Algoritmos y diagramas de flujo

Informática I - 2547100

Departamento de Ingeniería Electrónica
y de Telecomunicaciones

Facultad de Ingeniería

2016-2

Solving problems

A partir del planteamiento de un problema, es conveniente usar una **metodología** para la resolución del problema, que tendrá como objetivo final el algoritmo que dará una solución.

Polya's method

George Polya, Matemático húngaro, autor del libro *How to solve it* (1945). En él, Polya propone cuatro pasos generales para la solución de un problema:

1. **Entienda el problema**: parece obvio pero es con frecuencia un gran obstáculo.
   - ¿Entiende todas las palabras de la formulación del problema?
   - ¿Qué le están pidiendo que encuentre?
   - ¿Puede usted reformular el problema en sus propias palabras?
   - ¿Puede hacer un dibujo que represente el problema?
   - ¿Hay suficiente información para encontrar la solución?

Polya's method

2. **Diseñe un plan**: escoja una estrategia para resolver el problema (divida el problema en problemas más simples, elimine posibilidades, aproveche simetrías, suponga y verifique, etc).
3. **Implemente el plan**: más fácil que el paso 2, sólo requiere mucho cuidado en los detalles y paciencia.
4. **Revise**: haga una pausa, revise y reflexione sobre el trabajo hecho.

Algorithms

Recordemos que, un algoritmo es un proceso preciso, computable y finito, que paso a paso lleva a la solución de un problema.

Todo algoritmo debe tener tres partes:

Examples of algorithms

¿Cómo ingresar a la Universidad de Antioquia?

1. Comprar formulario de inscripción
2. Elegir carrera
3. Presentar examen
4. Si no pasa, volver al paso 1
5. Pagar matrícula
6. Elegir materias

¿Cómo dibujar una parábola en el plano cartesiano (-10,10)?

1. Asignar a x el valor *-10*
2. Asignar a *y* el valor de x^2
3. Dibujar un punto en la coordenada *x,y*
4. Sumar *1* a x
5. Si x es menor o igual a 10, vaya al paso 2

Data in an algorithm

Variable: espacio de memoria asociado con un identificador (nombre) que almacena un valor (un dato) que puede ser modificado por instrucciones del algoritmo.

- Variables de entrada y salida
- Variables auxiliaries
- Constante: un dato que no cambia

Una variable puede representar un número decimal, un numero entero, un arreglo de números o de caracteres, etc.

Algorithm design

Calcular el área de un triángulo

Análisis:

- ¿Cuál es el objetivo buscado?

  Calcular el área de un triángulo

- ¿Cuáles son los datos de entrada?

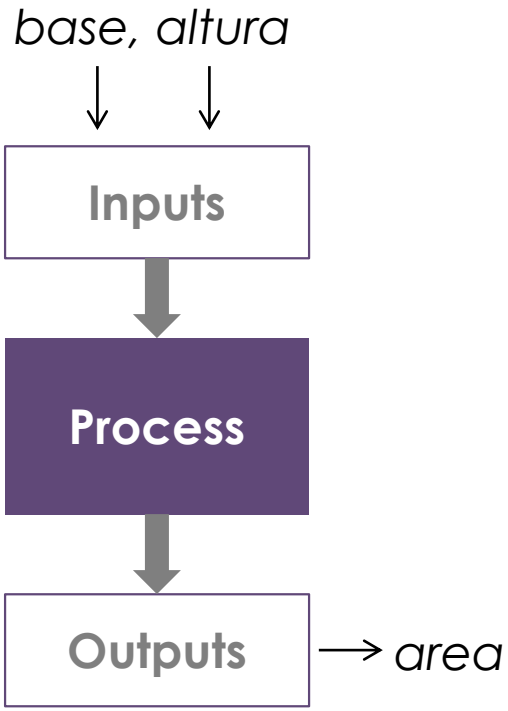
  Base y altura

- ¿Cuáles son los datos de salida?

  Área de un triángulo

- ¿Qué cálculos/procesos deben llevarse a cabo?

  area = (base * altura)/2



Algorithm example

Se requiere diseñar un algoritmo que **calcule el número de meses que hay entre los años A y B**.

Datos de entrada: los años especificados (A y B)
Datos de salida: numero total de meses transcurridos
Definición de variables:

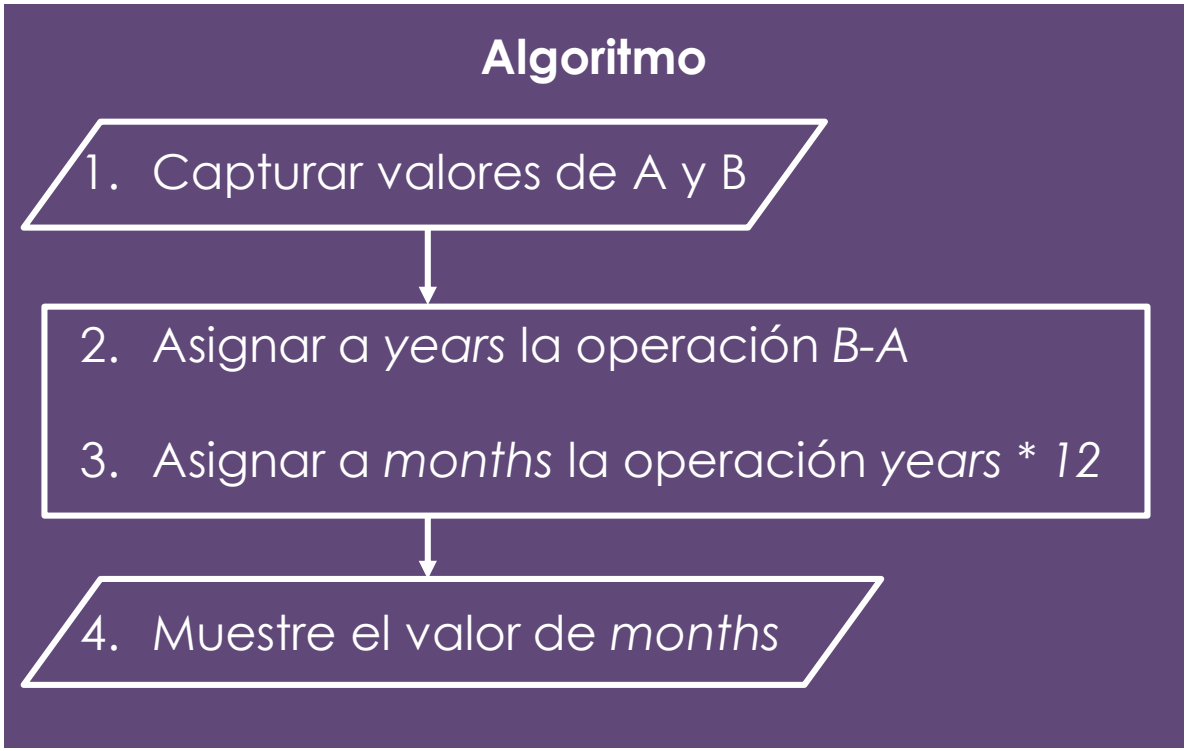
- *A: primer año*
- *B: segundo año*
- *years*: años transcurridos
- *months*: meses transcurridos

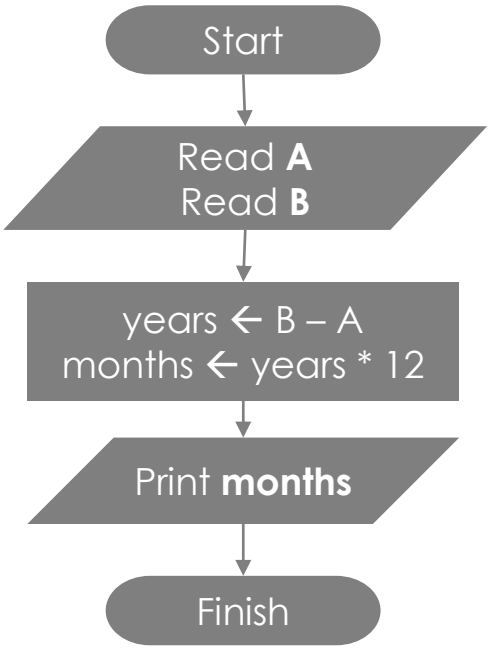
Algoritmo

1. Capturar valores de A y B
2. Asignar a *years* la operación *B-A*
3. Asignar a *months* la operación *years * 12*
4. Muestre el valor de *months*

Algorithms represented as flowcharts

Se requiere diseñar un algoritmo que **calcule el número de meses que hay entre los años A y B**.





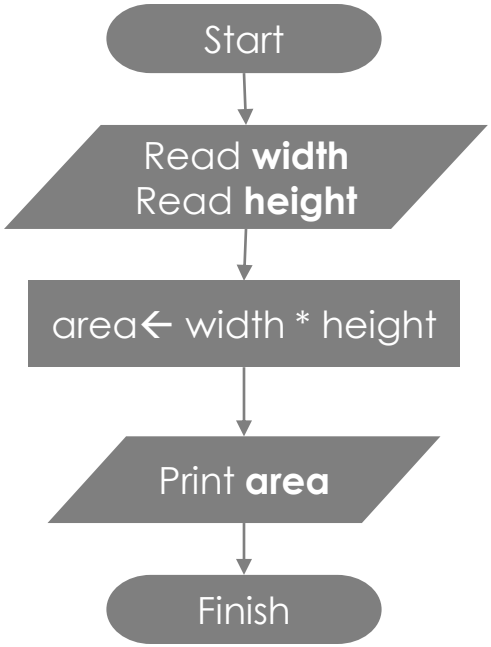
Algorithms represented as flowcharts

Escriba un algoritmo que calcule el área de un rectángulo dada la longitud de sus lados.

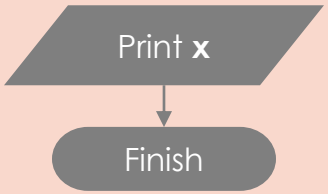
Datos de entrada: longitud de los lados
Datos de salida: área del rectángulo
Definición de variables:

- *width*: ancho
- *height*: alto
- *area*: área

¿Proceso?: multiplicar el ancho por el alto



Syntax vs. semantics

Sintáxis	Semántica
Conjunto de **reglas** que determinan los símbolos y las combinaciones de éstos, que son válidos en un lenguaje.	Es el **significado** de las expresiones permitidas por la sintáxis de los algoritmos y programas.
	*Determina si un número es primo*

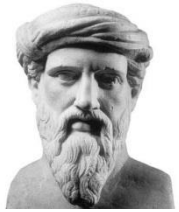
Distance between points

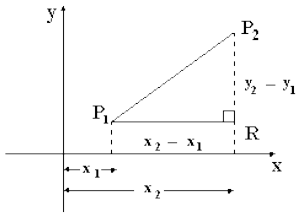
Escriba un algoritmo que calcule la distancia entre dos puntos en el plano cartesiano, dadas su coordenadas.

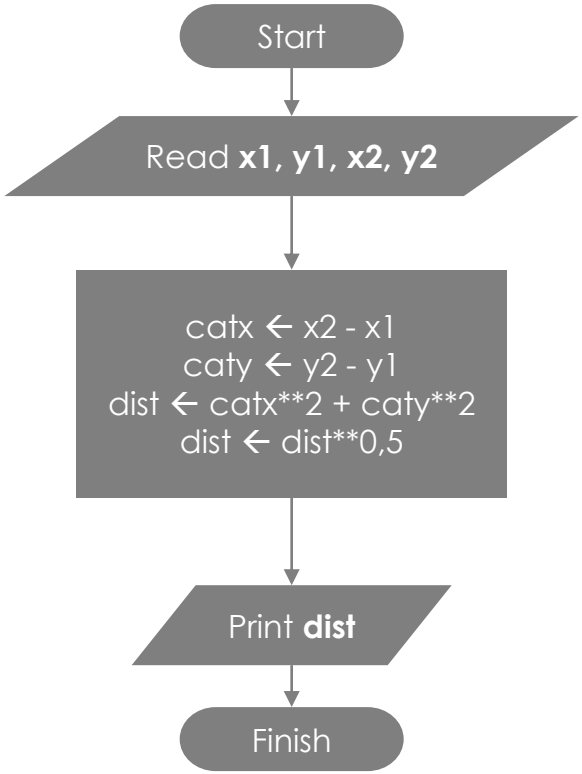
Datos de entrada: coordenadas x1, y1 y x2, y2
Datos de salida: distancia
Definición de variables:

- *x1, x2, y1, y2*: coordenadas
- *dist*: distancia
- *catx, caty*: catetos

¿Proceso?:







Primitives

- Son las operaciones permitidas en un algoritmo, para las cuales no es necesario hacer su propio algoritmo.
- En lenguajes de programación, son las instrucciones que el traductor entiende, que es capaz de traducir.

Operators and expressions

Operadores aritméticos			
Operador	**Nombre**	**Ejemplo**	**Resultado**
+	Suma	1 + 3	4
-	Resta	7 – 10	-3
*	Multiplicación	3*5	15
/	División	8/5	1.6
//	División entera	8//3	2
%	Modulo (Residuo)	15%7	1
**	Potencia	2**3	8

$$\frac{4x^2 - 2x + 8}{c - d}$$

```
(4*(x**2)-2*x+8)/(c-d)
```

Operadores relacionales			
Operador	**Nombre**	**Ejemplo**	**Resultado**
>	Mayor	1 > 3	False (F)
>=	Mayor o igual	2 >= 1	True (V)
<	Menor	-5 < -1	True (V)
<=	Menor o igual	3 <= 3	True (V)
!=	Diferente	13 != 4	True (V)
==	Igual (Comparación)	0 == 1	False (F)

$$(x_2 - x_1)^2 \geq (y_2 - y_1)^2$$

```
(x2-x1)**2 >= (x2-x1)**2
```

Assignment operator

Permite asociar el resultado de una expresión, a una variable. Una expresión por sí sola:

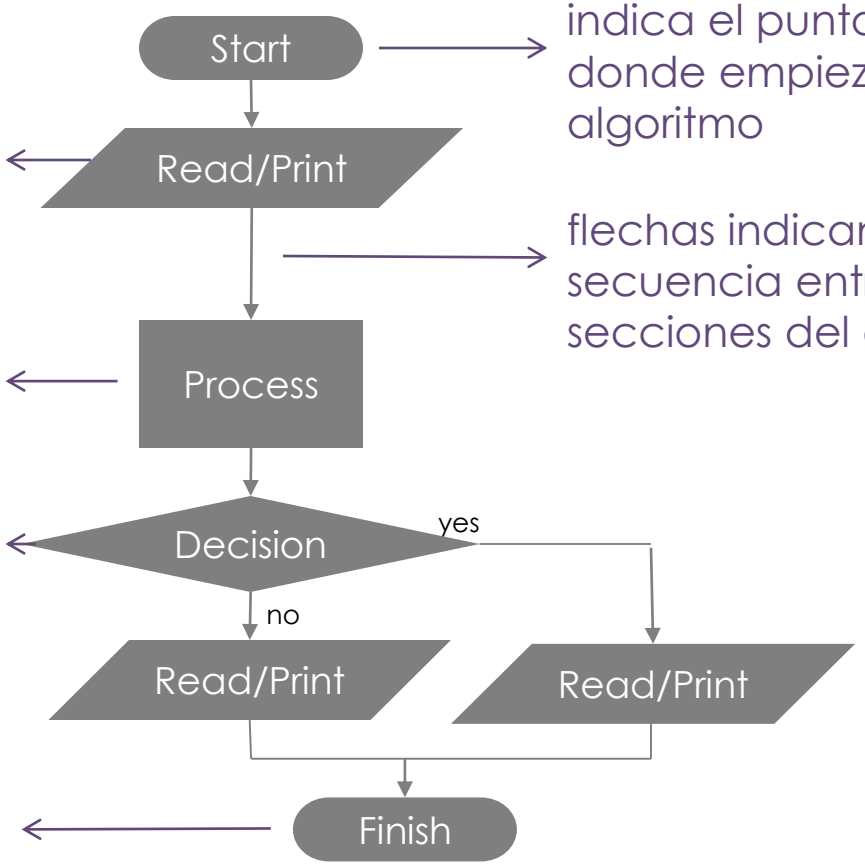
```
((x**3)-5*x-2)/(a%d)*k
```

¡no indica qué hacer con el resultado!

```
z ← ((x**3)-5*x-2)/(a%d)*k
```

¡El operador de asignación es el **único** que cambia el *estado* de una variable!

Flowcharts



indica el punto único donde empieza el algoritmo

indica la lectura (recepción) o impresión (envío o salida) de datos desde o hacia afuera del sistema

flechas indican la secuencia entre dos secciones del algoritmo

indica un proceso secuencial de operaciones o cálculos

indica la evaluación de una condición para determinar el flujo del algoritmo

indica el punto donde termina el algoritmo

Programa para probar diagramas de flujo: http://raptor.martincarlisle.com/

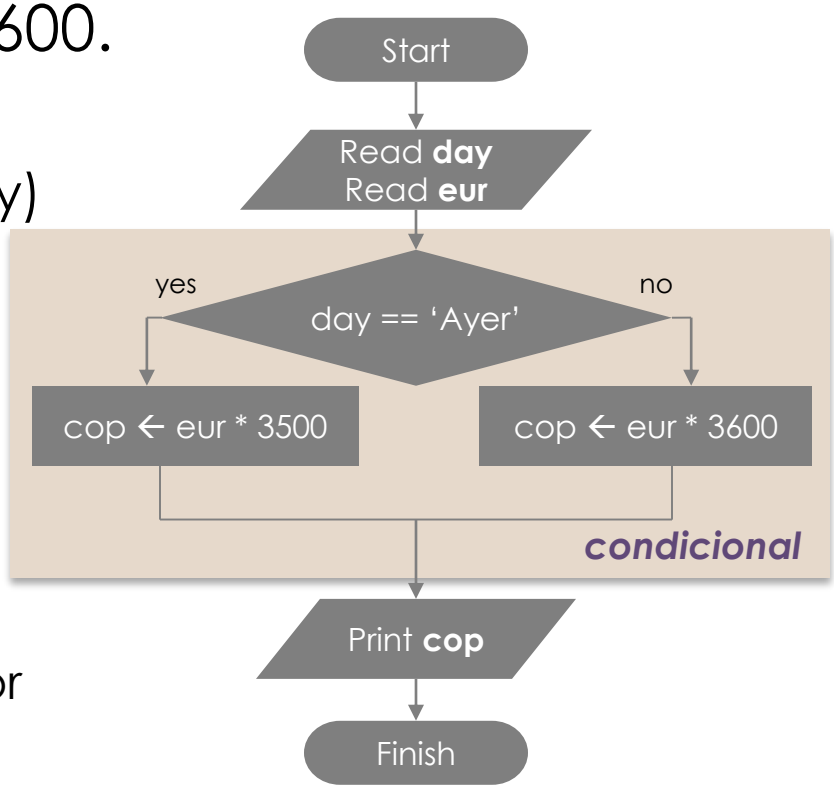
Conditional euro-peso conversion

Escriba un algoritmo que convierta una cantidad en euros a su equivalente en pesos, de ayer o de hoy, según se especifique. Asuma que el precio del euro ayer fue 3500 y el de hoy es 3600.

Datos de entrada: día (ayer u hoy) y cantidad de euros a convertir
Datos de salida: valor en pesos
Definición de variables:

- *day*: día
- *eur*: cantidad en euros
- *cop*: cantidad en pesos

¿Proceso?: dependiendo del día elegido, multiplicar eur por 3500 o por 3600



Conditionals

Sirven para representar alternativas de ejecución, es decir, que se haga una cosa u otra dependiendo del valor de una expresión lógica.

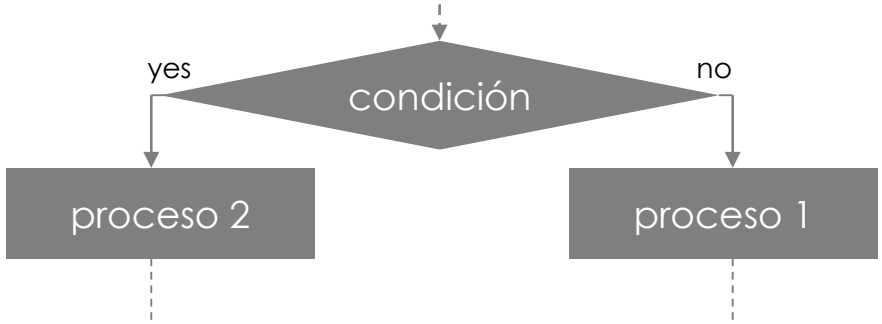
SI (expresión lógica)

Instrucciones que se realizan si la expresión es verdadera

DE LO CONTRARIO

Instrucciones que se realizan si la expresión es falsa

FIN SI



Leap year algorithm

Se requiere un algoritmo que al ser preguntado por un año cualquiera, diga si éste es un año bisiesto o no.

Atención: no todos los años múltiplos de 4 son bisiestos. Si es múltiplo de 100 no es bisiesto, a no ser que sea también múltiplo de 400.

Use el operador módulo (%).

Si X = A * B + R con 0 < R < A
entonces,
R = X % A

Datos de entrada: número entero *year*
Datos de salida: mensaje
Definición de variables:

- *year*: número entero (año)

¿Proceso?: verificar si el año ingresado es divisible por 400, por 100 y por 4 para decidir si es bisiesto o no.

Diagrama de flujo para el algoritmo del año bisiesto

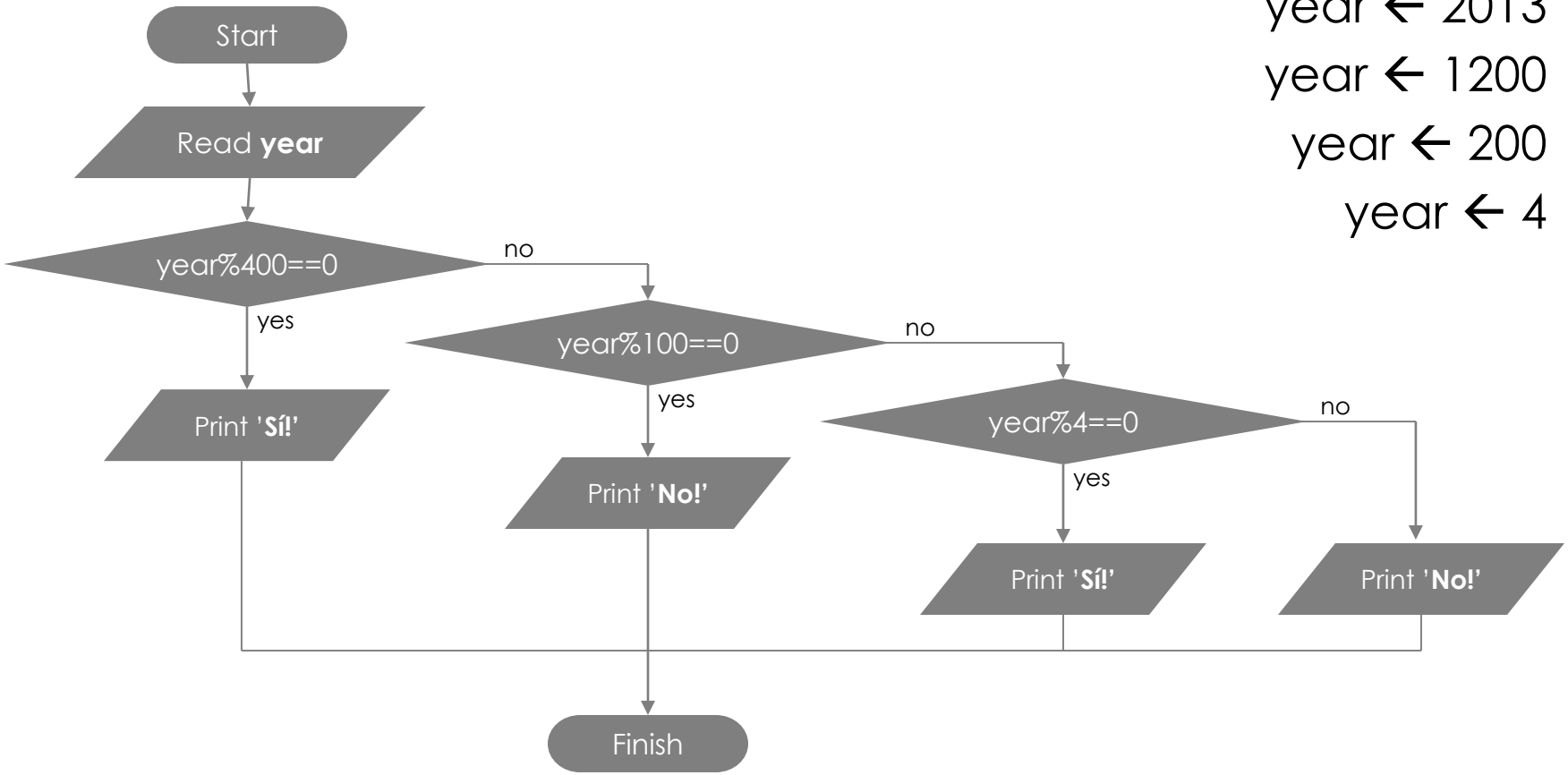
Hacer pruebas con:

year ← 2013

year ← 1200

year ← 200

year ← 4



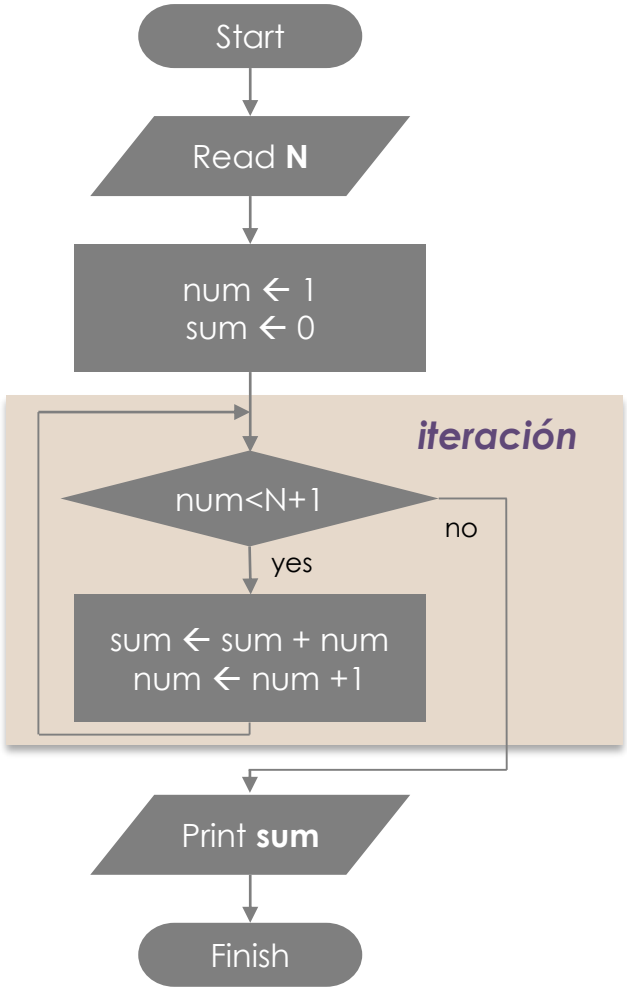
Sum of the first **N** integers

Escriba un algoritmo que calcule la suma de los primeros **N** números enteros, donde **N** es un dato de entrada.

Datos de entrada: número entero N
Datos de salida: suma de los primeros N números
Definición de variables:

- *N*: número entero
- *num*: número que se va incrementando (contador auxiliar)
- *sum*: suma

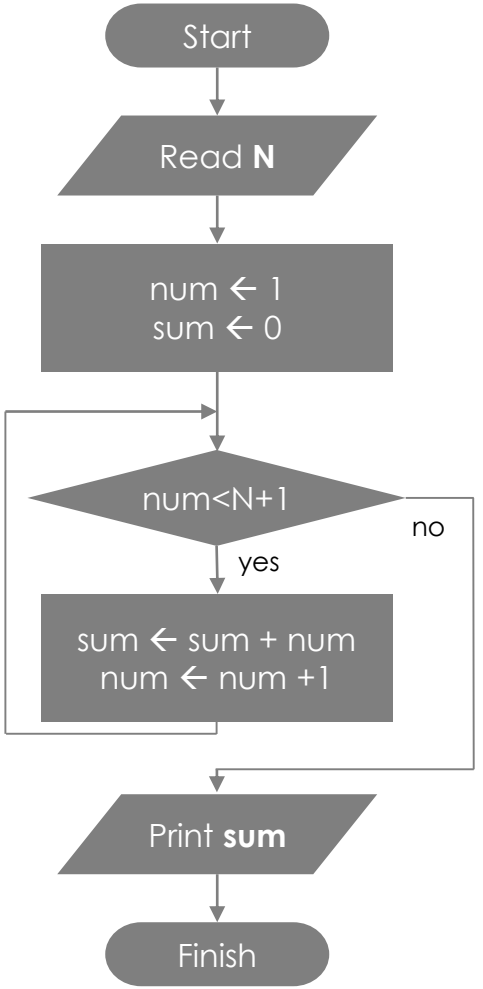
¿Proceso?: generar los números consecutivos desde el 1 hasta el *N* e irlos sumando



Prueba de escritorio

Es una prueba manual que se hace de un algoritmo para verificar su funcionamiento.

N	num	sum
4	1	0
4	2	1
4	3	3
4	4	6
4	5	10



Iteration

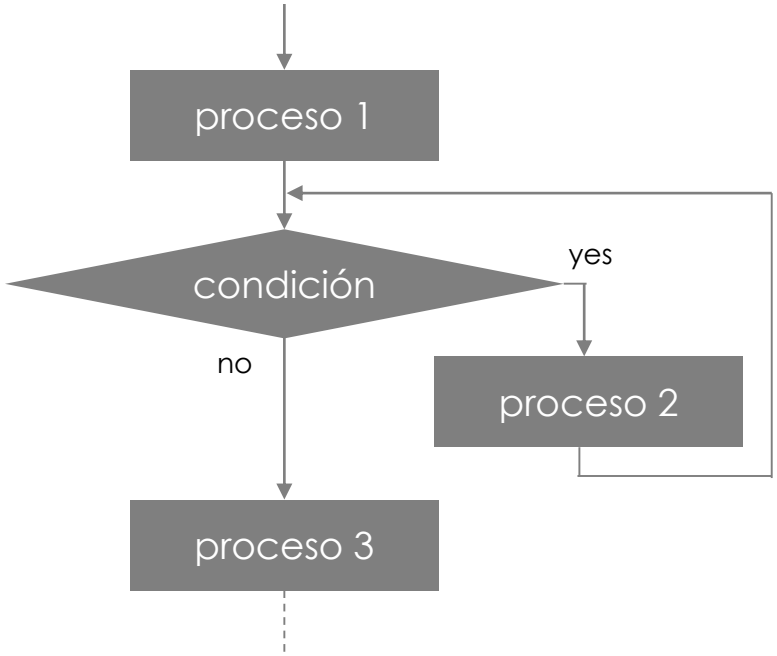
Las iteraciones, repeticiones, bucles o ciclos sirven para indicar la repetición de una o varias instrucciones mientras que se cumpla una condición.

MIENTRAS (expresión lógica)

Instrucciones que se realizan mientras que la expresión sea verdadera

FIN MIENTRAS

Instrucciones que se realizan luego de que la expresión se vuelva falsa



Counter and accumulator variables

Variable **contador**: sirve para contar cuantas veces se dá una condición o evento

-> cuántas veces se repite un ciclo, cuantas veces una variable supera un límite, etc

```
count ← count + 1
```

Variable **acumulador**: sirve para acumular los resultados de una operación que se repite

-> acumular la suma iterativa de varios elementos, acumular las multiplicaciones para calcular un factorial, etc

```
sum ← sum + res
sum ← sum * res
```

Block diagrams

Primer paso para identificar entradas, salidas y bloques funcionales (o procesos), entre otros.

Ej: Dada una lista de estudiantes y sus notas encriptadas, calcule la nota promedio del grupo y la lista de estudiantes que no pasaron. Se conoce además la contraseña y la nota mínima para pasar.