

# Condicionales



# Las sentencias condicionales

- Nos permiten comprobar condiciones.
- Decidir cómo se comportará nuestro programa.
- Decidir qué fragmento de programa se ejecutará.

## if

La forma más simple de un estamento condicional es un `if` (del inglés *si*) seguido de la condición a evaluar, dos puntos (`:`) y en la siguiente línea e indentado, el código a ejecutar en caso de que se cumpla dicha condición.

```
if condición:
```

```
    acción
```

```
    acción
```

```
    ...
```

```
    acción
```

```
comida = "Empanadas"
if comida == "Empanadas":
    print("Tenes buen gusto!!")
    print("Gracias")
```

42

```
comida "Empanadas"
```

```
def pedir_comida(comida):
```

```
    comida "Empanadas":
```

```
        print("Tenes buen gusto!!")
```

```
42 | print("Gracias")
```

## if ... else

Vamos a ver ahora un condicional algo más complicado. ¿Qué haríamos si quisiéramos que se ejecutaran unas ciertas órdenes en el caso de que la condición no se cumpliera? Sin duda podríamos añadir otro if que tuviera como condición la negación del primero:

```
if condición:  
| acciones  
if condición contraria:  
| otras acciones
```

Este tipo de combinación es muy frecuente, hasta el punto de que se ha incorporado al lenguaje de programación una forma abreviada que significa lo mismo:

```
if condición:  
| acciones  
else:  
| otras acciones
```

```
comida = "Empanadas"
comida = input("¿Que comida te gusta? ")

if comida == "Empanadas":
    print("Tenes buen gusto!!")
    print("Gracias")
elif comida == "Empanadas":
    print("Vaya, que lastima")
```

# forma compacta para estructuras condicionales múltiples

```
if condición:  
    ...  
else:  
    if otra condición:  
        ...
```

Un **else** inmediatamente seguido por un **if** puede escribirse así:

```
if condición:  
    ...  
elif otra condición:  
    ...
```

De esta manera nos ahorramos la indentación.

## if ... elif ... elif ... else

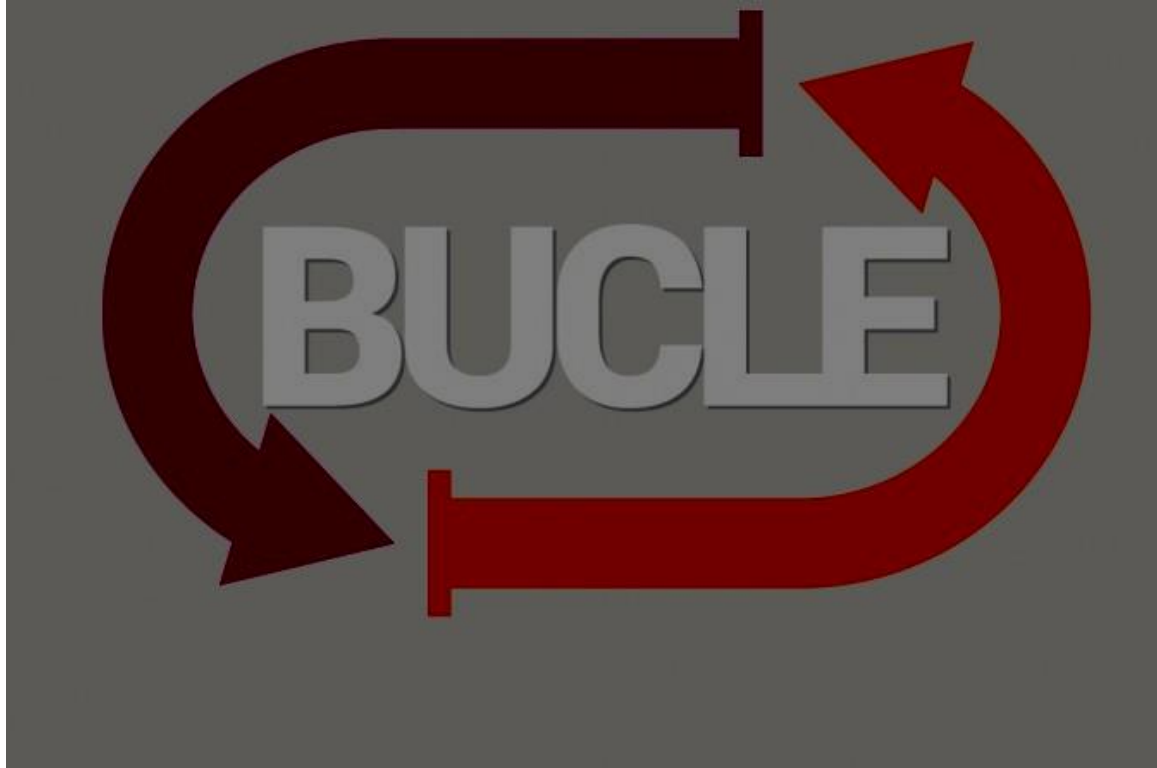
Todavía queda una construcción más que ver, que es la que hace uso del `elif`.

```
if numero < 0:
    print "Negativo"
elif numero > 0:
    print "Positivo"
else:
    print "Cero"
```

`elif` es una contracción de *else if*, por lo tanto `elif numero > 0` puede leerse como “si no, si numero es mayor que 0”. Es decir, primero se evalúa la condición del `if`. Si es cierta, se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple, se evalúa la condición del `elif`. Si se cumple la condición del `elif` se ejecuta su código y se continua ejecutando el código posterior al condicional; si no se cumple y hay más de un `elif` se continúa con el siguiente en orden de aparición. Si no se cumple la condición del `if` ni de ninguno de los `elif`, se ejecuta el código del `else`.



# Bucles



### 4.2.1. La sentencia while

En inglés, «while» significa «mientras». La sentencia **while** se usa así:

**while** *condición* :

| *acción*

| *acción*

| ...

| *acción*


y permite expresar en Python acciones cuyo significado es:

«*Mientras* se cumpla esta condición, repite estas acciones.»

Las sentencias que denotan repetición se denominan *bucles*.


```
1 i = 0
2 while i < 3:
3     print i
4     i += 1
5 print 'Hecho'
```

```
i = 0
while i < 3:
    print i
    i += 1
print 'Hecho'
```



⇐ la condición se satisface

```
i = 0
while i < 3:
    print i
    i += 1
print 'Hecho'
```



⇐ la condición no se satisface

## 4.2.5. El bucle for-in

Hay otro tipo de bucle en Python: el bucle **for-in**, que se puede leer como «para todo elemento de una serie, hacer...». Un bucle **for-in** presenta el siguiente aspecto:

```
for variable in serie de valores:
```

```
    acción
```

```
    acción
```

```
    ...
```

```
    acción
```

```
secuencia = ["uno", "dos", "tres"]
```

```
for elemento in secuencia:
```

```
    print (elemento)
```

```
uno
```

```
dos
```

```
tres
```

“para cada elemento en secuencia”. Y esto es exactamente lo que hace el bucle: para cada elemento que tengamos en la secuencia, ejecuta

# range()

- Es una función predefinida de Python
- Significa “rango”
- En principio lleva como parámetros dos valores: *valor inicial*, *valor final*
- Por defecto recorre desde el primer elemento hasta el último-1.

```
>>> range(2, 10) ↓  
[2, 3, 4, 5, 6, 7, 8, 9]  
>>> range(0, 3) ↓  
[0, 1, 2]  
>>> range(-3, 3) ↓  
[-3, -2, -1, 0, 1, 2]
```

# Las variantes son:

```
>>> range(5) ↵  
[0, 1, 2, 3, 4]
```

Si usamos tres argumentos, el tercero permite especificar un *incremento* para la serie de valores. Observa en estos ejemplos qué listas de enteros devuelve *range*:

```
>>> range(2, 10, 2) ↵  
[2, 4, 6, 8]  
>>> range(2, 10, 3) ↵  
[2, 5, 8]
```

Fíjate en que si pones un incremento negativo (un *decremento*), la lista va de los valores altos a los bajos. Recuerda que con *range* el último elemento de la lista no llega a ser el valor final

```
>>> range(10, 5, -1) ↵  
[10, 9, 8, 7, 6]  
>>> range(3, -1, -1) ↵  
[3, 2, 1, 0]
```



contador\_con\_for.py

contador\_con\_for.py

```
1 for i in range(1, 6):  
2     print i
```

Al ejecutar el programa, veremos lo siguiente por pantalla:

```
1  
2  
3  
4  
5
```

La lista que devuelve *range* es usada por el bucle **for-in** como serie de valores a recorrer.