir entre 2, 3, 4, … V2. Dividiendo por el m.c.d.(n, d). En cualquier caso, muestra en una lista los divisores que han simplificado la fracción.
	Asegúrate que el denominador (número de partes en que se divide la unidad) no es 0 ni negativo.

Un diagrama de flujo de la solución a V2 es



P72	Diferencia de días
	Uno de los instrumentos esenciales de la matemática financiera es el cálculo de intereses o beneficios que genera un capital en un tiempo dado. Por ello necesitaremos calcular con frecuencia los días que han pasado desde una fecha inicial hasta una fecha posterior final.
	Elaborar un programa que solicitadas dos fechas las valide y calcule los días que transcurren desde la menor a la mayor. V1. Fecha de referencia 1-1-1950. V2. Fecha de referencia 1-1-2000.
	☞ Las fechas se pedirán en formato AAAAMMDD separando la petición de los tres campos (AAAA, MM y DD). ☞ Una posible solución consiste en averiguar cuántos días han pasado desde una fecha común de referencia y restar las cantidades.

P73	Interés simple e interés compuesto
	La fórmula de cálculo del interés simple es I=CRT/36000 siendo C, el capital que produce intereses; R, el rédito o tanto por ciento de interés anual y T el tiempo en días que el capital está invertido. El interés se dice compuesto cuando el interés, una vez devengado, también produce intereses. La fórmula de cálculo del interés compuesto es $C_f=C_i(1+R/100)^{T/360}$ siendo C_f, el capital acumulado al final del periodo; C_i, el capital inicial invertido y R y T como antes. El denominador 36000 recoge la costumbre de utilizar días comerciales (12 meses de 30 días son 360 días comerciales). Está multiplicado por 100 por que el rédito se ha expresado en tanto por cien. Cuando se usa el año civil (días naturales) el denominador es 36500. El conteo de días siempre es en días naturales. Sin embargo, las entidades financieras usan a su conveniencia el denominador 36000 o 36500. Cuando han de pagar intereses (dinero de una cuenta corriente o un depósito a plazo de un cliente) usan 36500, cuando han de cobrar intereses (descuento comercial de una letra de cambio o un préstamo) usan 36000.
	Elaborar un programa que solicite un capital C, un tipo de interés anual R y dos fechas (inicial y final) y calcule el interés simple y compuesto. V1. Usando denominador 36000. V2. Posibilidad de elegir días civiles o comerciales.
	Usa P72 para calcular los días naturales que el capital produce intereses.

P74	Cuotas de capitalización y amortización
	Dos de los productos financieros de mayor interés práctico son el préstamo y el plan de pensión. En el préstamo un cliente recibe de un banco un capital que permite financiar la adquisición de un bien (coche, casa) o servicio y se compromete a devolverlo, junto con el interés compuesto que genera, en un plazo de tiempo fijado mediante el pago periódico de una cuota llamada de amortización. En el plan de pensión un cliente se compromete a depositar en un banco de forma periódica un dinero o aportación que podrá recuperar, junto con el interés compuesto que genera, a partir del momento de su jubilación, en pago único o en pagos periódicos vitales. La aportación es la cuota de capitalización. Cuando las cuotas son anuales (mensuales) se llaman anualidades (mensualidades). Las fórmulas son: 1) Para el préstamo $D = a \cdot \frac{(1+i)^t - 1}{i \cdot (1+i)^t}$ 2) Para el plan de pensiones $C = a \cdot (1+i) \cdot \frac{(1+i)^t - 1}{i}$ donde *a* la cuota periódica, *i* el tipo de interés en tanto por uno, *t* el tiempo, *D* y *C* el capital. El tiempo y el tipo de interés han de referirse a la misma periodicidad que la cuota. Si la cuota es anual han de expresarse en años, si la cuota es mensual en meses. Ejemplo: Calcular la cuota mensual de un préstamo de 120.000 € al 4% de interés anual a devolver en 20 años. La fórmula sustituida sería $120000 = a \cdot \frac{(1+\frac{4}{1200})^{20 \cdot 12} - 1}{\frac{4}{1200} \cdot (1+\frac{4}{1200})^{20 \cdot 12}}$. Obsérvese que i=4/1200 es un tanto por uno mensual y que t=20·12 son meses. Despejando a vale 727'18 €. Es decir, para devolver 120.000 € en 20 años a un 4% de interés hay que pagar 240 cuotas mensuales de 727'18 €. Habremos devuelto al banco 174.522,33 €.
OBJETIVO	Elaborar un programa que dependiendo de si se desea capitalizar o amortizar solicite los datos para calcular cuotas de préstamos y aportaciones de planes de pensiones. V1. Calcula cuotas a partir de capital, tipo de interés, periodicidad y plazo.

V2. Calcula capital a partir de cuotas, plazos, periodicidad y tipo de interés.

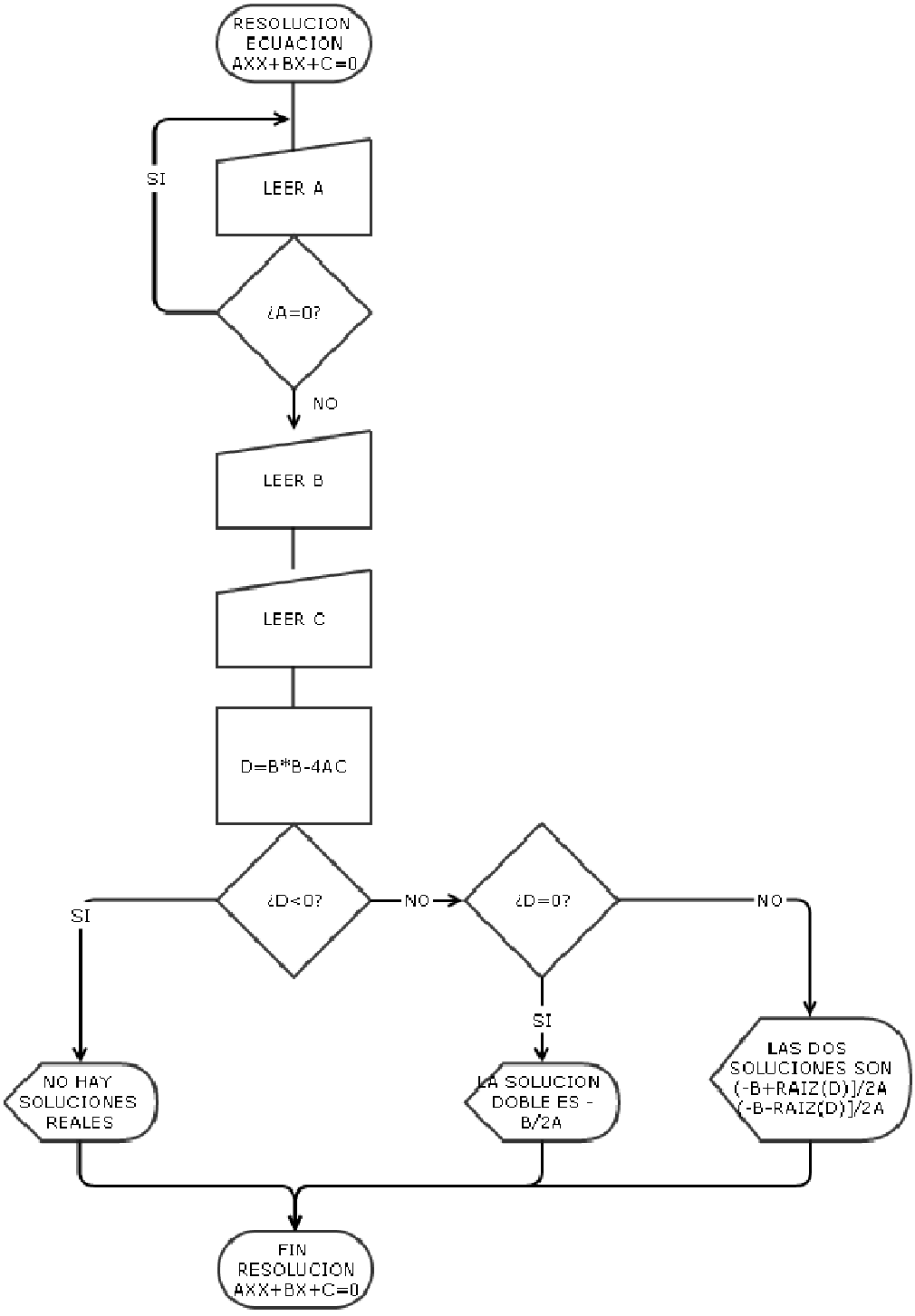
Las fórmulas de las anualidades despejadas son:

1) Para el préstamo $a = \frac{D \cdot i \cdot (1+i)^t}{(1+i)^t - 1}$

2) Para el plan de pensión $a = \frac{C \cdot i}{(1+i) \cdot [(1+i)^t - 1]}$.

P75	Resolución de la ecuación de 2º grado
	Uno de los problemas más frecuentes en matemáticas de secundaria es la resolución de una ecuación de 2º grado. Su forma general es $ax^2+bx+c=0$ con $a\neq 0$. Su solución general es $x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. El número de soluciones depende del valor del discriminante $D=b^2-4ac$: 2, si es positivo; 1 (doble), si es cero y ninguna, si es negativo (no existe la raíz cuadrada real de un número negativo).
	Elaborar un programa que solicite el valor de a, b y c y resuelva la ecuación $ax^2+bx+c=0$ con $a\neq 0$.
	☞ Asegúrate de que $a\neq 0$. ☞ Calcula el discriminante D para informar el número de soluciones.

Un diagrama de flujo con la solución es:



P76	Búsqueda binaria en una lista ordenada
	En una búsqueda secuencial se comienza con el primer elemento y se busca en ella hasta encontrar el elemento deseado o alcanzar el final. De este modo, si tuviéramos que localizar el número de abonado en un guía de teléfonos de una gran ciudad el tiempo de búsqueda se podría eternizar. La búsqueda binaria utiliza el método "divide y vencerás" para localizar el valor deseado. Se examina primero el elemento central de la lista. Si es el buscado la búsqueda ha terminado. Si no lo es, se determina si es el buscado es anterior o posterior al central reduciendo la búsqueda a la primera o a la segunda mitad de la lista original. A continuación se reproduce el proceso con la mitad seleccionada.
OBJETIVO	Elaborar un programa que informe si un valor está o no está en una lista ordenada usando búsqueda binaria. Si está debe informar la posición.
	Crea la lista solicitando datos en pantalla y usando el P42 para ordenarla. BYOB tiene una función que hace esto mismo.¿Cuál?  ¿Funcionará igual que nuestro algoritmo?

Pseudocódigo UPSAM 2.0 con la solución es:

```
algoritmo búsqueda_binaria
//Búsqueda en una lista ordenada X de tamaño N
//K es el valor buscado
//X[CENTRAL] es el elemento que ocupa la posición CENTRAL
//de la lista X
//ent(x) es la función parte entera de x.
leer(K)
BAJO←1
ALTO←N
CENTRAL← ent((BAJO+ALTO)/2)
mientras (BAJO <= ALTO) y (X[CENTRAL] <> K) hacer
   si    K< X[CENTRAL] entonces
     ALTO←CENTRAL-1
   si_no
     BAJO←CENTRAL+1
   fin_si
   CENTRAL← ent((BAJO+ALTO)/2)
fin_mientras
si K = X[CENTRAL] entonces
   escribir('Valor encontrado en', CENTRAL)
si_no
   escribir('Valor no encontrado')
fin_si
fin
```

P77	Devolver cambio con monedas
	Estando disponibles un número indefinido de monedas de 100, 50, 20, 10, 5, 2 y 1 céntimo un cajero utiliza de forma inconsciente, sin pensarlo dos veces, el siguiente algoritmo para devolver cierta cantidad a un cliente: comenzamos con nada y en cada fase añadimos al dinero devuelto una moneda de la mayor denominación posible sin que se exceda la cantidad a pagar. Ejemplo: para pagar 2'89 € seleccionamos la siguiente secuencia de monedas 100, 100, 50, 20, 10, 5, 2 y 2. Este algoritmo con un suministro limitado de alguna moneda o menos denominaciones deja de funcionar. Ejemplo: para pagar 2'13 € con monedas de {100, 50, 20, 5, 2} seleccionamos la secuencia de monedas 100, 100, 5, 5, 2 y falta 1. Para ello hace falta un algoritmo "dinámico" más complicado (véase P78).
	Elaborar un programa que especifique el conjunto de monedas de que disponemos y cómo pagar cualquier cantidad de céntimos introducida.
	☞ Solicita primero las monedas disponibles. ☞ Echa un vistazo al pseudocódigo solución.
	Pseudocódigo **función** devolver_cambio(N) **const** C={100,50,20,10,5,2,1} S←∅ s←0 **mientras** s<>N **hacer** x←el mayor elemento de C tal que x+s<=N **si** no existe ese elemento **entonces** **devolver** "No encuentro la solución" S←S ∩ {una moneda de valor x} s←s+x **fin_mientras** **devolver** S Notas C y S son listas. La operación ∩ es añadir a la lista.  ¿Cómo hago para determinar x?

P78 Devolver cambio con monedas (y 2)

En nuestro afán de pagar al cliente cierta cantidad N, empleando el menor número de monedas posible, prepararemos una tabla con resultados intermedios útiles que serán combinados en la solución del caso considerado. Consideraremos n denominaciones diferentes. Llamaremos d_i al valor de la denominación i. Disponemos de suministro ilimitado. Prepararemos una tabla con una fila para cada denominación posible y una columna para cada cantidad desde 0 hasta N. La tabla tiene dimensión n·N.

El elemento c(i,j) de la tabla contendrá el número mínimo de monedas para pagar la cantidad j *empleando solamente las denominaciones hasta la i*.

De esta forma el número mínimo de monedas será c(n,N).

Ejemplo: n=3, d_1=1, d_2=2, d_3=5, N=10. La tabla a construir será:

Cantidad	0	1	2	3	4	5	6	7	8	9	10
d_1=1	0	1	2	3	4	5	6	7	8	9	10
d_2=2	0	1	1	2	2	3	3	4	4	5	5
d_3=5	0	1	1	2	2	1	2	2	3	3	2

Pero ¿cómo rellenamos la tabla?

1) La primera columna corresponde a no devolver cantidad alguna: la rellenaremos de ceros.
2) El resto lo rellenaremos bien por filas o bien por columnas. Tenemos dos opciones:
   A) Decidir no utilizar monedas de denominación i. Queda todo por pagar. Debe ser c(i,j)=c(i-1,j).
   B) Emplear al menos una moneda de denominación i. Quedarán por pagar $j-d_i$. Se necesitarán $c(i,j-d_i)$.
   Luego $c(i,j)=1+ c(i,j-d_i)$.
3) Como queremos minimizar el número de monedas finalmente haremos que c(i,j) sea el valor mínimo de ambas opciones: $\min[1+ c(i,j-d_i), c(i-1,j)]$.

Precaución. Cuando i=1 uno de los elementos a comparar cae fuera de la tabla. Lo mismo sucede cuando $j<d_i$. Supondremos que vale infinito. Si ocurren ambas circunstancias estamos en la situación imposible de pagar una cantidad j solamente con monedas de la 1ª denominación.

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

Elaborar un programa que especifique el conjunto de tipos de monedas de que disponemos y cómo pagar cualquier cantidad de céntimos introducida.

☞ Echa un vistazo al pseudocódigo solución.

Pseudocódigo

```
función devolver_cambio(N)
vector d[1..n]={100,50,20,5,2}
matriz c[1..n,0..N]
desde i←1 hasta n hacer
   c[i,0]←0
fin_desde
desde i←1 hasta n hacer
     desde j←1 hasta N hacer
       si      i=1 y j<d[i] entonces c[i,j] ← +∞
       si_no  si i=1 entonces c[i,j]←1+c[i,j-d[1]]
       si_no  si j<d[i] entonces c[i,j]←c[i-1,j]
       si_no c[i,j]←min(c[i-1,j], 1+c[i,j-d[i]])
     fin_desde
fin_desde
devolver c[n,N]
```

Notas

Un vector es una lista. Una matriz es una lista de listas.

P79	Comprobación probabilística de primalidad

No se conoce ningún algoritmo para resolver con certeza y en un tiempo razonable si un entero impar grande es primo. Este hecho es crucial para la criptografía.

Un algoritmo probabilístico o algoritmo de Monte Carlo es aquel que al enfrentarse a una decisión sigue un curso aleatorio en lugar de invertir tiempo en averiguar cual de las alternativas es la mejor.

La historia de la comprobación probabilística de primalidad comienza con Pierre de Fermat quien en 1640 demostró su teorema menor:

"Sea n un número primo. Entonces a^{n-1} mod n = 1 para cualquier entero positivo a menor que n".

El propio Fermat, en su búsqueda de fórmulas que produjesen números primos, propuso que $F_n = 2^{2^n} + 1$ era primo. El quinto número F_5=4294967297 era demasiado grande para que Fermat demostrara su primalidad. Si Fermat hubiera podido calcular que $3^{F_5-1} \mod F_5 = 3029026160 \neq 1$ podría haber sabido que F_5 no es primo. De hecho Euler, casi un siglo después, lo factorizó: F5=641·6700417.

Es decir, se puede usar la versión contrapositiva del teorema menor de Fermat para saber que un número grande es compuesto.

"Si a es un entero positivo menor que n y a^{n-1} mod ≠ 1 entonces n no es primo, es decir, n es compuesto".

Lo que inspira el siguiente algoritmo probabilista

```
función Fermat(n)
a←aleatorio(1,n-1)
si expomod(a,n-1,n)=1 entonces devolver cierto
si_no devolver falso
```

Si llamamos a Fermat(n) y nos devuelve falso sabemos que n es compuesto. Pero si nos devuelve verdadero no podremos afirmar que n es primo pues existen los llamados **testigos falsos de primalidad**, p.e, 4^{14} mod 15 = 1 a pesar de que 15 es compuesto.

Existe una mejora llamada algoritmo de Miller-Rabin que reduce el número de testigos falsos. **Siempre** devuelve cierto cuando n es primo y devuelve falso con una probabilidad superior al 75% de que sea compuesto.

Rabin y Pratt obtuvieron con su algoritmo que $2^{500}-153$ y $2^{400}-593$ eran primos y hallaron los primos gemelos de 123 cifras $338P+821$ y $338P+823$ (P es el producto de todos los primos menores que 300).

Necesitaremos definir la función

```
función pruebaB(a, n)
s←0
t←n-1
repetir
   s←s+1
   t←t/2
hasta que t mod 2 = 1
x←expomod(a, t, n)
si x=1 o x=n-1 entonces devolver cierto
desde i←1 hasta s-1 hacer
    x←x² mod n //En cada una de las s-1 iteraciónes eleva x al cuadrado
fin_desde
si x=n-1 entonces devolver cierto
devolver falso
```

y finalmente el algoritmo es

```
función MillerRabin(n)
a←aleatorio(2,n-2)
devolver pruebaB(a,n)
```

cuya reiterada llamada reduce la probabilidad de error, es decir, que un compuesto devuelva cierto, a $1-4^{-k}$:

```
función RepetirMillerRabin(n, k)
desde i←1 hasta k hacer
si MillerRabin(n) falso entonces devolver falso
fin_desde
devolver cierto
```

Elaborar un programa que señale la probable primalidad o no de un natural grande usando la función Fermat.

V2. Idem usando la reiteración de Miller-Rabin (n>4).

Echa un vistazo al pseudocódigo y codifica todas las funciones.


```
función expomod(a, n, z)
//Calcula aⁿ mod z
i←n, r←1, x←a
mientras i>0 hacer
   si i es impar entonces r←a mod z
   x←x² mod z
   i←i/2
fin_mientras
devolver r
```

P80	Generar un número grande probablemente primo
	Si deseamos seguridad encriptando necesitamos números primos grandes. En P79 hemos hecho casi todo el trabajo para poder comprobar la muy probable primalidad de un número impar. Nos falta simplemente seleccionar impares aleatorios de muchos dígitos que pasen muchas veces la comprobación de Miller-Rabin. Con todo, el algoritmo que se propone puede devolver un número compuesto con la misma o mayor facilidad que un número primo. Hay que tener en cuenta que la densidad de los números primos viene dada por la fórmula $\pi(x) \approx \frac{x}{\log x}$ siendo π(x)=número de primos menores o iguales que x. Es decir si un número tiene 1000 cifras su logaritmo decimal valdrá 1000 y habrá un nº primo de cada mil. Como 4^5=1024, es decir aproximadamente 1000, el error de RepetirMillerRabin(1000,5) es igual de probable y podemos obtener un número primo y un falso primo con idéntica probabilidad. La clase Java BigInteger tiene un constructor que permite obtener un número probablemente primo del tamaño que se desee en un instante. Invocar esta clase es equivalente al programa que pretendemos realizar en este proyecto.
	Elaborar un programa que genere un entero impar probablemente primo del número de dígitos indicado.
	 Echa un vistazo al pseudocódigo.

	**función** primoaleatorio(l, k)
	// l es el número de dígitos que se desea tenga el probable primo
	// k es el número de repeticiones que hace que un determinado número
	// sea primo con un probabilidad de error de $1-4^{-k}$
	repetir
	n←1+2·aleatorio($10^{l-1}/2$, $10^{l}/2-1$)
	hasta que RepetirMillerRabin(n, k) cierto
	devolver n

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

P81	Algoritmo de Maurer para certificar que un número es primo de forma probabilística.
	EL algoritmo de Maurer genera un primo certificado de forma probabilística y requiere, según su autor, algo más de tiempo de ejecución que el de Miller-Rabin, que genera un número probablemente primo.
	Elaborar un programa que genere un entero impar primo del número de dígitos indicado siguiendo el algoritmo de Maurer.
	Echa un vistazo al pseudocódigo del artículo *A study of Maurer's algorithm for finding provable primes in relation to the Miller-Rabin algorithm de* Mathias Romme Schwarz y Sören Grönnegaard Andersen (2007).

Algorithm 2 maurer_fast_prime(int k)

Require: $k \in Z_+$

```
if k < 20 then
  repeat
3:  n ∈ [2, 2^k]
    isPrime = true
    for all primes p ≤ √n do
6:    if n ≡ 0 (mod p) then
        isPrime = false
    until isPrime
9:  return n
else
    c = 0.1, m = 20
12: B = c · k^2
    if k > 2m then
      repeat
15:     s ∈ [0, 1]
        r = 2^(s−1)
      until k − rk > m
18: else
      r = 0.5
    q = maurer_fast_prime(⌊r · k⌋)
21: I = ⌊2^(k−1) / 2q⌋
    succes = false
    while ¬success do
24:   repeat
        R ∈ [I + 1, 2I]
        n = 2Rq + 1
27:     mayBePrime = true
        for all primes p ≤ B do
          if n ≡ 0 (mod p) then
30:         mayBePrime = false
      until mayBePrime
      a ∈ [2, n − 2]
33:   b = a^(n−1)
      if b = 1 then
        b ≡ 2^(2R) (mod n)
36:     d = gcd(b − 1, n)
        if d = 1 then
          success = true
39: return n
```

Erratas, aclaraciones y sugerencias:

En la línea 3 debería decir $n \in [2^{k-1}, 2^k]$.

En la línea 34 debería decir **if** $b \equiv 1 \pmod{n}$ **then.**

$\in$ significa un número aleatorio dentro del intervalo señalado.

Las líneas 2 a 10 devuelven un primo n cuando tiene hasta 20 dígitos binarios (n<1048576).

Las líneas 13 a 18 forman una rutina que calcula r en función de k y m

Las líneas 24 a 31 forman una rutina que propone n usando I, q y B.

Reescribimos el algoritmo.

```
maurer_fast_prime (int k)
si k<20 entonces devolver primo_pequeño(k)
si_no
   c=0.1, m=20, B=c·k²
   calcula_r(k,m)
   q=maurer_fast_prime (r·k)// llama con el entero anterior a rk
   I=2^(k-1)/2q                  // el entero menor que 2^(k-1)/2q
   success=falso
   mientras success sea falso hacer
      n=proponer_n(I,q,B)
      a=numero aleatorio(2,n-2)
      b=a^(n-1)
      si b=1 mod n entonces
         b=2^((n-1)/q) mod n
         d=mcd(b-1,n)
         si d=1 entonces success=cierto
         si_no success=falso
      si_no success=falso
   fin_mientras
devolver n
```

P82	Algoritmo de Agrawal, Kayal y Saxena (AKS-2002) para certificar que un número es primo de forma determinista.
OBJETIVO	Elaborar un programa que compruebe si un entero impar grande es primo o compuesto de forma determinista siguiendo el algoritmo AKS.
	☞ Echa un vistazo al pseudocódigo publicado en el artículo *El algoritmo de Agrawal, Kayal y Saxena para decidir primalidad* por los matemáticos mexicanos José de Jesús Ángel Ángel y Guillermo Morales-Luna.

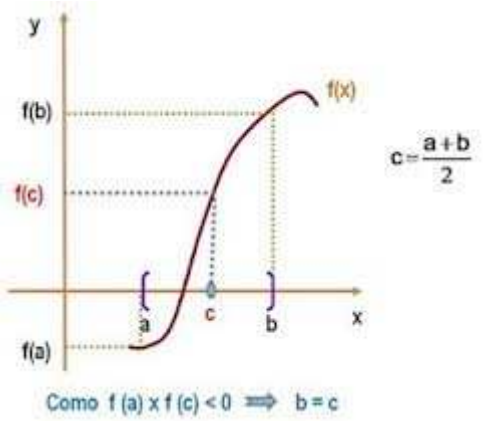
Input: $n \geq 1$
Output: A decision whether n is PRIME or COMPOSITE

1.- If $(n = a^b)$, with $a \in \mathbb{N}$, $b > 1$, then
output COMPOSITE
2.- else $r = 2$
3.- While $(r < n)$ do {
4.- If (mcd $(n, r) \neq 1$) then
output COMPOSITE
5.- else If (r is prime) then
6.- let q be the largest prime factor of $r - 1$
7.- If $(q \geq 4\sqrt{r}\log n)$ and $n^{\frac{r-1}{q}} \not\equiv 1 \bmod r$
8.- then break:
9.- $r := r + 1$
10.- }
11.- For $a = 1$ to $2\sqrt{r}\log n$ do
12.- If $(x - a)^n \not\equiv (x^n - a) \bmod (x^r - 1, n)$ then
output COMPOSITE
13.- else output PRIME

Tabla 1: Algoritmo AKS-2002

P83	Resolución numérica de ecuaciones. Método de la bisección.

El teorema de Bolzano establece que si una función f(x) es continua en [a, b] y cumple que f(a)f(b)<0 entonces existe un punto intermedio s en (a, b) donde f(s)=0.
Este punto s es una solución de la ecuación f(x)=0.



El algoritmo de la bisección divide el intervalo original por su punto medio, digamos c, y a continuación examina entre qué nuevos extremos persiste el cambio de signo: ¿f(a)f(c)<0? ó ¿f(c)f(b)<0? Si se da f(a)f(c)<0 el nuevo intervalo será [a,c]. Si se da f(c)f(b)<0 el nuevo intervalo será [c,b]. Y volvemos a aplicar el teorema hasta "aislar" el punto (s) donde valdrá "casi cero".

OBJETIVO

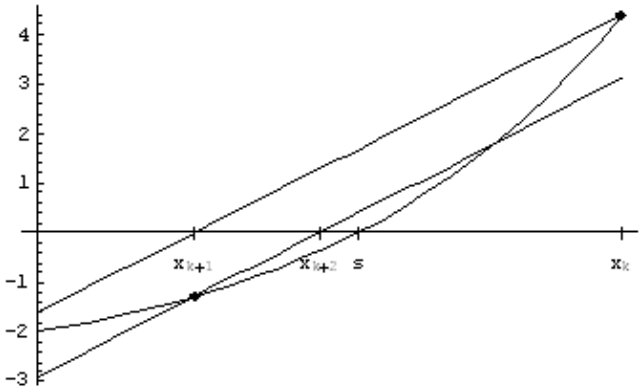
Elaborar un programa que encuentre una solución de la ecuación f(x)=0 en el intervalo [a, b] cumpliendo f(x) las condiciones del teorema de Bolzano.

Echa un vistazo al pseudocódigo

```
Función biseccion(f, a, b, tol, d)
//f es la función continua
//[a,b] el intervalo donde f(a)f(b)<0
//tol es la tolerancia o variación mínima de la x
//Si tol=10^-4 la solución tendrá seguro 3 cifras decimales exactas.
//d el valor de f(x) mínimo que asimilamos como aproximadamente cero.
mientras b-a>=tol hacer
    c=a+(b-a)/2
    si abs(f(c))<d entonces devolver c
    si_no
        si f(a)f(c)<0 entonces b=c
        si_no         entonces a=c // f(a)f(c)>0
fin_mientras
```

¿Por qué nos basta con que -d < f(c) < d si d es un número positivo muy pequeño, por ejemplo, una millonésima?

P84	Resolución numérica de ecuaciones. Método de Whittaker y de Newton-Raphson.



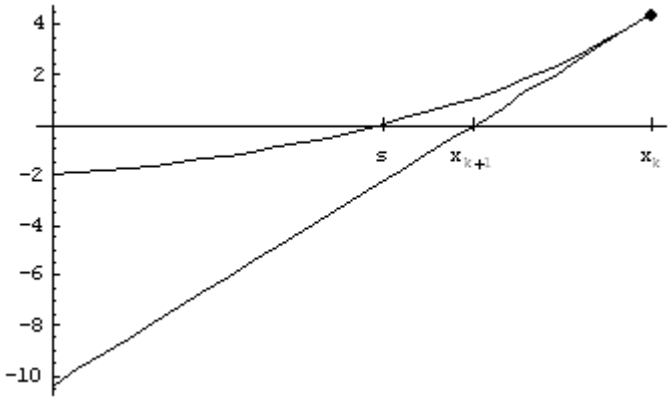
Idea geométrica del método de Whittaker

En el método de Whittaker f es continua, se parte de un valor inicial arbitrario x_0 y se aproxima la raíz de la ecuación $f(x)=0$ mediante la abscisa del correspondiente punto de corte con el eje 0X de la recta que pasa por los sucesivos puntos $(x_k, f(x_k))$ y que tiene una pendiente predeterminada y fija m.

Las abscisas de los puntos se calculan así:

$$x_{k+1}=x_k-f(x_k)/m.$$

Nota: si hacemos m variable con valor $f'(x_k)$ se tiene el método de Newton-Raphson.



Idea geométrica del método de Newton-Raphson

Elaborar un programa que encuentre una solución de la ecuación $f(x)=0$ usando el método de Whittaker con m=10 y valor inicial x_0.

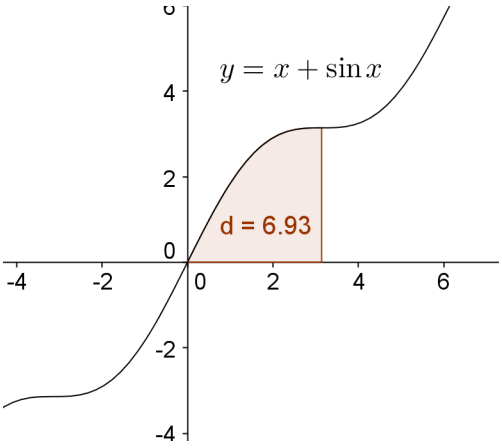
V2. Añade un contador de iteraciones e informa de los valores de tol y d.

Echa un vistazo al pseudocódigo solución

```
Función whittaker(f, x0, m, tol, d)
//m es una pendiente arbitraria pero fija. Ej: 10.
//tol es la tolerancia o variación mínima de la x
//Si tol=10-4 la solución tendrá seguro 3 cifras decimales exactas.
//d el valor de f(x) mínimo que asimilamos como aproximadamente cero.
x1=0
Mientras abs(x1-x0)>=tol hacer
   x1=x0-f(x0)/m
   si abs(x1)<d entonces devolver x1
   si_no
      x0=x1
fin_mientras
```

P85	Calculo de la integral definida. Fórmulas de Newton-Cotes.

Dada una función continua y positiva $f(x)$ en el intervalo $[a,b]$ el área de la figura plana, comprendida entre la gráfica de $f(x)$, el eje de abscisas y las rectas verticales $x=a$ y $x=b$, es la integral definida $\int_a^b f(x)\cdot dx$.



Queremos calcular el área que encierra la curva f(x)=x+sen x, el eje 0X y las rectas x=0 y x=π. El cálculo simbólico de la integral definida se hace aplicando la regla de Barrow a la integral indefinida en los extremos de integración.

$$\int_0^{\pi} (x+\operatorname{sen}\, x)\cdot dx = \frac{\pi^2}{2}+2$$

Este es el valor del área sombreada que en la figura se informa que es 6.93.

Las fórmulas de Newton-Cotes nos permiten aproximar el área buscada utilizando valores de la función en puntos equidistantes multiplicados por ciertos coeficientes.

Fórmulas de Newton-Cotes cerradas.

$$\int_a^b f(x)dx = \frac{b-a}{D}\sum_{i=0}^{n} \alpha_i \cdot f(x_i) + R_f((a,b))$$

n	α_i (j=0, …, n)	D	$R_f((a,b))$	Nombre
1	1 1	2	$(h^3/12)\cdot f''(\xi)$	Trapecio
2	1 4 1	6	$(h^5/90)\cdot f^{(iv)}(\xi)$	Simpson
3	1 3 3 1	8	$(3h^5/80)\cdot f^{(iv)}(\xi)$	Regla 3/8
4	7 32 12 32 7	90	$(8h^7/945)\cdot f^{(vi)}(\xi)$	Milne
5	19 75 50 50 75 19	288	$(275h^7/12096)\cdot f^{(vi)}(\xi)$	
6	41 216 27 272 27 216 41	840	$(9h^9/1400)\cdot f^{(viii)}(\xi)$	Weddle

Por ejemplo, la regla 3/8 (n=3) dice que

$$\int_a^b f(x)dx = \frac{b-a}{8}[1\cdot f(x_0)+3\cdot f(x_1)+3\cdot f(x_2)+1\cdot f(x_3)]$$

siendo $x_0=a$, $x_3=b$, $x_1=a+(b-a)/3$, $x_2=a+2\cdot(b-a)/3$

es decir $x_i=a+i\cdot(b-a)/n$. R_f es una cota del error cometido.

PROGRAMACIÓN ELEMENTAL CON SCRATCH 1.4/BYOB 3.1

Programa el cálculo de la integral definida de una función f(x) entre a y b mediante la regla del trapecio, la regla de Simpson, la regla 3/8, la regla de Milne y la regla de Weddle.

Comprueba tus resultados aplicándolos a f(x)=x+sen x en [0,π].

Echa un vistazo al pseudocódigo solución

```
Función tres_octavos(f, a, b)
D←8, n←3
I←0
coef←{1,3,3,1} //coef es una lista de n+1 elementos
desde i←0 hasta n hacer
   x←a+i·(b-a)/n
   I=I+coef(i+1)·x
fin_desde
devolver I
```

¿Qué modificarías en el pseudocódigo para obtener la regla del trapecio, la de Simpson, la de Milne y la de Weddle?

P86	Combinatoria
	En cálculo de probabilidades se emplea la regla de Laplace $P(S)=\frac{\text{casos favorables}}{\text{casos posibles}}$. Tanto el numerador como el denominador requieren contar las posibles realizaciones de un suceso, asunto que resuelve el cálculo combinatorio. Se llaman **permutaciones** de m elementos a cualquier ejemplar que los contiene todos. Si no se repiten elementos el ejemplar los tendrá todos (m) y se diferenciará de otro ejemplar por el orden en que aparecen. Se calculan con la fórmula: $P_m=m!$ Ej: Palabras de 5 letras con las de CLASE. Hay 5!=5·4·3·2·1=120 Si un elemento se repite r veces y otro s veces la fórmula es $P_m^{r,s}=\frac{m!}{r!s!}$ Ej: palabras de 5 letras con las letras de CLASS. Hay 5!/2!=5·4·3=60 Cuando de un conjunto con m elementos formamos un subconjunto de n elementos no repetidos hablaremos de **variación**, si una ordenación diferente los hace diferentes, o de **combinación**, en caso contario (distinto orden es el mismo ejemplar). Sus fórmulas son: $V_{m,n}=\frac{m!}{(m-n)!}$ y $C_{m,n}=\frac{m!}{n!(m-n)!}=\binom{m}{n}$ Ej: Podium de una carrera atlética de velocidad: $V_{8,3}=8\cdot 7\cdot 6=336$ Ej: Apuestas distintas de la lotería primitiva: $C_{49,6}=\binom{49}{6}=13983816$ Si los ejemplares pueden contener elementos repetidos hablaremos de variaciones con repetición y combinaciones con repetición. Sus fórmulas son: $VR_{m,n}=m^n$ y $CR_{m,n}^{r,s}=VR_{m,n}/P_m^{r,s}$. Ej: Apuestas distintas de la quiniela de fútbol: $VR_{3,15}=3^{15}=14348907$
	Elabora un programa que ayude a resolver un problema de combinatoria aplicando la fórmula adecuada.
	Pregunta primero si cada ejemplar debe contener todos los elementos (m=n, permutaciones). En caso de que no (n<m), a continuación pregunta si el orden los hace diferentes (variaciones) o no (combinaciones). Por último, hay que saber si se pueden repetir o no, es decir, cuánto vale r, s, … en caso de que sí se repitan.

P87	Las torres de Hanoi

El problema de las torres de Hanoi es un problema clásico de recursión propuesto por el matemático francés Édouard Lucas en 1883. Se tienen 3 torres y un conjunto de discos de diferentes tamaños. Cada disco tiene una perforación en el centro que le permite ensartarse en cualquiera de las tres torres. Los discos han de encontrarse siempre situados en alguna de las torres.

Inicialmente todos están en la misma, ordenados de mayor a menor, como se muestra en el dibujo.

1:origen 2:destino 3:auxiliar

Se deben averiguar los movimientos necesarios para pasar todos los discos a otra torre, utilizando una tercera como auxiliar y cumpliendo las siguientes reglas:

- En cada movimiento sólo puede intervenir un disco
- No puede quedar un disco sobre otro de menor tamaño

Para 3 discos la solución es mover el disco superior de 1 a 2, de 1 a 3, de 2 a 3, de 1 a 2, de 3 a 1, de 3 a 2, de 1 a 2.

OBJETIVO

Escribir un programa que resuelva el problema de las torres de Hanoi para n discos (n>2).

Este es el pseudocódigo de Joyanes en su libro

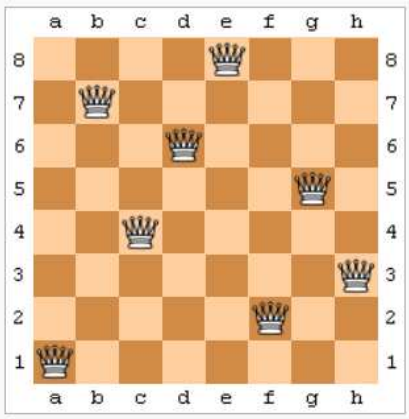
```
algoritmo  Hanoi
var
   entero: n
inicio
   escribir('Nº de discos:')
   leer(n)
   llamar_a  Mover_torre(n,1,2,3)
fin

procedimiento Mover_torre( E entero: N ;   E entero: orig,dest,aux)
inicio
   si N = 1 entonces
      escribir('Paso de ', orig,' a ', dest )
   si_no
      llamar_a Mover_torre ( N-1, orig, aux, dest )
      escribir('Paso de ', A,' a ', B )
      llamar_a Mover_torre ( N-1, aux, dest, orig )
   fin_si
fin_procedimiento
```

Errata: la última instrucción escribir debe decir:

escribir('Paso de ', orig, 'a ', dest)

P88 El problema de las ocho reinas

Fue propuesto por el ajedrecista alemán Max Bezzel en 1848. Disponer ocho reinas en un tablero de ajedrez 8x8 de manera que no se amenacen entre si (una reina amenaza a las piezas situadas en su misma fila, columna o diagonal). El problema se puede generalizar a cualquier número natural n de reinas en un tablero nxn.



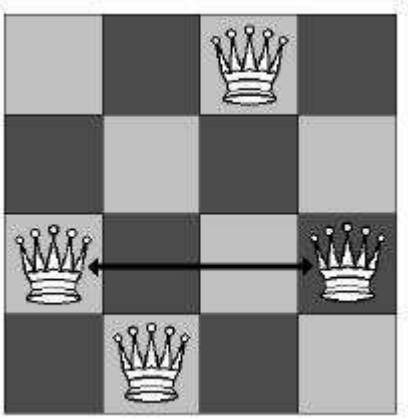
Solución única 2

En el caso de n=8 tiene 12 soluciones básicas, que con rotaciones, simetrías y traslaciones son 92.
En los casos n=2 y n=3 no hay solución.
El dibujo muestra la solución que coloca la primera reina en (a,1)

OBJETIVO

Elaborar un programa que resuelva el problema de las n reinas solicitando el valor de n.

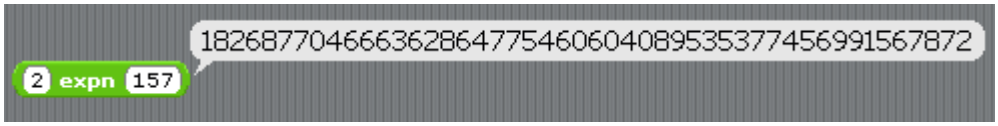
Usa una lista de n elementos con valores iniciales a cero. El número de orden del elemento representará la fila donde está la reina. El contenido representará la columna donde está la reina. El movimiento inicial puede ser {1,0,0,0,0,0,0,0} que significa que hay una única reina situada en la fila 1, columna 1.


Ejemplo de reinas amenazadas

Si intentamos situar ahora una segunda reina:
{1,1,0,0,0,0,0,0} no es válido porque ambas están en la *misma columna*.
{1,2,0,0,0,0,0,0} no es válido porque ambas están en la *misma diagonal*.

Este es el pseudocódigo de Joyanes en su libro:

```
algoritmo Ejericio_5_24
const n = 8
tipos array de entero[1..n] : ListaDamas

var
   ListaDamas : D
   entero : i
   lógico : solución
inicio
   // Inicializar el array
   desde i ← 1 hasta n hacer
      d[i] ← 0
   fin_desde
   Ensayar(d,1,solución)
   si no solución entonces
      escribir('No hay solución')
   si_no
      // se deja al lector hacer la presentación del resultado
   fin_si
fin

lógico función PosiciónVálida(E ListaDamas d ;  E entero : i)
var
   entero : j
   lógico : válida
inicio
   válida ← verdad
   desde j ← 1 hasta i - 1 hacer
      // No se ataca en la columna
      válida ← válida y (d[i] <> d[j])
      // no se ataca en una diagonal
      válida ← válida y (d[i] + i <> d[j] + j )
      // no se ataca en la otra diagonal
      válida ← válida y (d[i] - i <> d[j] - j )
   fin_desde
   devolver(Válida)
fin_función

procedimiento Ensayar( E/S ListaDamas : d ; E entero : i ;
                       S lógico  : Solución)
inicio
   si i = n + 1 entonces
      solución ← verdad
   si_no
      solución ← falso
      repetir
         d[i] ← d[i] + 1
         si PosiciónVálida(d,i) entonces
            Ensayar(d,i+1,solución)
      hasta_que solución o  (d[i] = n)
      si no solución entonces
         d[i] ← 0
      fin_si
   fin_si
fin_procedimiento
```

P89	Números apocalípticos
	Un número de la forma 2^n que contiene los dígitos 666, es decir, que contiene al número de la bestia, se llama número **apocalíptico**. El número de la bestia es igual a la suma de los cuadrados de los siete primeros números primos: $$2^2+3^2+5^2+7^2+11^2+13^2+17^2=666,$$ El número de la bestia coincide con la concatenación de los símbolos de la numeración romana: $DCLXVI=666$ 2^{157} es un número apocalíptico como puedes observar en la siguiente imagen  Las primeras potencias de dos que producen números apocalípticos son 157, 192, 218, 220.
	Elabora un programa que determine todos los números apocalípticos cuya potencia sea inferior a 1000.
	 Utiliza la función  de exponenciación rápida pedida en el proyecto P38.  Elabora una función que responda cierto si la variable contiene la secuencia de "letras" 666.
	La sucesión de las potencias solución es la secuencia de Sloane A007356.

P90	Números económicos
	Un número entero positivo n se dice **económico** si el número de dígitos de su descomposición como producto de primos elevados a potencias es inferior al número de dígitos del número n. Ejemplo: 125 tiene tres dígitos y el producto de sus factores primos (como potencias) sólo 1 (5^3). Los primeros números económicos son 125, 128, 243, 256, 343, 512, 625, 729, ...
	Elabora un programa que determine si un entero positivo es económico. V2 Compón una lista con los números económicos menores que 10000.
	 Utiliza P3 para descomponer y elabora una rutina que cuente los dígitos de la descomposición (número de factores primos distintos)
	Puedes consultar la sucesión completa en la secuencia de Sloane A046759.

P91	Números felices
	Dado un entero positivo s_0 sea s_1 la suma de los cuadrados de sus dígitos. De forma similar s_2 será la suma de los cuadrados de los dígitos de s_1 y así sucesivamente. Reiterando este proceso se llega, antes o después, a uno de estos 10 números: 0, 1, 4, 16, 20, 37, 42, 58, 89 ó 145. Si llegamos al 1 el número s_0 del cual partimos se dice número **feliz**. Ejemplo: 7 es feliz pues $s_0=7$, $s_1=7^2=49$, $s_2=4^2+9^2=97$, $s_3=9^2+7^2=130$, $s_4=1^2+3^2+0^2=10$, $s_5=1^2+0^2=1$, $s_6=1=s_7$ … Los primeros números felices son 1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100, …
	Elabora un programa que determine si un entero positivo es feliz. V2 Compón una lista con los números felices menores que 10000.
	 Elabora una lista con los números que han de detener la repetición de la suma de cuadrados de dígitos. Si la suma obtenida está contenida en la lista podremos averiguar si el número del que proviene es o no feliz.
	Puedes consultar la sucesión completa en la secuencia de Sloane A007770

P92	Números super-d
	El número 261 se dice que es un número super-tres porque $3 \cdot 261^3 = 53338743$ contiene 3 *treses* consecutivos. Es decir, un número n es **super-tres** si la expresión decimal de $3 \cdot n^3$ contiene 3 *treses* consecutivos. De forma análoga se define un **número super-d** como aquel que verifica que la expresión decimal de $d \cdot n^d$ contiene d números d consecutivos.
	Elabora un programa que determine si un entero positivo es super-dos. Ej: 19 es super-dos porque $2 \cdot 19^2 = 722$ contiene 2 doses consecutivos. V2 Compón una lista con los números super-tres menores que 10000.
	☞ Reutiliza las funciones de P89.
	Tabla de los primeros números super-d para d pequeño. d Sloane Números super-d 2 A032743 19, 31, 69, 81, 105, 106, 107, 119, ... 3 A014569 261, 462, 471, 481, 558, 753, 1036, ... 4 A032744 1168, 4972, 7423, 7752, 8431, 10267, ... 5 A032745 4602, 5517, 7539, 12955, 14555, 20137, ... 6 A032746 27257, 272570, 302693, 323576, ... 7 A032747 140997, 490996, 1184321, 1259609, ... 8 A032748 185423, 641519, 1551728, 1854230, ... 9 A032749 17546133, 32613656, 93568867, ...

P93	Algoritmo 196
	Considera un número positivo de 2 o más dígitos, invierte el orden de sus cifras y súmalo al número original. La reiteración de esta operación de sumar el número invertido con el original acaba produciendo en muchas ocasiones un capicúa. Ej: 195+591=786, 786+687=1473, 1473+3741=5217, 5217+7125=12342, 12342+24321=36663 que es capicúa producido en la 5ª iteración. El primer número que se sospecha que no acaba produciendo un capicúa es el 196 razón por la cual se nombra de esta guisa al algoritmo. Los primeros números que se sospecha que no producen capicúas al aplicarles el algoritmo 196 son: 196, 295, 394, 493, 592, 689, 691, 788, 790, 879, 887, … Puedes consultar la secuencia de Sloane A023108 para indagar cuáles se cree que son los 45 primeros. Ej: El capicúa que se alcanza comenzando con 89 es especialmente grande: 8813200023188.
	Elabora un programa que calcule el capicúa que se alcanza y en qué iteración aplicando el algoritmo 196 a los enteros comprendidos entre 10 y 200 excepto el 196.
	☞ Elabora una función (algo196) que invierta y sume el número invertido con el original ☞ Reutiliza el proyecto P17 Palíndromos para comprobar que alcanzas un capicúa y se detiene el proceso iterativo.

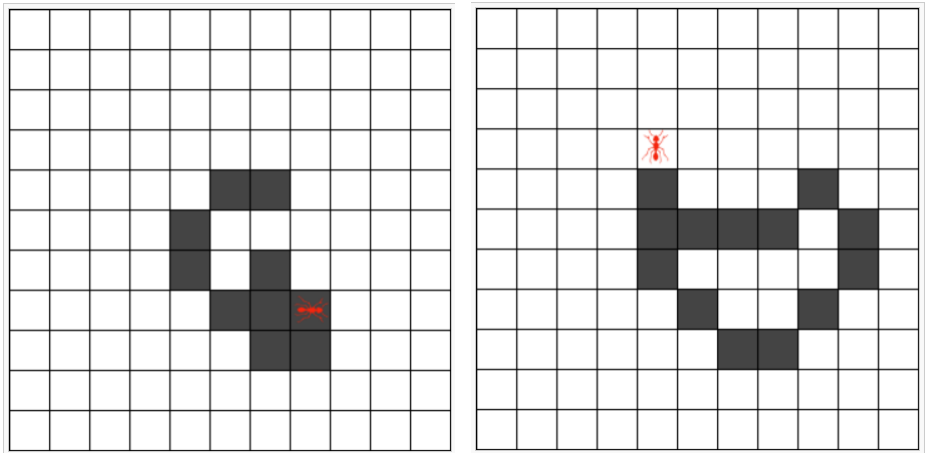
P94	La hormiga de Langton

Un autómata celular es una rejilla de celdas que cambian de estado de acuerdo con un conjunto de reglas. **La hormiga de Langton** es uno de estos autómatas. Una hormiga vive en una rejilla de cuadrados blancos y coloreados de acuerdo con las reglas siguientes:

1) Si está en una celda en blanco gira 90º a la derecha y avanza una celda.
2) Si está en una celda de color gira 90º a la izquierda y avanza una celda.
3) La celda a la que avanza invierte su color.

El resultado es un movimiento aparentemente caótico. Pero después de unos miles de movimientos entra en un bucle de 104 pasos que dibuja una avenida en diagonal. No importa en qué celda comience ni en qué dirección.

Conviene considerar la rejilla como un donut cortado y desplegado. Al alcanzar el borde superior/inferior de la rejilla cabe continuar por abajo/arriba. Al alcanzar el borde derecho/izquierdo cabe continuar por el izquierdo/derecho.



La hormiga de Langton en dos momentos diferentes: mirando al este sobre una casilla coloreada y al norte sobre una blanca.

OBJETIVO

Elabora un programa que reproduzca el autómata la hormiga de Langton comenzado en un píxel y una dirección (N, S, E, O) aleatorios y considerando una rejilla rectangular centrada del tamaño de la pantalla de BYOB (480x360 pixeles).

Te propongo el siguiente pseudocódigo que desmenuza el problema en problemas más pequeños.

Hace falta aderezarlo con un menú con cuatro botones de opción: comenzar, pausar, continuar y finalizar.

Utiliza las variables (x, y), para la posición en pantalla; d, para la dirección del movimiento y la matriz color(x, y) para el estado de cada celda.

```
algoritmo hormiga_de_Langton
generar_posición_inicial_aleatoria
repetir
   aplicar_movimiento (x, y)
hasta que 0=1 //no termina nunca
fin_algoritmo

procedimiento generar_posición_inicial_aleatoria
x←numero aleatorio (0,480)
y←numero aleatorio (0,360)
x←x-240
y←y-180
d←numero aleatorio (1,4)
d←(d-2)*90
ir_a (x, y) // instrucción BYOB ir a x: x y: y
apuntar_en_dirección d // instrucción BYOB apuntar en dirección d
fin_procedimiento

procedimiento aplicar_movimiento (x, y)
si color(x, y)=0 entonces
   girar_90º_derecha
   mover 1
si_no
   girar_90º_izquierda
   mover 1
fin_si
x←((posición_en_x+240) mod 480)-240 // instrucción BYOB posición en x
y←((posición_en_y+180) mod 360)-180 // instrucción BYOB posición en y
modificar_rejilla (x, y)
ir_a (x, y)
fin_procedimiento
procedimiento modificar_rejilla (x, y)
si color(x, y)=0 entonces
   color(x, y)←1
si_no
   color(x, y)←0
fin_procedimiento
```

P95	Algoritmo de factorización Pollard-ρ

El **algoritmo Pollard-ρ** es un algoritmo especializado de factorización de números enteros inventado por John Pollard en 1975 y especialmente efectivo a la hora de factorizar números compuestos que tengan factores pequeños. El algoritmo emplea una función módulo *n* a modo de generador de una secuencia pseudoaleatoria.

Hace funcionar una de las secuencias el doble de rápido que la otra, es decir, por cada iteración de una de las copias de la secuencia, la otra hace dos iteraciones. Sea *x* el estado actual de una secuencia e *y* el estado actual de la otra. En cada paso se toma el MCD de $|x - y|$ y *n*. Si este MCD llega a ser *n*, entonces finaliza el algoritmo con el resultado de fracaso, ya que esto significa que $x = y$ y, como está basado en el algoritmo de la liebre y la tortuga, la secuencia ya ha completado su ciclo y seguir más allá sólo conseguiría repetir trabajo ya realizado.

OBJETIVO

Codifica el algoritmo de Pollard-ρ para factorizar un entero positivo n.

El algoritmo, según el libro de Cormen, es:

```
POLLARD-RHO(n)
1  i = 1
2  x_1 = RANDOM(0, n − 1)
3  y = x_1
4  k = 2
5  while TRUE
6      i = i + 1
7      x_i = (x_{i−1}^2 − 1) mod n
8      d = gcd(y − x_i, n)
9      if d ≠ 1 and d ≠ n
10         print d
11     if i == k
12         y = x_i
13         k = 2k
```

Notas. No es necesario el subíndice de la x. La instrucción print d ha encontrado un factor que añadiremos a una lista. Incluimos una condición para hacerlo acabar y que no trabaje de forma innecesaria.

El algoritmo quedaría:

```
1  i = 1
2  x = aleatorio (0, n-1)
3  y = x
4  k = 2
5  mientras cierto
6      i = i+1
7      x = (x²-1) mod n
8      d = mcd(y-x, n)
9      si d≠1 y d≠n entonces
10         añadir d a lista
       fin_si
       if d=n entonces falso
       fin_si
11     si i=k entonces
12         y = x
13         k = 2k
       fin_si
   fin-mientras
```

Reutiliza el proyecto P4 Máximo común divisor.

P96	Casando cadenas de caracteres

Los programas de edición de textos necesitan con frecuencia localizar todas las ocurrencias de cierto patrón proporcionado por el usuario. Hacen falta algoritmos eficientes para este problema llamado "case de cadenas". Entre sus aplicaciones están el tratamiento de secuencias de aminoácidos del ADN o la localización de páginas web en Internet que son relevantes para el contenido que se desea localizar. Un ejemplo es:

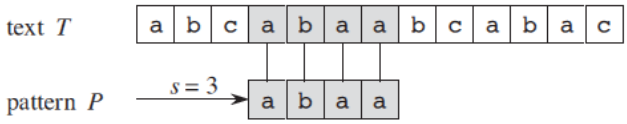


Figure 32.1 An example of the string-matching problem, where we want to find all occurrences of the pattern P = abaa in the text T = abcabaabcabac. The pattern occurs only once in the text, at shift $s = 3$, which we call a valid shift. A vertical line connects each character of the pattern to its matching character in the text, and all matched characters are shaded.

Veamos dos algoritmos para programarlos. El **algoritmo ingenuo** mediante un bucle localiza todas las coincidencias de los m caracteres del patrón P en el texto T:

```
NAIVE-STRING-MATCHER(T, P)
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4      if P[1..m] == T[s + 1..s + m]
5          print "Pattern occurs with shift" s
```

El algoritmo KMP (ideado por Knuth, Morris y Pratt) es una mejora de una autómata finito que trata letra a letra:

```
KMP-MATCHER(T, P)
1   n = T.length
2   m = P.length
3   π = COMPUTE-PREFIX-FUNCTION(P)
4   q = 0                                  // number of characters matched
5   for i = 1 to n                         // scan the text from left to right
6       while q > 0 and P[q + 1] ≠ T[i]
7           q = π[q]                       // next character does not match
8       if P[q + 1] == T[i]
9           q = q + 1                      // next character matches
10      if q == m                          // is all of P matched?
11          print "Pattern occurs with shift" i − m
12          q = π[q]                       // look for the next match
```

Y se apoya en la siguiente función (observa que devuelve una lista llamada π):

```
COMPUTE-PREFIX-FUNCTION(P)
 1  m = P.length
 2  let π[1..m] be a new array
 3  π[1] = 0
 4  k = 0
 5  for q = 2 to m
 6      while k > 0 and P[k + 1] ≠ P[q]
 7          k = π[k]
 8      if P[k + 1] == P[q]
 9          k = k + 1
10      π[q] = k
11  return π
```

OBJETIVO

Codifica el algoritmo ingenuo y el KPM para encontrar los cases de una cadena patrón dentro de una cadena de texto.

☞

P97	Estadísticos robustos: la línea de Tukey
	Uno de los problemas del enfoque clásico de la estadística descriptiva cuando analiza datos es la presencia de valores atípicos (*outliers*). Su presencia se detecta fácilmente realizando un gráfico de caja y bigotes. John Tukey propuso la línea Mediana-Mediana como un ajuste más robusto (resistente a valores atípicos) que la recta de regresión por mínimos cuadrados (véase P63).
	Codifica el algoritmo para calcular la línea de Tukey de un conjunto de n puntos.
	El procedimiento de construcción de la línea de Tukey es el siguiente: Dado un conjunto de n puntos $S=\{(x_i, y_i): 1<=i<=n\}$ 1) Ordenar los puntos de S por abscisas crecientes 2) Dividir el conjunto ordenado en tres partes G_1, G_2 y G_3 que serán del mismo cardinal si n es múltiplo de 3. En caso de que no lo sea, situar el punto sobrante en el grupo central o los dos puntos sobrantes en los grupos extremos. 3) Calcular las medianas de cada coordenada en cada grupo. Sean (x_{m1}, y_{m1}), (x_{m2}, y_{m2}) y (x_{m3}, y_{m3}) dichas medianas. 4) Calcular la pendiente de la línea de Tukey: es la pendiente de la recta que pasa por (x_{m1}, y_{m1}) y (x_{m3}, y_{m3}). 5) Calcular la ordenada en el origen de la línea de Tukey: es la media de y_{m1}, y_{m2} e y_{m3}.

P98	Estadísticos robustos: estimador M de Huber
	La mediana es una medida de centralización robusta dado que está poco afectada por valores atípicos. En su cálculo solo influyen los valores centrales. Huber (1964) propuso un estimador central más robusto que la mediana. Para ello generalizó las llamadas **medias recortadas y winsorizadas**: ordenados los datos o bien se eliminan una porción de los datos extremos (media recortada) o bien se igualan los valores extremos a valores más cercanos a la mediana (winsorización).
	Codifica el algoritmo para calcular el estimador M de Huber de un conjunto de n datos.
	El procedimiento de cálculo más sencillo del estimador M de Huber es el siguiente: Dados n datos $x_1, x_2, x_3, \ldots, x_n$ 1) Calcular su media m_0 y su desviación típica s_0. 2) Construir el primer intervalo de aceptación $[m_0 - 1.5s_0, m_0 + 1.5s_0]$ 3) Valores menores que $m_0-1.5s_0$ se igualan a $m_0-1.5s_0$; valores mayores que $m_0+1.5s_0$ se igualan a $m_0+1.5s_0$ 4) Calcular la nueva media m_1 y desviación típica s_1. 5) Construir el siguiente intervalo de aceptación $[m_1 - 1.134s_1, m_1 + 1.134s_1]$ 6) Valores menores que $m_1-1.134s_1$ se igualan a $m_1-1.134s_1$; valores mayores que $m_1+1.134s_1$ se igualan a $m_1+1.134s_1$ 7) Repetir 4, 5 y 6 hasta que la diferencia de medias $\lvert m_{i-1} - m_i \rvert$ sea menor que el orden de precisión, p.e., 0.0001 El último m_i calculado es el estimador M de Huber.