Estructuras de control
Fundamentos de la programación

Salvador Sánchez, Miguel-Angel Sicilia

Universidad de Alcalá

Septiembre de 2015

Licencia

El flujo de control

- Flujo de control: secuencia de ejecución de las instrucciones del programa.

El flujo de control

- Flujo de control: secuencia de ejecución de las instrucciones del programa.
- En un programa sin estructuras de control el flujo es **secuencial**.

El flujo de control

- Flujo de control: secuencia de ejecución de las instrucciones del programa.
- En un programa sin estructuras de control el flujo es **secuencial**.
- El flujo de control incluye la **activación** de módulos, es decir, la ejecución del código dentro de los mismos cuando se los invoca.

El flujo de control

- Flujo de control: secuencia de ejecución de las instrucciones del programa.
- En un programa sin estructuras de control el flujo es **secuencial**.
- El flujo de control incluye la **activación** de módulos, es decir, la ejecución del código dentro de los mismos cuando se los invoca.
- Para modificar el flujo de control se utilizan las estructuras de control de flujo: **selectivas** y **repetitivas**.

El flujo de control

- Flujo de control: secuencia de ejecución de las instrucciones del programa.
- En un programa sin estructuras de control el flujo es **secuencial**.
- El flujo de control incluye la **activación** de módulos, es decir, la ejecución del código dentro de los mismos cuando se los invoca.
- Para modificar el flujo de control se utilizan las estructuras de control de flujo: **selectivas** y **repetitivas**.

Estructuras de control selectivas

Estructuras selectivas

Definición

Una **estructura selectiva** permite, de acuerdo a una condición, ejecutar o no ciertas instrucciones.

- Cuando se requiere actuar de modo diferente ante diferentes entradas o en diferentes estados.

Estructuras selectivas

Definición

Una **estructura selectiva** permite, de acuerdo a una condición, ejecutar o no ciertas instrucciones.

- Cuando se requiere actuar de modo diferente ante diferentes entradas o en diferentes estados.
- Una condición lógica (verdadero/falso) determina qué instrucciones se deben ejecutar.

Estructuras selectivas

Definición

Una **estructura selectiva** permite, de acuerdo a una condición, ejecutar o no ciertas instrucciones.

- Cuando se requiere actuar de modo diferente ante diferentes entradas o en diferentes estados.
- Una condición lógica (verdadero/falso) determina qué instrucciones se deben ejecutar.
  - Operadores condicionales en python: ==, >, <, <=, >=, !=
  - Operadores lógicos: `and`, `or`, `not`

Estructuras selectivas

Definición

Una **estructura selectiva** permite, de acuerdo a una condición, ejecutar o no ciertas instrucciones.

- Cuando se requiere actuar de modo diferente ante diferentes entradas o en diferentes estados.
- Una condición lógica (verdadero/falso) determina qué instrucciones se deben ejecutar.
  - Operadores condicionales en python: ==, >, <, <=, >=, !=
  - Operadores lógicos: `and`, `or`, `not`
- Los dos posibles resultados de la evaluación de la condición (verdadero/falso) llevan a la ejecución de uno o de ningún bloque.

Ejemplo: Par o impar

```
n = int (input ("Introduce un entero: "))
if ( n %2 == 0 ):
    print("El numero es Par")
else:
    print("El numero es Impar")
```

Sangrado

- Cada vez que una sentencia acaba con dos puntos Python espera que la sentencia o sentencias que le siguen aparezcan con un mayor sangrado.

Sangrado

- Cada vez que una sentencia acaba con dos puntos Python espera que la sentencia o sentencias que le siguen aparezcan con un mayor sangrado.
- Es la forma de marcar el inicio y el fin de una serie de sentencias que *dependen* de otra.

Sangrado

- Cada vez que una sentencia acaba con dos puntos Python espera que la sentencia o sentencias que le siguen aparezcan con un mayor sangrado.
- Es la forma de marcar el inicio y el fin de una serie de sentencias que *dependen* de otra.
- Excepción: si solo hay una sentencia que depende de otra, pueden escribirse ambas en la misma línea.

Consejos sobre sangrado

- Líneas en blanco y comentarios no afectan al sangrado

Consejos sobre sangrado

- Líneas en blanco y comentarios no afectan al sangrado
- Eliminar los tabuladores en el editor

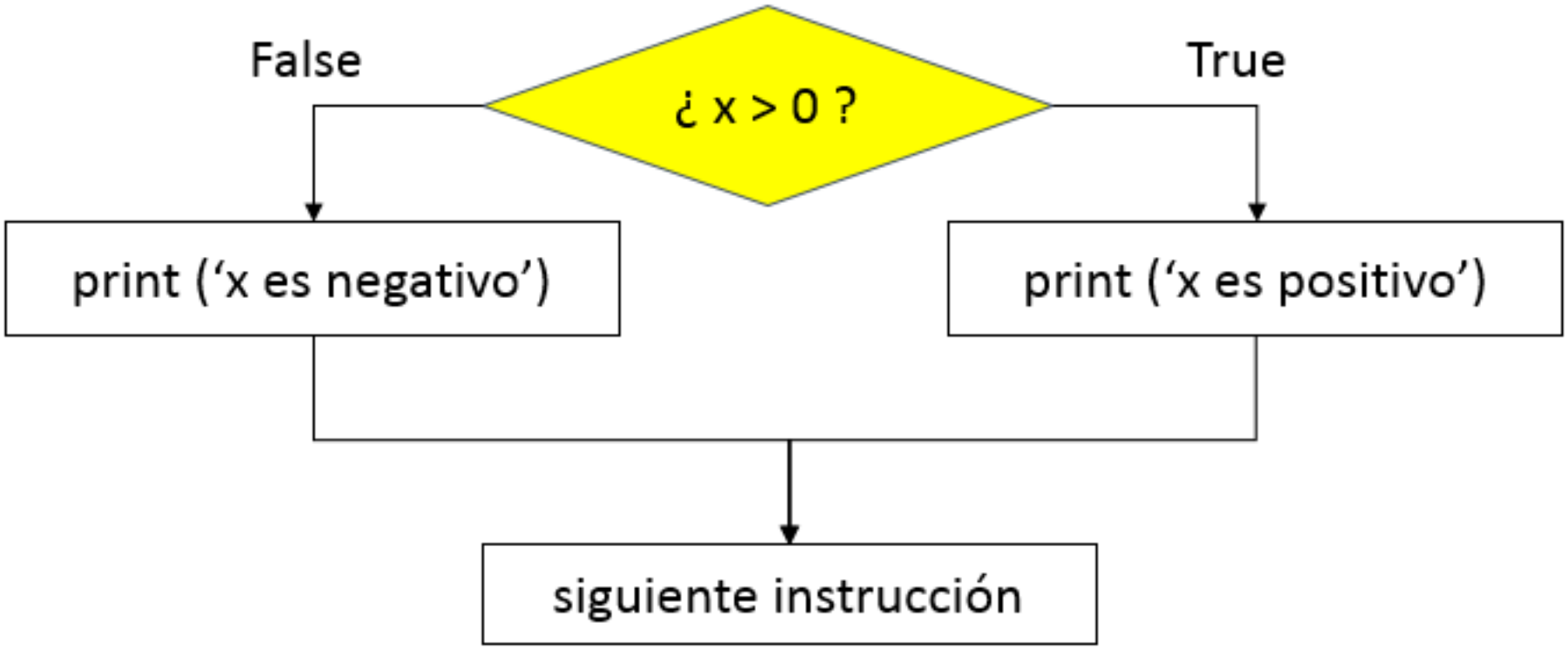
Consejos sobre sangrado

- Líneas en blanco y comentarios no afectan al sangrado
- Eliminar los tabuladores en el editor
- Usar "tablas mentales":

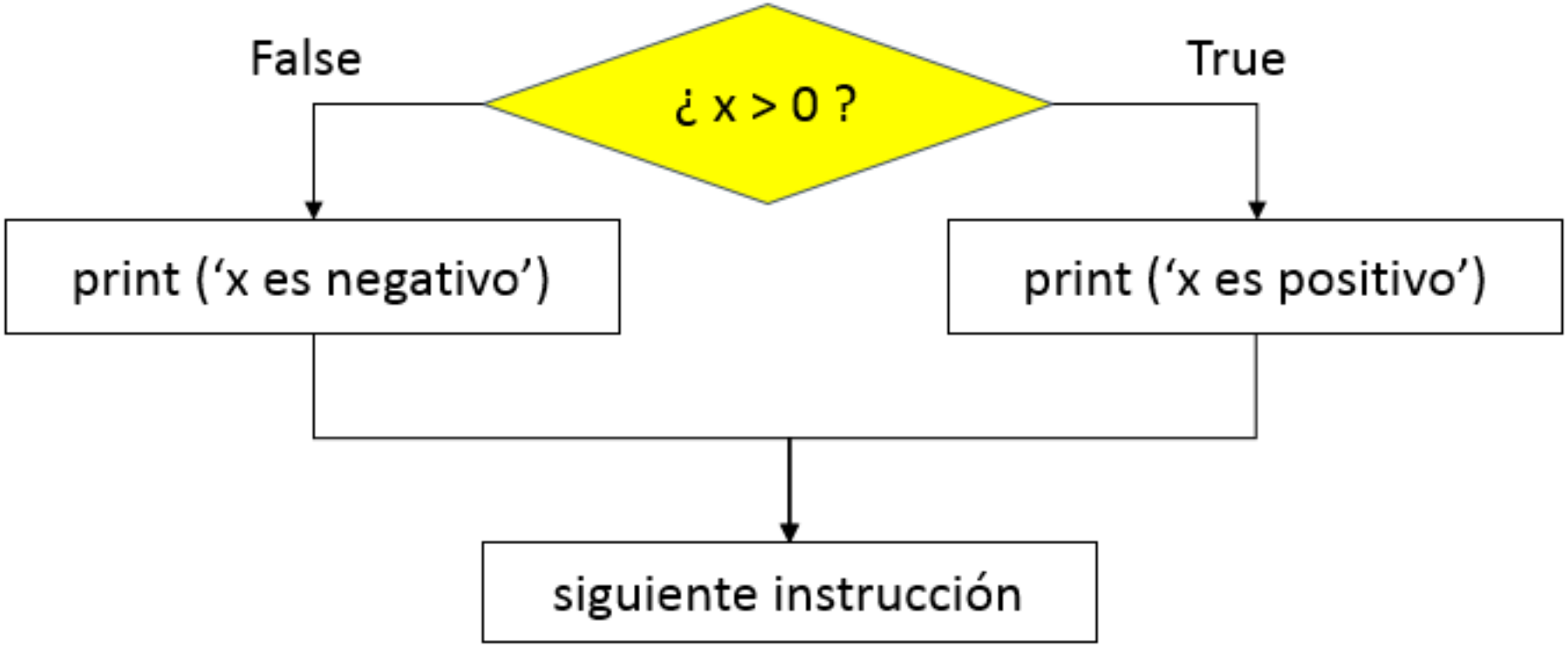
```
x = 5
if x > 2 :
    print 'Mayor de 2'
    print 'Aún es mayor'
print 'Hemos terminado con el 2'

for i in range(5) :
    print i
    if i > 2 :
        print 'Mayor que 2'
    print 'Terminado con i', i
print 'Todo terminado'
```

Flujo de control selectivo

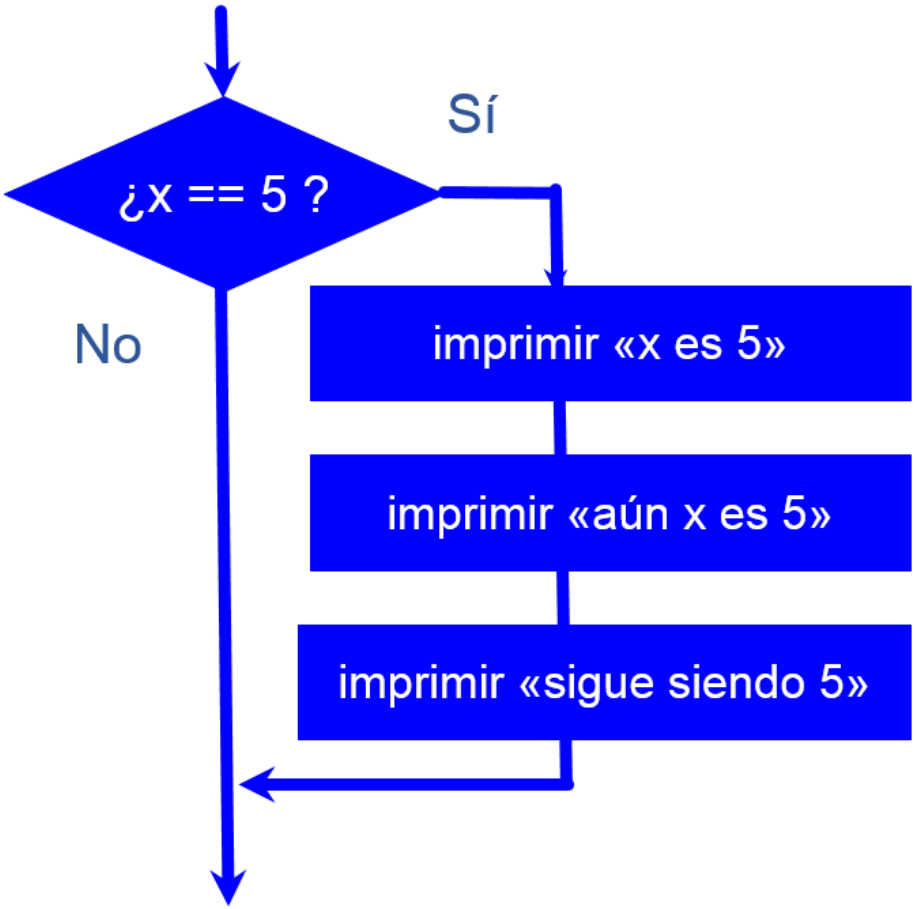


Flujo de control selectivo



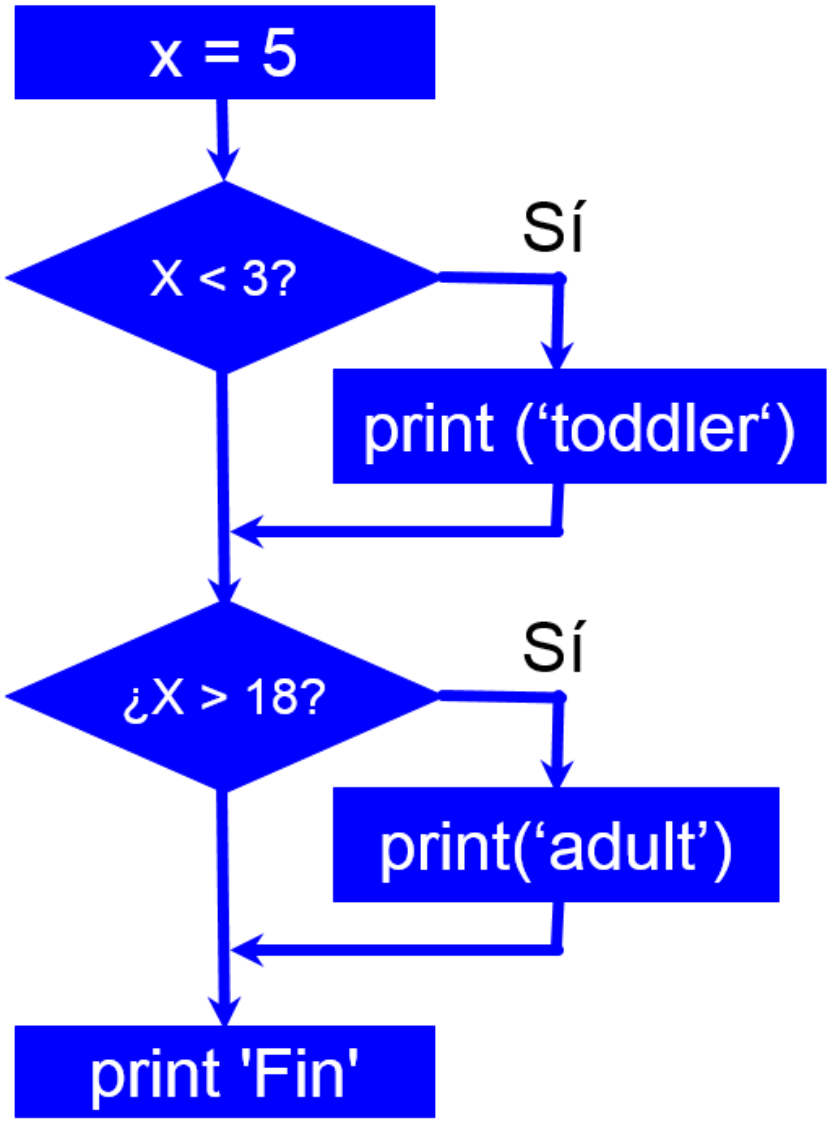

```
if x > 0:
   print('x es positivo')
else:
   print('x es negativo')
```

If simple




```
if x == 5:
  print ('x es 5')
  print ('aun x es 5')
  print ('sigue siendo 5')
```

Ifs sucesivos




```
x = 5
if x < 3:
   print ('toddler')
if x > 18:
   print ('adult')
print('Fin')
```

Ejemplo: Par o impar, modularizado

- Como toda expresión, la condición puede incluir llamadas a funciones.

Ejemplo: Par o impar, modularizado

- Como toda expresión, la condición puede incluir llamadas a funciones.
- Ej. pueden utilizarse funciones que retornan valores lógicos en la condición.

Ejemplo: Par o impar, modularizado

- Como toda expresión, la condición puede incluir llamadas a funciones.
- Ej. pueden utilizarse funciones que retornan valores lógicos en la condición.

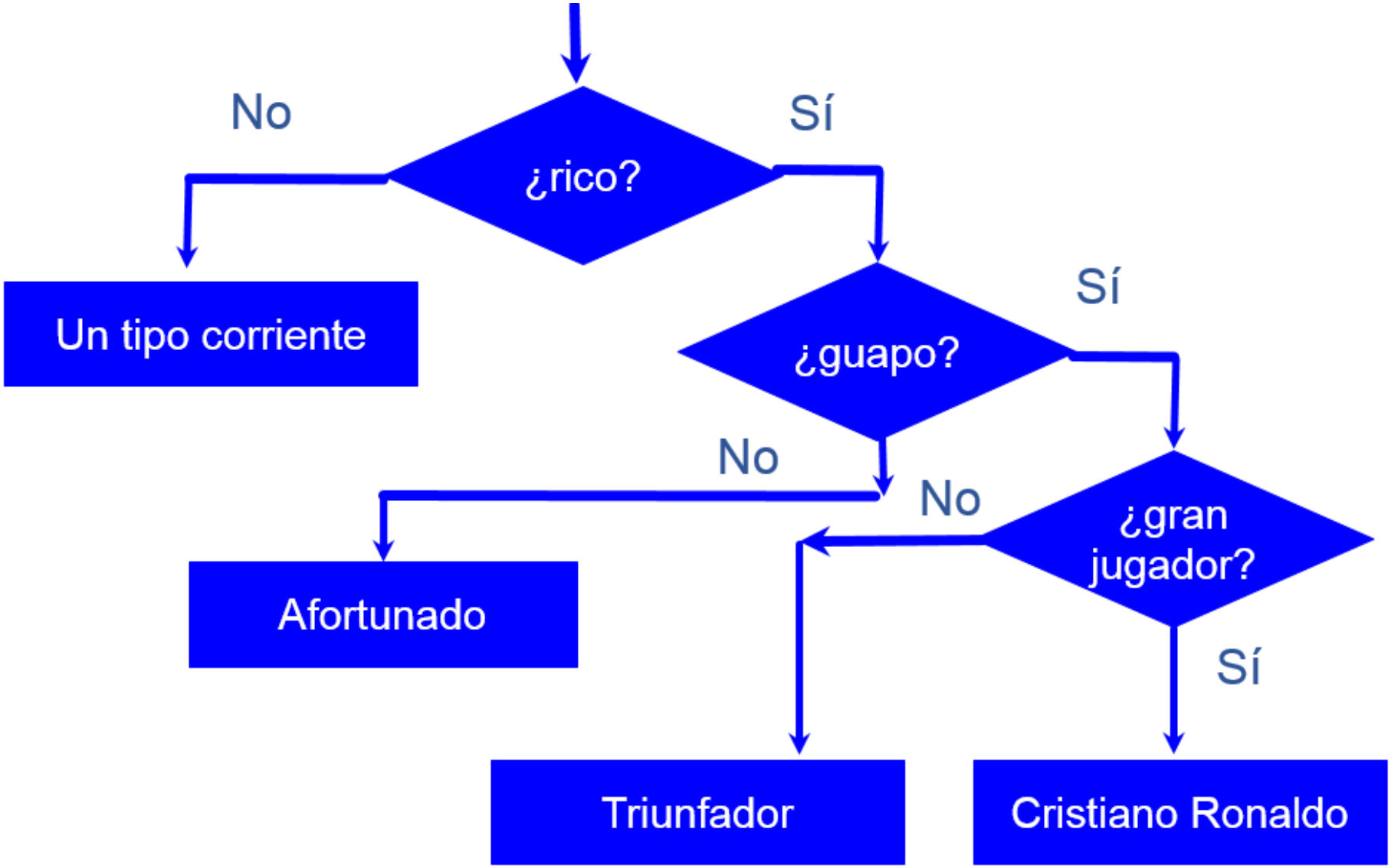
```
def es_par(x):
    return x % 2 == 0

n = int (input ("Introduce un entero: ")
if es_par(n):
    print("El numero es Par")
else:
    print("El numero es Impar")
```

Estructuras anidadas

- Los bloques “si” y “si no” de una selectiva (y de cualquier estructura de control) pueden a su vez contener otras estructuras de control.
- Por ejemplo: Un año es bisiesto si es divisible entre cuatro, excepto si es múltiplo de cien, en cuyo caso no es bisiesto salvo, a su vez, que sea múltiplo de cuatrocientos, en cuyo caso sí es bisiesto.

Ifs anidados



Estructuras selectivas múltiples

- Si la condición puede es de un tipo diferente al lógico permite más de dos flujos de control alternativos.

Estructuras selectivas múltiples

- Si la condición puede es de un tipo diferente al lógico permite más de dos flujos de control alternativos.
- La condición es una expresión, pero la decisión se toma de acuerdo a valores concretos o rangos de valores.

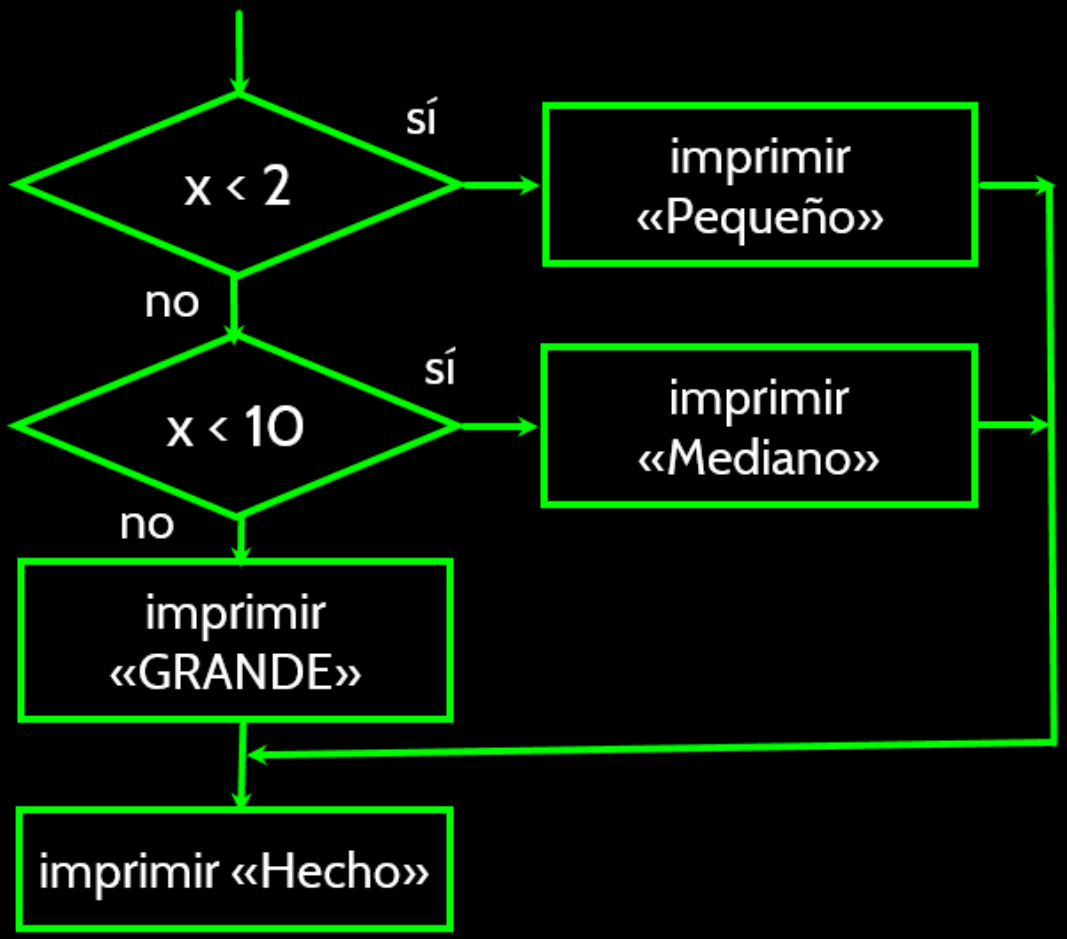
Estructuras selectivas múltiples

- Si la condición puede es de un tipo diferente al lógico permite más de dos flujos de control alternativos.
- La condición es una expresión, pero la decisión se toma de acuerdo a valores concretos o rangos de valores.

```
dia = int(input("Introduzca dia de la semana: " ))
if dia == 1    : print( "Lunes" )
elif dia == 2  : print( "Martes" )
elif dia == 3  : print( "Miercoles" )
        ...
elif dia == 6  : print( " Sabado" )
elif dia == 7  : print( " Domingo" )
else           : print( " ERROR: Dia incorrecto." )
```

Condicional múltiple

```
if x < 2 :
    print 'Pequeño'
elif x < 10 :
    print 'Mediano'
else :
    print 'GRANDE'
print 'Hecho'
```



Ejercicio

¿Cuál no se imprimirá?

```
if x < 2 :
    print('menor que 2')
elif x >= 2 :
    print('2 o más')
else :
    print('Otra cosa')
```

```
if x < 2 :
    print('menor que 2')
elif x < 20 :
    print('menor que 20')
elif x < 10 :
    print('menor que 10')
else :
    print('Otra cosa')
```

Ejercicio colectivo

Escribamos un programa para solicitar al usuario el número de horas y el precio por hora con vistas a calcular su salario bruto. Las horas que sobrepasen 40 se considerarán extra y pagadas a 1,5 veces el precio de la hora regular.

Ejercicio colectivo

Escribamos un programa para solicitar al usuario el número de horas y el precio por hora con vistas a calcular su salario bruto. Las horas que sobrepasen 40 se considerarán extra y pagadas a 1,5 veces el precio de la hora regular.

```
horas = float(input("Indique numero de horas trabajadas: "))
ratio = float(input("Introduzca el precio por hora: "))
if (0 < horas < 40) and (ratio > 0):
    print ("Sueldo final: ", horas * ratio)
elif (horas > 40) and (ratio > 0) :
    print ("Sueldo final: ", 40*ratio + (horas-40)*1.5*ratio)
else:
    print("Error: los datos son incorrectos.")
```

Excepciones

Excepciones

- En ocasiones, la detección de posibles errores con `if` resulta tediosa, pues modifica el aspecto del programa al llenarlo de comprobaciones.
- Las excepciones permiten separar el código "de negocio" del tratamiento de errores

```
try:
    acción potencialmente errónea
    acción potencialmente errónea
    ...
    acción potencialmente errónea
except:
    acción para tratar el error
```

Excepciones (ejemplo 1)

```
anno_actual = 2015
x = input("Introduzca el anno en que nacio: ")
try:
   anno_nacimiento = int(x)
except:
 print("Algo ha fallado: no es posible calcular su edad")

edad = anno_actual - anno_nacimiento
print("Su edad es: ", edad, " annos")
```

Excepciones (ejemplo 2)

```
anno_actual = 2015
x = input("Introduzca el anno en que nacio: ")
try:
  anno_nacimiento = int(x)
  edad = anno_actual - anno_nacimiento
  if edad<0:
      raise
  print("Su edad es: ", edad, " annos")
except:
 print("Algo ha fallado: no es posible calcular su edad")
```

Estructuras de control iterativas

El flujo de control: Iteraciones

Definición

Una estructura iterativa (bucle) engloba un conjunto de instrucciones que se ejecutan ninguna, una o tantas veces como indique una determinada condición.

Conceptualmente existen 3 tipos de bucles:

- *Desde* (número de iteraciones conocido)
- *Mientras* (0 o más iteraciones) y
- *Repetir* (1 o más iteraciones)

Bucles *desde* (*for*)

- Permite ejecutar una sentencia o bloque de sentencias un número conocido de veces.
- Itera sobre una lista de valores conocidos, bien numéricos (bastante frecuente) o de otro tipo.
- Una variable de control toma sucesivamente todos los valores de la lista.

```
for amiga in ["Marta", "Luna", "Ana", "Kira", "Patti"]:
  invitacion = amiga + ", te espero en mi fiesta."
  print(invitacion)
```

Función range() de Python

- Función que prepara una sucesión de elementos a menudo utilizados por una estructura `for`.
- Sintaxis:
  - `range (5)` : [0, 1, 2, 3, 4] desde 0 hasta el número menos 1
  - `range (2,5)` : [2, 3, 4] de 2 a 5 - 1
  - `range (3, 10, 2)` : [3, 5, 7, 9] de 3 a 10 menos 1 de 2 en 2

```
for i in range(10):
   print (i)
>>> 0 1 2 3 4 5 6 7 8 9
```

- Se lee: *"para todo elemento de la serie, hacer. . ."*

Ejemplo 1 *for*

¿Cuál es la salida de este código?

```
for i in range (1,20)
  print (i, end="")
```

Ejemplo 1 *for*

¿Cuál es la salida de este código?

```
    for i in range (1,20)
      print (i, end="")
```

>>>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Ejemplo 2 *for*

¿y de éste?

```
words = ['cat', 'dog', 'lion']
for w in words:
  print (w, end='','')
```

Ejemplo 2 *for*

¿y de éste?

```
words = ['cat', 'dog', 'lion']
for w in words:
  print (w, end='','')
```

>>>cat, dog, lion

Ejemplo 3 *for*

¿y de este otro?

```
x = 0
for i in range (1, 20, 2):
    x += i
print(x)
```

Ejemplo 3 *for*

¿y de este otro?

```
x = 0
for i in range (1, 20, 2):
    x += i
print(x)
```

>>>100

Ejemplo 4 *for*

Mostrar las tablas de multiplicar de los número pares

Ejemplo 4 *for*

Mostrar las tablas de multiplicar de los número pares

```
for i in range(2,11,2):
    for j in range(1,11):
        print(i,"x",j,"=",i*j)
    print("----------------")
```

Ejemplo 5 *for*

Obtener el factorial de un número

```
def factorial(n):
    """ int --> int
          OBJ: Factorial de n
          PRE: n >= 0 """
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result
```

Sentencia *while*

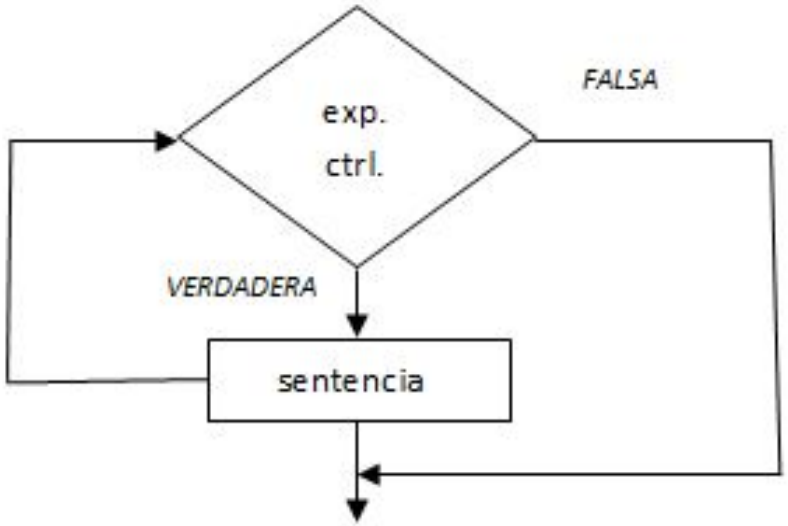
- Permite ejecutar una sentencia o bloque de sentencias 0 o más veces

```
while (condicion):
    sentencia(s)
```

Sentencia *while*

- Permite ejecutar una sentencia o bloque de sentencias 0 o más veces

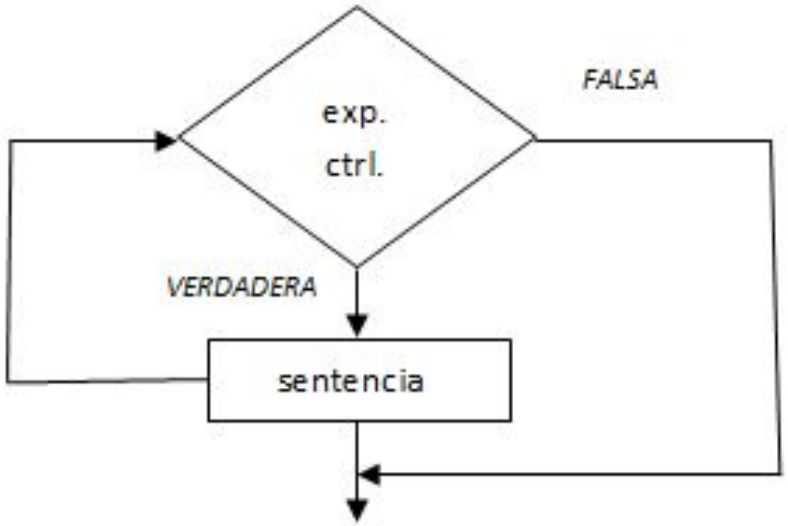
```
while (condicion):
    sentencia(s)
```



Sentencia *while*

- Permite ejecutar una sentencia o bloque de sentencias 0 o más veces

```
while (condicion):
    sentencia(s)
```

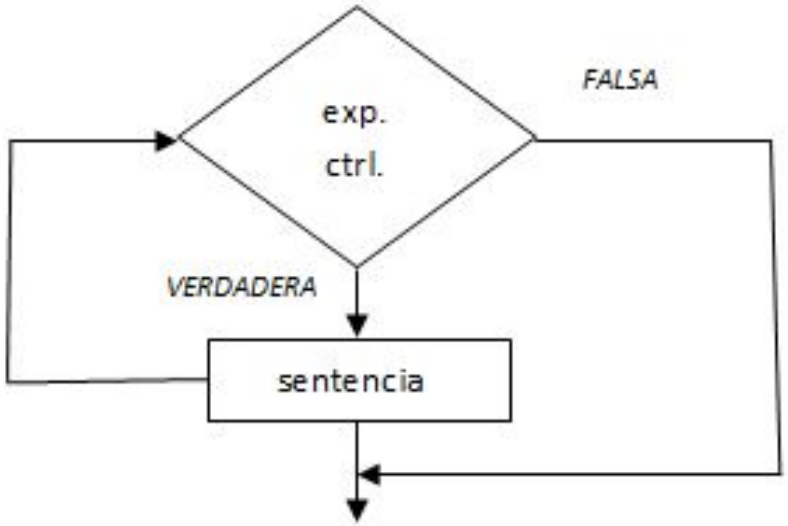


- Hay que estudiar detenidamente la expresión de control para que el bucle tenga fin.
- Las variables que intervienen en la condición deben modificarse dentro de la sentencia del bucle.

Sentencia *while*

- Permite ejecutar una sentencia o bloque de sentencias 0 o más veces

```
while (condicion):
    sentencia(s)
```



- Hay que estudiar detenidamente la expresión de control para que el bucle tenga fin.
- Las variables que intervienen en la condición deben modificarse dentro de la sentencia del bucle.

Bucles infinitos

- Ejecución continua de un bucle
- Efecto normalmente no deseado, derivado de un error en la condición o en la modificación de las variables que gobiernan el bucle

```
n = 10
while n>0:
    print (n)
    n = n + 1
print ('Listo!')
```

Ejemplo 1 *while*

Cuenta atrás desde 10 para el despegue...

Ejemplo 1 *while*

Cuenta atrás desde 10 para el despegue...

```
n = 10
while n > 0:
    print n
    n = n-1
print ('Despegue!')
```

Ejemplo 2 *while*

Invertir las cifras de un número entero.

```
inverso = 0
n = int(input("Entre el numero a invertir: "))
n = abs(n)
while (n > 0):
   inverso *= 10
   inverso += n % 10
   n //= 10
print("Resultado = ", inverso)
```

Resumen sentencia *while*

1. Se evalúa la condición, obteniendo `True` o `False`.
2. Si la condición es *False*, se sale de la sentencia *while* y el flujo de control continúa en la siguiente sentencia.
3. Si la condición es *True* ejecuta el cuerpo del bucle y vuelve al paso 1.

Este tipo de flujo se denomina *bucle* o *lazo* porque el tercer paso enlaza directamente con el primero estableciendo un mecanismo de repetición.

Salida forzada de un bucle: break

- `break` fuerza la salida de un bucle normalmente sujeto a una condición determinada.

```
while True:
    line = input('> ')
    if line == 'fin':
        break
    print (line)
print ('Fin!')
```

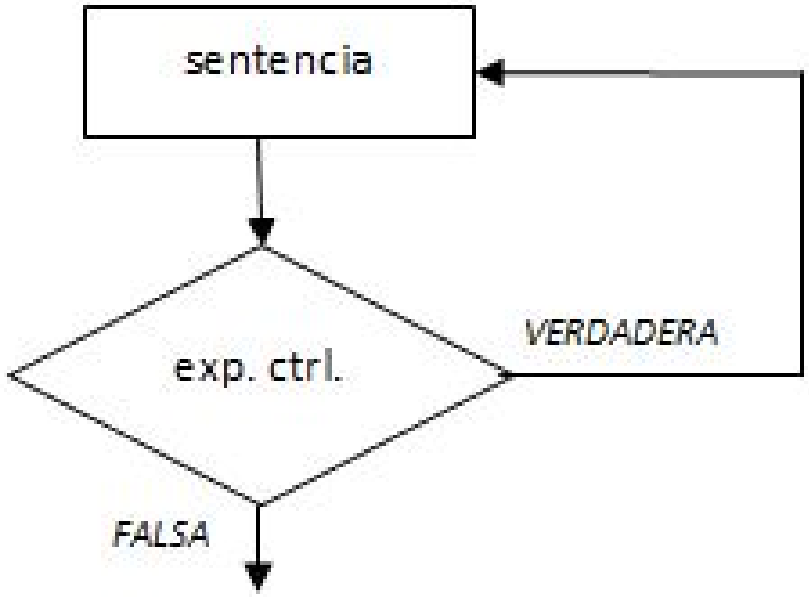
Salida forzada de un bucle: continue

- `continue` fuerza la terminación de la iteración actual y la evaluación de la condición que podría llevar a una nueva iteración o a terminar.

```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'fin':
        break
    print (line)
print ('Fin!')
```

Sentencia *repetir*

- Representa la repetición de una sentencia o bloque de sentencias 1 o más veces



- Python no ofrece una construcción específica para este concepto.

Implementación con *while*

Calcular la media de todos los números introducidos por teclado hasta que se teclee -1

Implementación con *while*

Calcular la media de todos los números introducidos por teclado hasta que se teclee -1

```
CENTINELA = -1
cuantos = 0
suma = 0
valor = int(input("Introduzca un valor (-1
    para terminar):"))
while (valor != CENTINELA):
    suma += valor
    cuantos += 1
    valor = int(input("Introduzca un valor (-1
        para terminar):"))
if (cuantos>0):
    print ("La media es: ", suma/cuantos)
```

Implementación con `while True`

Comprobar que un número leído está entre 1 y 7...

```
while True:
    i = input("Entre un numero entre 1 y 7: ")
    if 1 <= int(i) <= 7: break
print('Correcto, gracias!')
```

Bucles anidados

Obtener la siguiente secuencia:

```
1  =  1
1+2  =3
1+2+3=6
1+2+3+4=10
....
```

Bucles anidados

```
count = 10 # Numero de sumas
for i in range(1, count+1):
      sum = 1
      j = 1
      print("1", end="");
      # Calcula la suma de los enteros de 1 hasta i
      while(j < i):
        j = j + 1
        sum = sum + j
        print(" + ", j, end="") # Escribir +j en misma linea
      print(" = ", sum)
```

¿Podría ser el bucle interno de tipo for? ¿y de tipo do...while? ¿qué es más adecuado, si el número de iteraciones está determinado por *count*?

Resumen

- Las estructuras selectivas permiten ejecutar bloques de código diferentes dependiendo de una condición.
- Las estructuras iterativas permiten ejecutar bloques de código repetidas veces.
- Existen varios tipo de bucles: 0-n, 1-n (repetir), for.
- Aunque poco ortodoxas, existen formas de salir de un bucle a mitad de una iteración (`break` y `continue`)
- Las estructuras de control pueden anidarse

Referencias y reconocimientos

- Algunos contenidos de esta presentación han sido adaptados de los materiales del curso de "Programming for Everybody (Python)", creado por Charles Severance y disponible en `https://www.coursera.org/course/pythonlearn`.
- También hemos tomado explicaciones del libro "Introducción a la programación con Python 3" de Marzal y su equipo de la Univ. Jaume I: `http://issuu.com/universitatjaumei/docs/sapientia_93`