Introducción a la programación

Parte 2

Estructuras de control

- En los ejemplos vistos hasta ahora, las instrucciones se ejecutan en forma secuencial.
- Se empieza por la primera instrucción y se continúa en orden hasta la última.
- Todas las instrucciones son ejecutadas.
- Ninguna instrucción se ejecuta más de una vez.

Estructuras de control

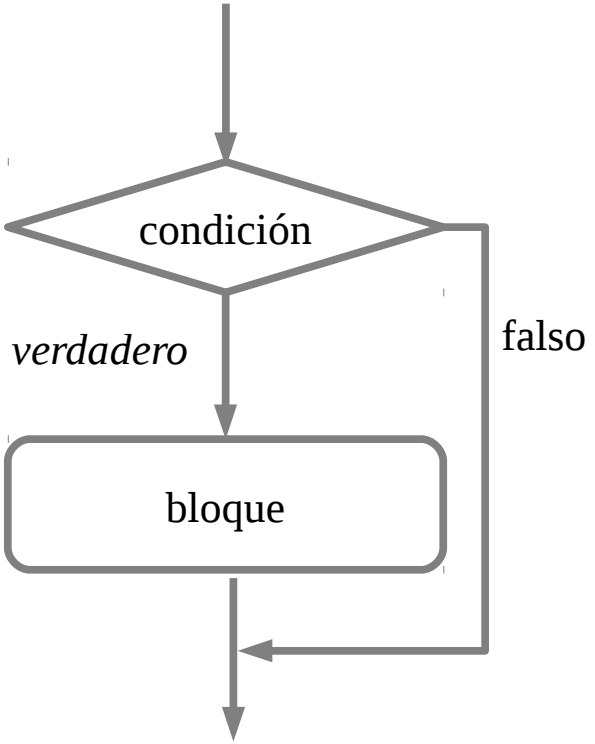
- Las estructuras de selección e iteración permiten modificar la ejecución del programa.
- **Selección**: se chequean condiciones para decidir qué instrucciones ejecutar. Esto implica que algunas instrucciones pueden no ejecutarse.
- **Iteración**: algunas instrucciones se ejecutan varias veces.

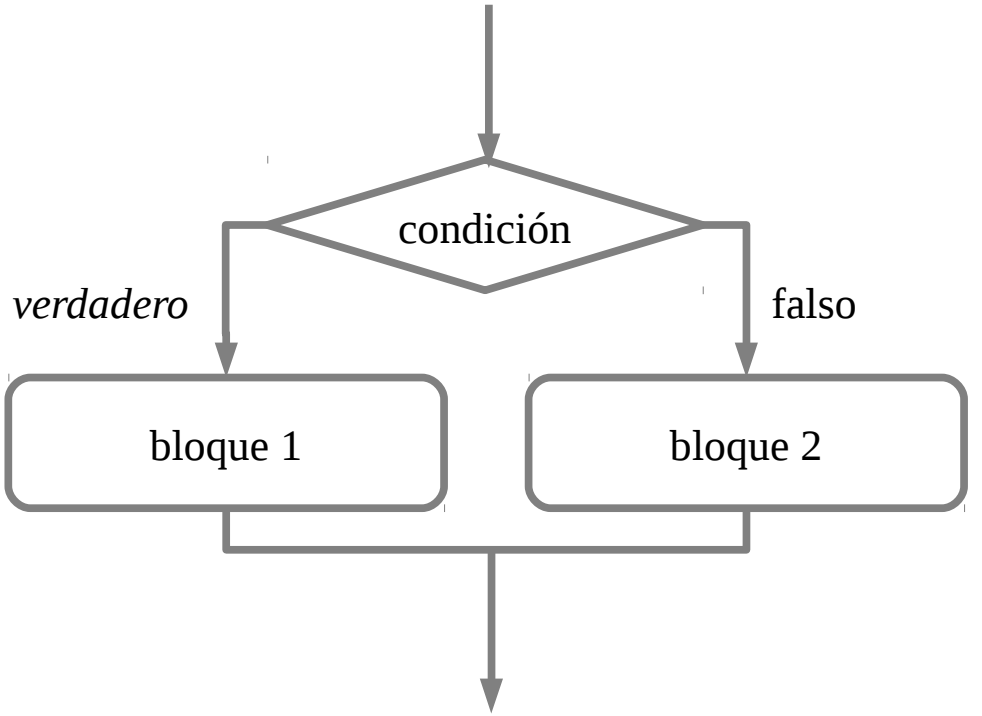
Estructuras de control: selección

Retomamos el algoritmo del lavarropas:

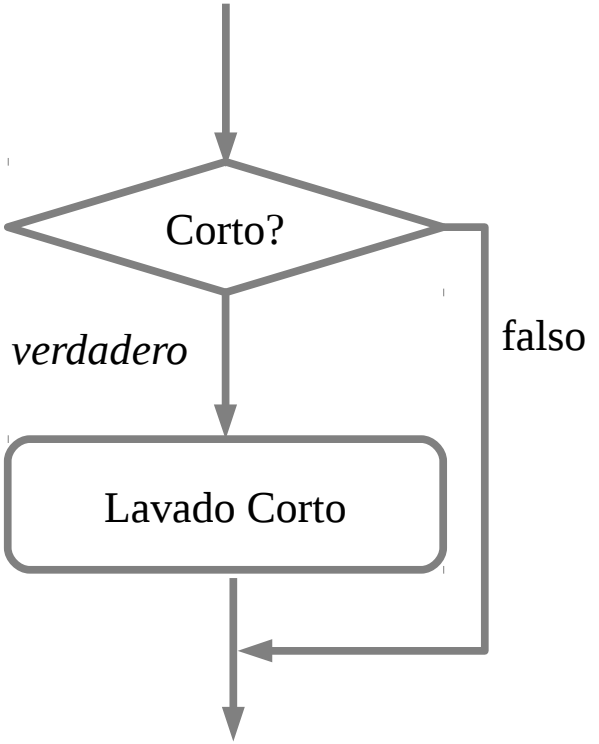
¿Qué tenemos que modificar para que el algoritmo reciba un dato de entrada que indique el tipo de lavado a realizar (corto, normal, largo) y se comporte de forma diferente según ese dato?

Estructuras de control: selección

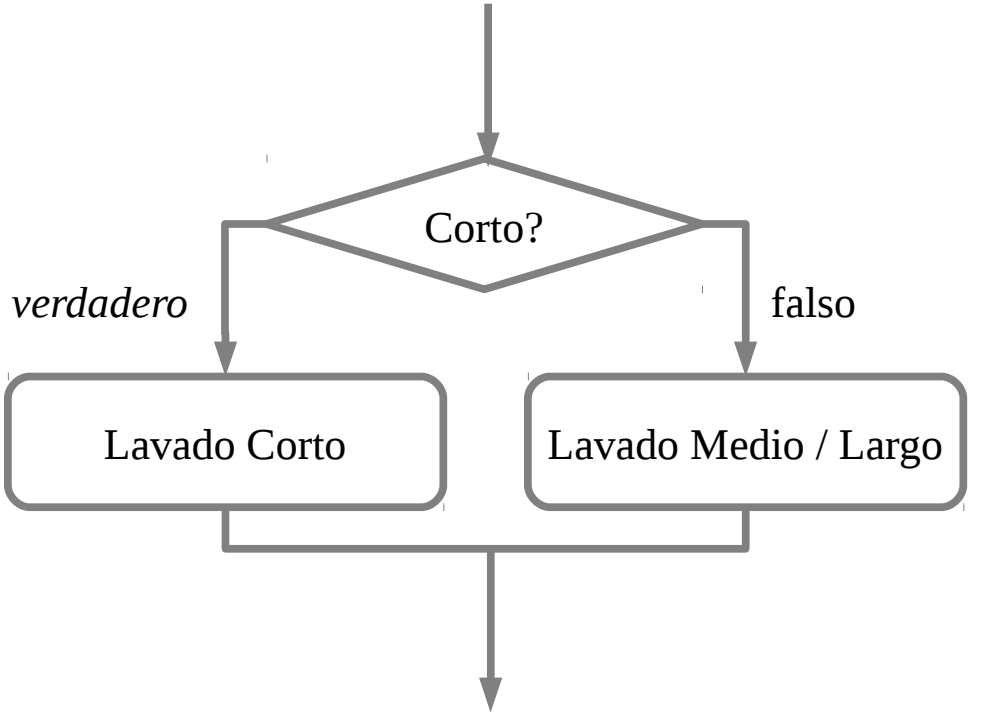




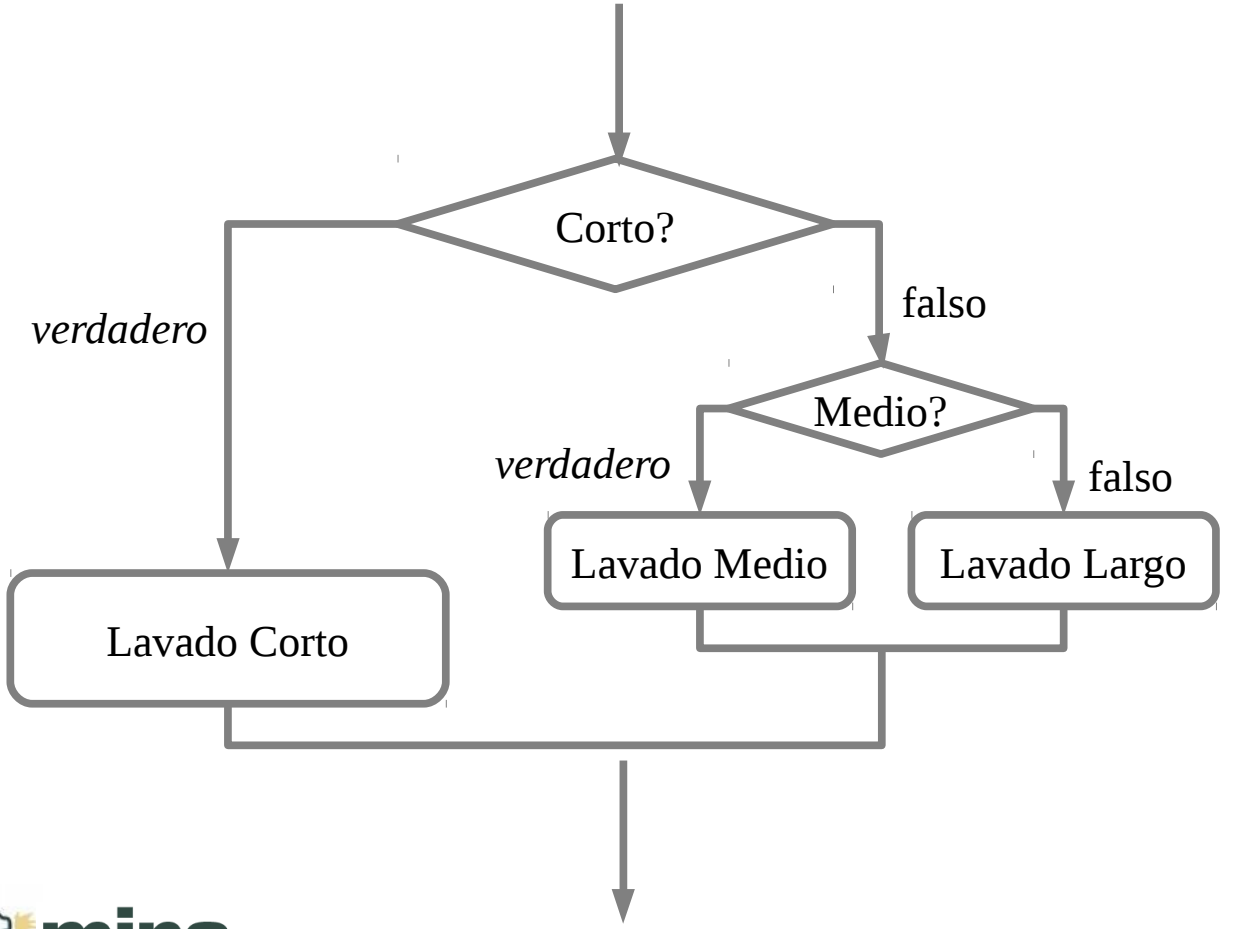
Estructuras de control: selección



Estructuras de control: selección



Estructuras de control: selección



Estructuras de control: selección

Instrucción **if**

```
if condicion:
   bloque
```

Instrucción **if else**

```
if condicion:
   bloque1
else:
   bloque2
```

Instrucción **if elif [else]**

```
if cond1:
   bloque1
elif cond2:
   bloque2
elif cond3:
   bloque3
else:
   bloque4
```

bloque: secuencia de instrucciones indentadas

```
if cond:
   instr1
   instr2
   instr3
else:
   instr4
```

Estructuras de control: selección

Las condiciones son expresiones booleanas, por lo que al ser evaluadas toman los valores True o False.

¿Qué se despliega en los siguientes casos, suponiendo que **a** vale 2 y **b** vale 5?

```
   if  a > b :
      print 'entro al if'
   else:
      print 'entro al else'
------------------------------
   if (a > 9) or (b == 5) :
      print 'es verdadero'
   else:
      print 'es falso'
```

Estructuras de control: selección

¿Qué se despliega en los siguientes casos, suponiendo que **a** vale 2 y **b** vale 5?

```
   if  (a > b) and (b > 0) :
      print 'entro al if'
   else:
      print 'entro al else'
--------------------------------
   if a == b :
      print 'son iguales'
--------------------------------
   if a == 0 :
      print 'a es 0'
   elif a > 0 :
      print 'mayor que 0'
   else:
      print 'otro caso'
```

Ejercicios (selección)

1) Escribir un programa que lea dos valores de la entrada y despliegue el mayor.
2) Escribir un programa que lea dos valores de la entrada, a y b, y despliegue uno de los siguientes mensajes, según corresponda: “a es mayor que b”, “a es igual a b” o “a es menor que b”.
3) Reescribir el programa que calcula cociente y resto chequeando si el divisor es distinto de 0.

Estructuras de control: iteración

Con lo visto hasta ahora, ¿podemos escribir un programa que implemente el algoritmo para sumar salarios?

Estructuras de control: iteración

Necesitamos repetir varias veces una misma operación:

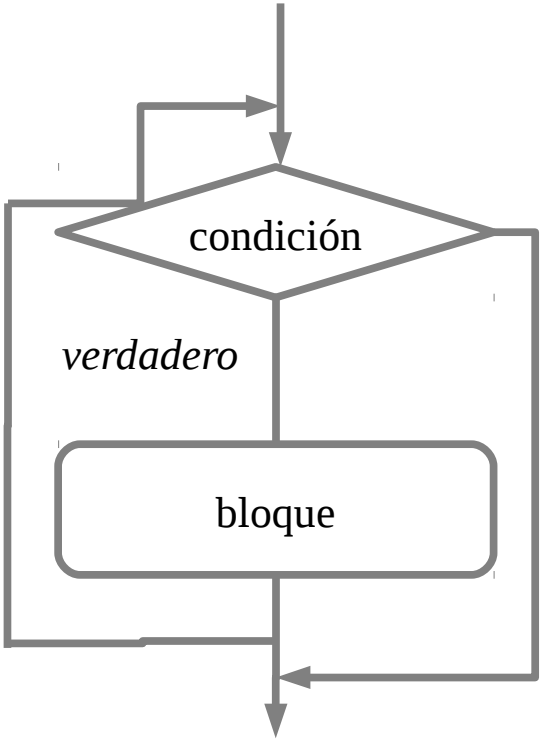
sumar el nuevo salario al valor que teníamos anotado

Tenemos dos estructuras para repetir o iterar:

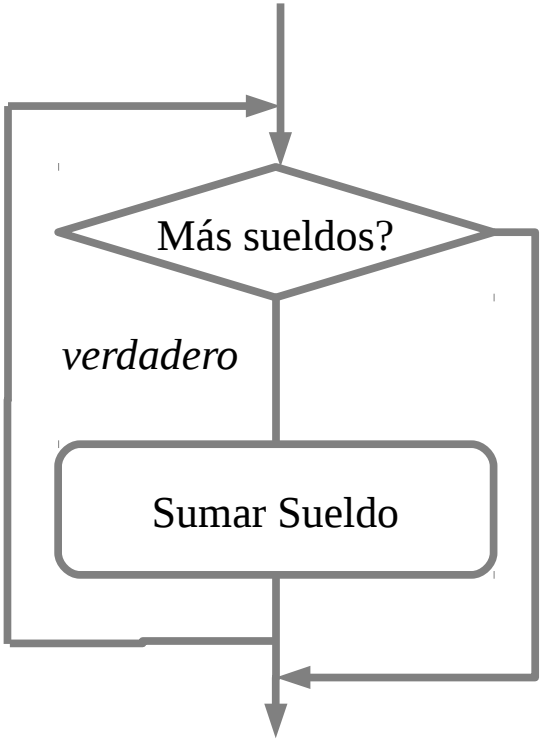
for -> iteración definida

while -> iteración condicional

Estructuras de control: iteración



Estructuras de control: iteración



Estructuras de control: iteración

En el ejemplo de los salarios,

- Si sabemos de antemano cuántos elementos tiene la lista, iteramos con **for** indicando cuántos pasos de iteración debemos dar.

Estructuras de control: iteración

Iteración definida

```
# itero cant veces
for i in range(0,cant):
    bloque
```

Se ejecuta el bloque *cant* veces.

En la primera ejecución la variable i toma el valor *0*, en las siguientes se va incrementando hasta tomar el valor *cant-1*.

Estructuras de control: iteración

```
# leo 10 valores de la entrada y los sumo

print 'ingrese 10 valores'
suma = 0
for i in range(0,10):
    valor = input()
    suma = suma + valor
print 'suma=', suma
```

Estructuras de control: iteración

En el ejemplo de los salarios,

- Si no sabemos cuántos elementos tiene la lista, iteramos con **while** poniendo como condición de terminación de la iteración llegar al final de la lista o a un valor especial que indica el fin de los datos (centinela).

Estructuras de control: iteración

Iteración condicional

```
# itero mientras se cumpla la condición
while condicion:
    bloque
```

1)Se evalúa la condición.

2)Si es verdadera se ejecuta el bloque y se vuelve al paso 1.

3)Si es falsa se saltea el bloque y se continúa con las instrucciones que estén después de él.

Estructuras de control: iteración

```
# leo enteros positivos de la entrada y los sumo
# hasta leer el valor -1

print 'ingrese valores positivos y al final -1'
suma=0
valor=input()
while valor <> -1:
    suma = suma + valor
    valor = input()
print 'suma=', suma
```

Ejercicios (iteración)

1) Escribir un programa que lea una cierta cantidad de valores enteros de la entrada y despliegue su suma y su promedio. La cantidad de valores a leer será ingresada por el usuario al principio.

2) Escribir un programa que lea una secuencia de valores positivos de la entrada hasta leer el valor -1 y luego despliegue el máximo valor leído.

Ejercicios (iteración)

1) Escribir un programa que cuente la cantidad de divisores que tiene un entero ingresado por el usuario.
2) Escribir un programa que diga si un entero ingresado por el usuario es primo.

Subprogramas: Funciones

- Un subprograma (en Python: función) es un fragmento de código que se comporta de manera independiente dentro de un programa.
- Las funciones pueden ser invocadas varias veces desde otras partes del programa(incluso desde ellas mismas).
- Las funciones son una herramienta de *modularización*.

Subprogramas: Funciones

- La modularización es importante porque permite dividir un problema en subproblemas (divide y vencerás).
- De este modo nos concentramos en la resolución de cada subproblema.
- Permite repartir el trabajo entre diferentes personas.
- Además, facilita la reutilización de código.

Subprogramas: Funciones

- Las funciones reciben parámetros y retornan resultados.
- Se definen utilizando la palabra: **def**.
- Los resultados se devuelven utilizando la instrucción **return**.

Subprogramas: Funciones

```
# calcula la cantidad de divisores de un entero
def cant_divisores(entero):
    cant = 2     # todos los enteros tienen
                 # al menos 2 divisores
                 # el 1 y el propio número
    for divisor in range(2,entero):
        if entero % divisor == 0:
            cant = cant+1
    return cant
```

Nota: el range optimizado debería ir hasta entero/2+1.

Subprogramas: Funciones

Las funciones se invocan desde el código del programa o desde otras funciones

```
# calcula la cantidad de divisores de un entero
def cant_divisores(entero):
    cant = 2
    for divisor in range(2,entero):
        if entero % divisor == 0:
            cant = cant+1
    return cant

# código principal
num = input(Ingrese un valor entero:)
cant_div = cant_divisores(num)
print 'Resultado: ', cant_div
```

Ejercicios (funciones)

1) Escribir la función **convertir**, que recibe tres dígitos y retorna el entero formado por esos tres dígitos.

Luego escriba un programa que pida al usuario que ingrese tres dígitos y despliegue el entero correspondiente, usando la función definida.

Ejercicios (funciones)

2) Escribir un programa que despliegue los primeros 20 números primos. Para esto, defina la función es_primo que recibe un número y retorna un valor booleano indicando si ese número es primo o no.

Referencias

Algoritmia

D. Harel, *Algorithmics: The Spirit of Computing,* Addison-Wesley, Reading, MA, 1st edition, 1987; 2nd edition, 1992. 3rd edition (with Y. Feldman), 2004.

Introducción a la programación

Apuntes teóricos y ejercicios del curso *Programación 1* de fing. (http://www.fing.edu.uy/inco/cursos/prog1)

Python

Tutorial: http://docs.python.org/tutorial/

Otros: http://en.wikibooks.org/wiki/Python_Programming

Español: http://pyspanishdoc.sourceforge.net/tut/tut.html