Programación I
Primeros pasos en Python

Pablo Garaizar Sagarminaga
Borja Sanz Urquijo

Facultad de Ingeniería

Índice

- Hola mundo.
- Variables y tipos de datos.
- Entrada/Salida básica.
- Condicionales.
- Bucles.

Hola mundo

El intérprete de Python

- En otros lenguajes (ej: C):
  - Programar → **Compilar** → Ejecutar
  - El código fuente se convierte en binario antes de ejecutar.
- En Python, normalmente:
  - Programar → **Interpretar** (Ejecutar)
  - El código fuente se interpreta y ejecuta al vuelo.

Hola mundo en consola

```
python
>>> print('Hello world!')
Hello world!
```

Hola mundo en fichero

- En hola.py:

```
# Mi primer programa
print('Hello world!')
```

- Interpretar:

```
python hola.py
Hello world!
```

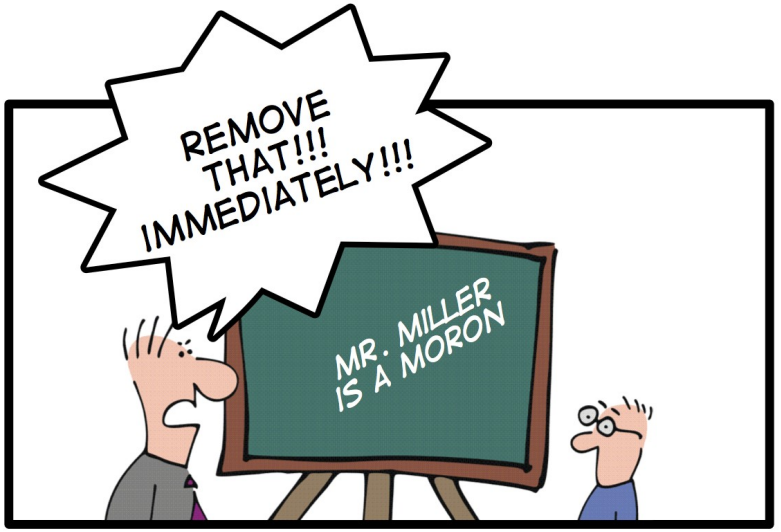
Hola mundo

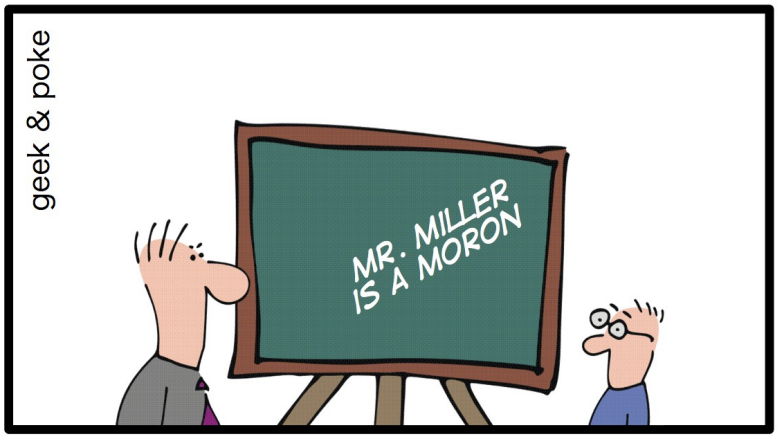
- print() es una **función**:
  - Sus argumentos van entre **paréntesis**.
  - **Imprime** por pantalla lo que le pasemos como argumento.

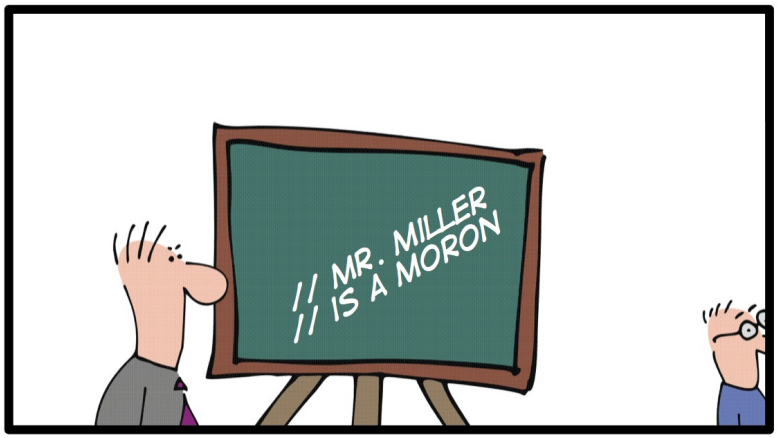
Comentarios

- Es muy importante comentar nuestro código:
  - Mejor explicar **significado** que **sintaxis**:
    - Mal comentario: (a >= 65) # es a mayor que 65?
    - Buen comentario: (a >= 65) # está jubilado?
  - Comentar → **Documentar**.
- Cómo:

```
# Comentario de línea
''' Bloque de comentario '''
```







RECENTLY IN THE CLASSROOM

Variables y tipos de datos

Variables

- Los **datos** de un programa se almacenan en unas **“cajas”** denominadas **variables**.
  - En función de qué queramos guardar, necesitamos una **tipo** de caja u otra:
    - Número entero → int
    - Número decimal → float
    - Frase → str
    - Lógico → bool

Variables

- En Python los tipos son **dinámicos**:
  - Puedes decir que la variable **a** es un entero y luego que es un booleano.

```
a = 1
a = True
```

Variables

- **Nombres** de las variables:
  - **No** pueden empezar por número.
  - **No** pueden contener caracteres especiales (!).
  - **No** pueden contener espacios.
  - Si necesitamos varias palabras, usamos **camelCase**:

```
puntosExtraPorPeloteo = 4
```

https://www.python.org/dev/peps/pep-0008/#naming-conventions

Variables

- Asignación múltiple:

```
a, b = 1, 2 # a = 1 y b = 2
a, b = b, a # intercambiamos los valores de a y b
```

Operadores aritméticos

- **Suma**: 4 + 2
- **Resta**: 4 – 2
- **Multiplicación**: 4 * 2
- **División**: 4 / 2
- **División entera**: 5 // 2
- **Resto**: 5 % 2
- **Potencia**: 5 ** 2

Incremento / decremento

- **Incremento**:
  - a = a + 1
  - a += 1
- **Decremento**:
  - a = a - 1
  - a -= 1

Precedencia

- *¿Cuánto es 5 * 3 + 2?*
- ***PEMDAS**:*

  1) Paréntesis
  2) Exponentes
  3) Multiplicaciones y Divisiones
  4) Sumas y restas

  Resultado: 17.

Números decimales

- También suelen llamarse **reales** o de **“coma flotante”**.
- Las siguientes operaciones dan como resultado un **float**:
  - int / int.
  - Cualquier operación de int con float.
  - Cualquier operación de float con float.

Booleanos

- Una variable booleana solo puede ser **True** o **False**.
  - Son muy útiles para definir **condiciones** en alternativas (if) o bucles (for, while, etc.).

```
es_lunes = True


if(es_lunes):
  print('Aumentando las ganas de llorar')
else:
  print('¡¡Ya queda menos para el viernes!!')
```

Operadores booleanos

- **Lógicos**:
  - AND: a and b
  - OR: a or b
  - NOT: not a
- **Relacionales**:
  - Igual: == (ojo, **DOS** símbolos de igualdad)
  - Distinto: !=
  - Mayor: > Mayor o igual: >=
  - Menor: < Menor o igual: <=

Strings

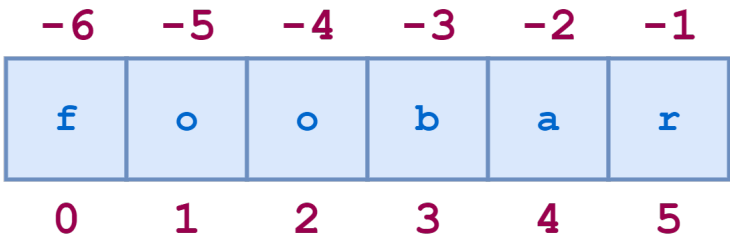
- Son “cadenas de caracteres”, es decir, palabras o frases:

```
nombre = “Kepa Sakolegi”
```

- Podemos acceder a un carácter concreto usando corchetes [ ]:

```
print(nombre[1])
```

  - Los índices empiezan en 0.
  - Si queremos empezar por el final → índices negativos.



Strings

- Operadores para strings:
  - + → concatenar

    `nombre = “Kepa” + “Sakolegi”`
  - * → copiar n veces (entero)

    `batman = “na” * 20`
  - in → ¿existe esto dentro del string?

    `“hola” in “hola mundo” # Devuelve True`

Strings

- Funciones interesantes:

```
print(s.capitalize()) # Primera letra en mayúsculas

print(s.lower()) # Todo a minúsculas

print(s.upper()) # Todo a mayúsuculas

print(s.swapcase()) # Inversión

print(s.title()) #Primera letra de cada palabra en
mayúsculas

print(s.islower()) # Devuelve True si está en
minúsculas
```

Strings

- Strings con formato:

```
print(f‘El nombre de usuario {username} accedió al sistema el {fecha} por última vez’)
```

https://realpython.com/python-string-formatting/

Conversión de tipos

- Existen maneras de convertir un tipo de datos en otro:
  - **Promociones (casting)**:

    ```
    suma = 12 + int(“52”)
    ```

  - **Coerciones**: los int se convierten automáticamente a floats aunque no lo pidamos

    ```
    nota = 5.0 + 2
    ```

Listas

- A veces, queremos guardar varios datos en un mismo sitio.
  - En lugar de hacer esto:

```
edad1 = 18
edad2 = 19
edad3 = 18
edad4 = 17
edad5 = 20
```

  - Podemos crear una **lista** así:

```
edades = [ 18, 19, 18, 17, 20 ];
```

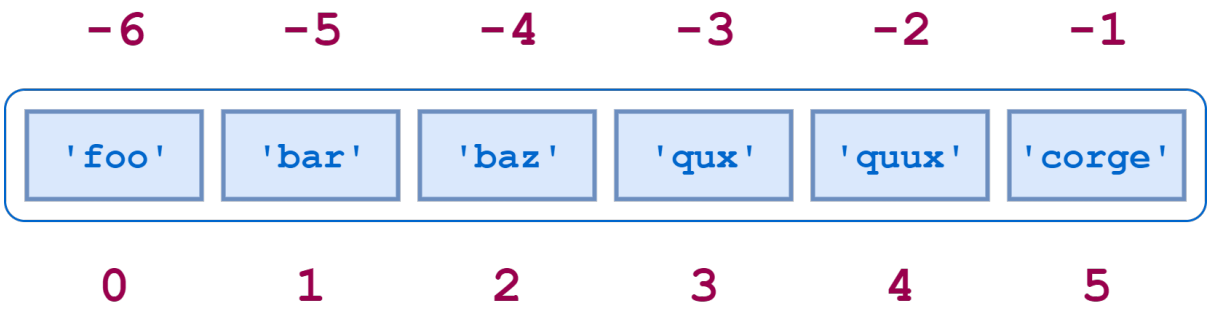
Listas

- Características importantes de las listas en Python:
  - Están **ordenadas**.
  - Pueden contener **cualquier número** de elementos y de distintos **tipos**.
  - Los elementos pueden accederse por su **índice**.
  - Se puede **anidar** listas dentro de otras listas.
  - Son mutables y dinámicas, pueden **cambiar**.

Listas

- Los **índices** funcionan de forma parecida a los de los strings:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```



- También se pueden definir **rangos** ([inicio:fin]) o **strides** ([inicio:fin:salto]):

```
print(a[0:3])
print(a[1:6:2])
```

Listas

- Si no defines **inicio** o **fin** en un rango o stride, se entiende que empiezan desde 0 y terminan al final de la lista.
- Llamadas interesantes:

```
print(lista[::-1]) # Imprime la lista de atrás a delante
copia = lista[:] # Hace una copia de lista
```

Listas

- Tamaño: len, min, max

```
print(len(lista)) # Imprime el tamaño de la lista
```

- Insertar en una lista:

```
lista.append('mundo') # Al final

lista.insert(2, 'hola') # En la posición 2

lista.index('hola') # ¿En qué posición está 'hola'?
```

- Pertenencia: in

```
print('elemento' in lista) # True si está en la lista
```

Listas

- Operaciones con listas:
  - Insertar:

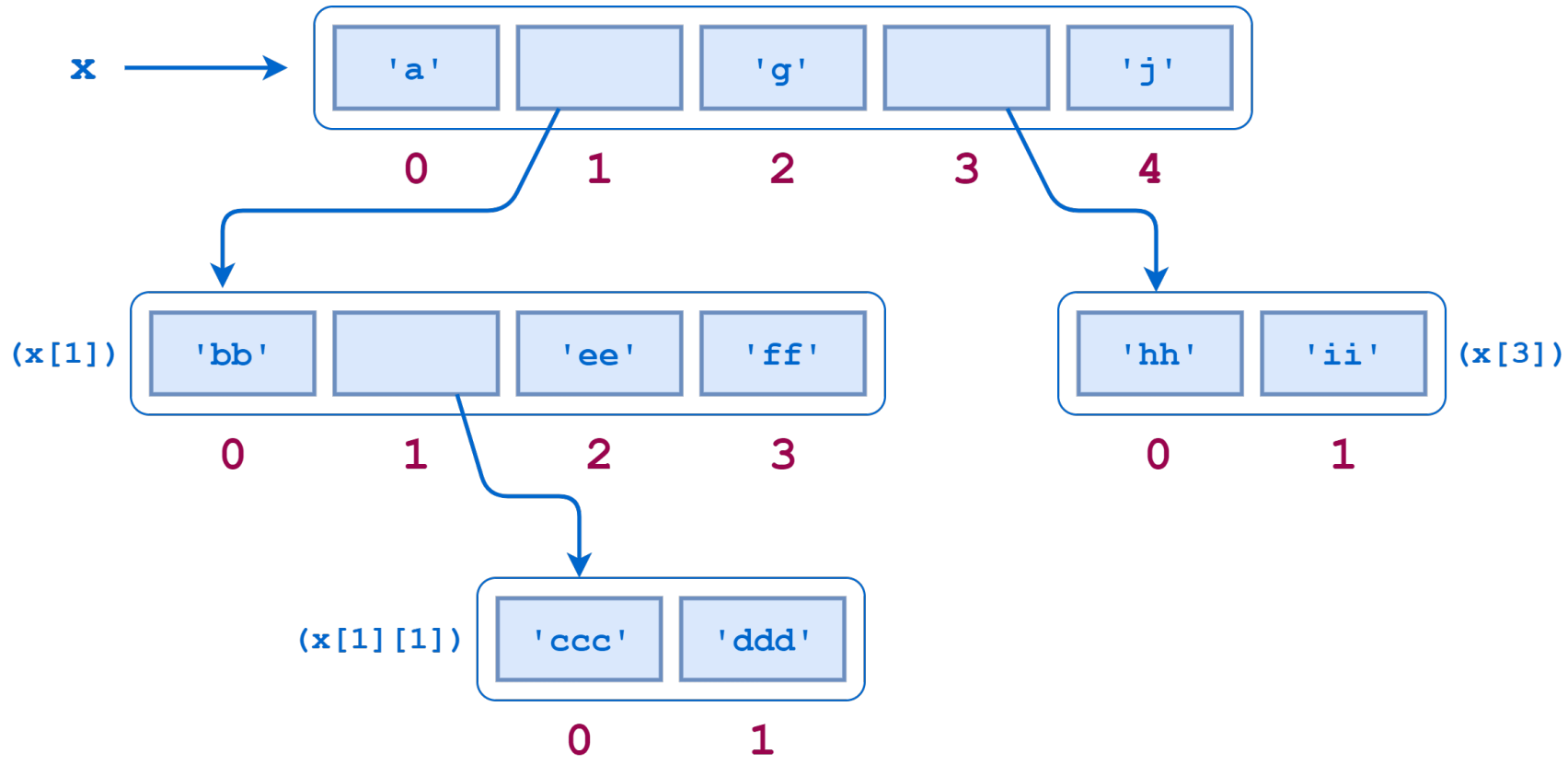
```
lista = lista + [‘hola’, ‘mundo’] # Al final
lista = [‘hola’, ‘mundo’] + lista # Al principio
lista[1:2] = [‘hola’, ‘mundo’] # Entre el 2º y 3º
```

  - Multiplicar:

```
lista = lista * 3 # Triplica los elementos de la lista
```

Listas anidadas

```
x = ['a', ['bb', ['ccc', 'ddd'], 'ee', 'ff'], 'g', ['hh', 'ii'], 'j']
```



Constantes

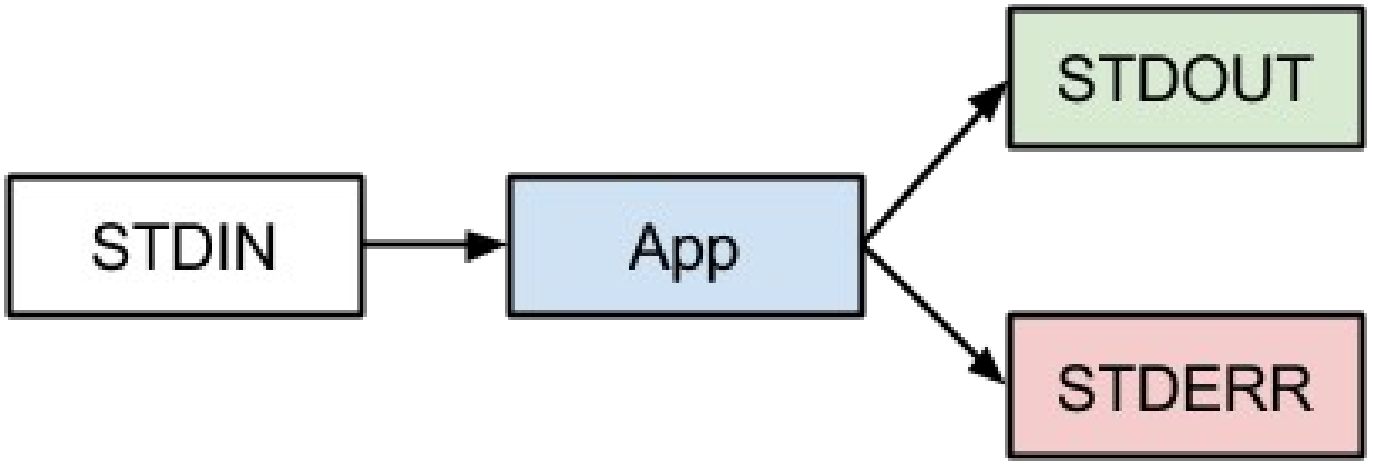
- Hay valores que no son variables, ejemplo: PI.

```
PI = 3.1415927
```

- Normalmente, se nombran en todo en **mayúscula**.

Entrada/Salida básica

E/S básica



E/S básica

- **Salida**:
  - Función print()

```
print(“Aquí va una línea\nAquí un \t tabulador”)
```

- **Entrada**:
  - Función input()

```
ciudad = input(“Dime el nombre de la ciudad:”)
temp = float(input(“Dime su temperatura:”))
```

Condicionales

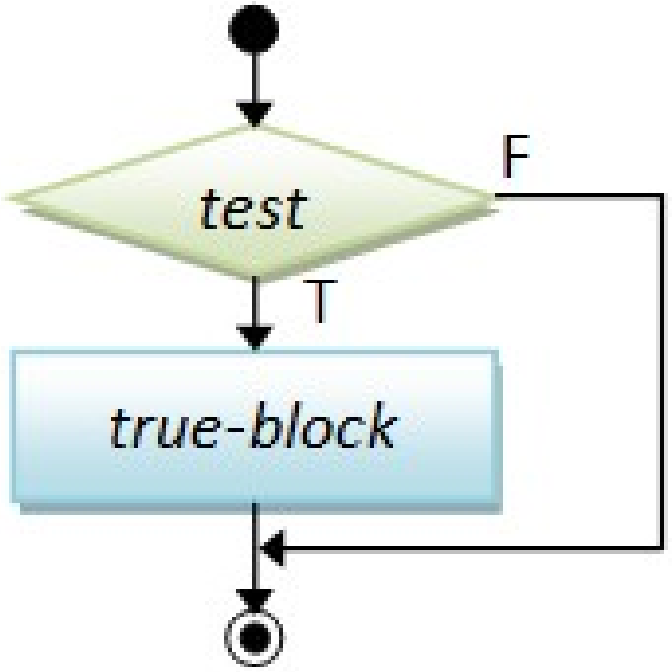
Condicionales

- También llamadas **“alternativas”**.
- Estructuras que cambian el **flujo de ejecución** de un programa.
  - A veces hay que hacer A, otras veces, B.
  - **¿Cómo decidir?** → en función de una **condición**.

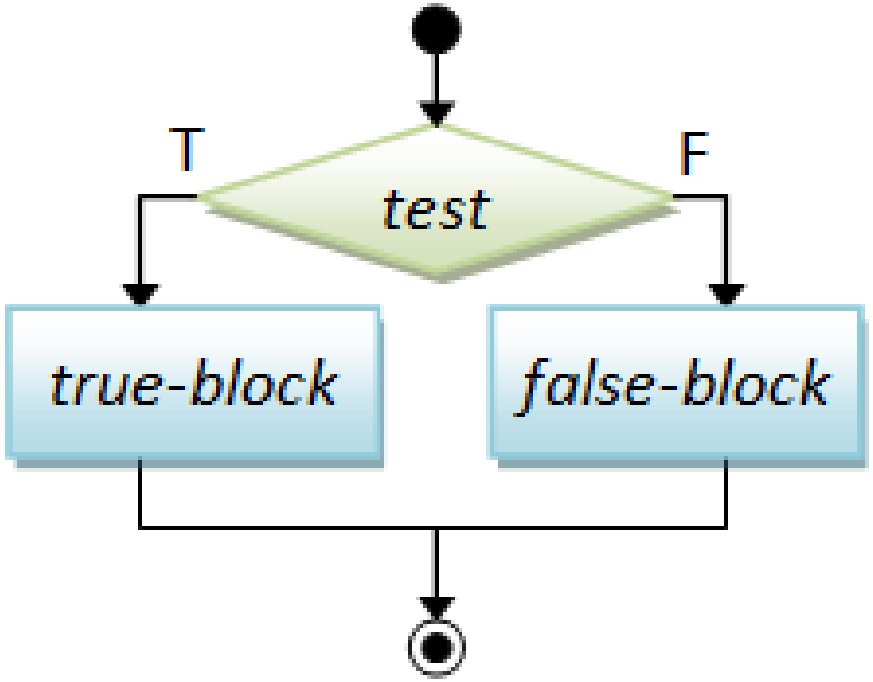
```
if (puntos > 1000):
  vidas = vidas + 1
```

```
if (dinero < 0):
  print(“Bancarrota”)
else:
  print(“Vamos de compras”)
```

if



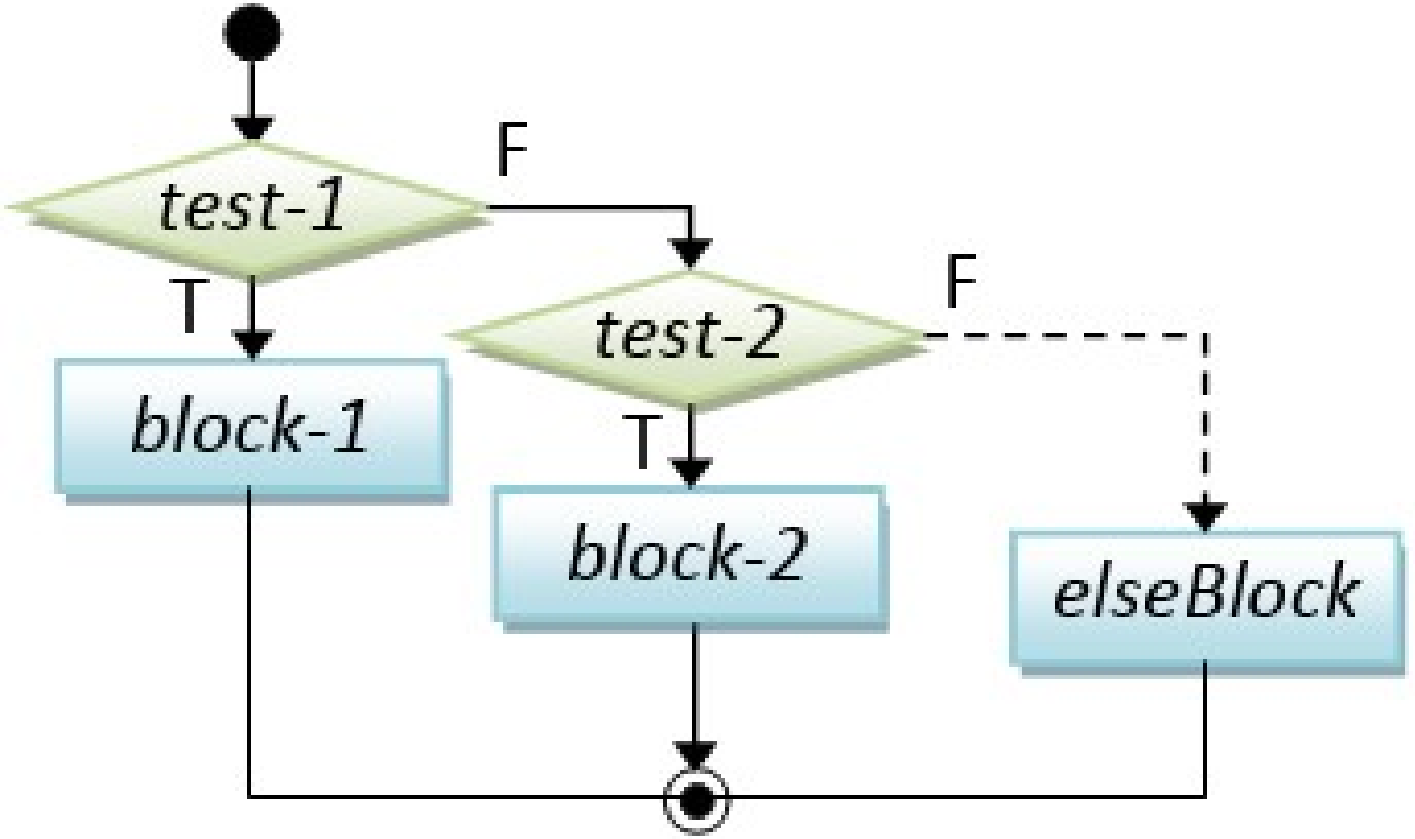
if-else



if encadenados

```
if (edad < 4):
   tarifa = 0 # tarifa infantil
elif (edad < 18):
   tarifa = 1 # tarifa juvenil
elif (edad < 65):
   tarifa = 2 # tarifa adulta
else:
   tarifa = 3 # tarifa jubilación
```

if encadenados

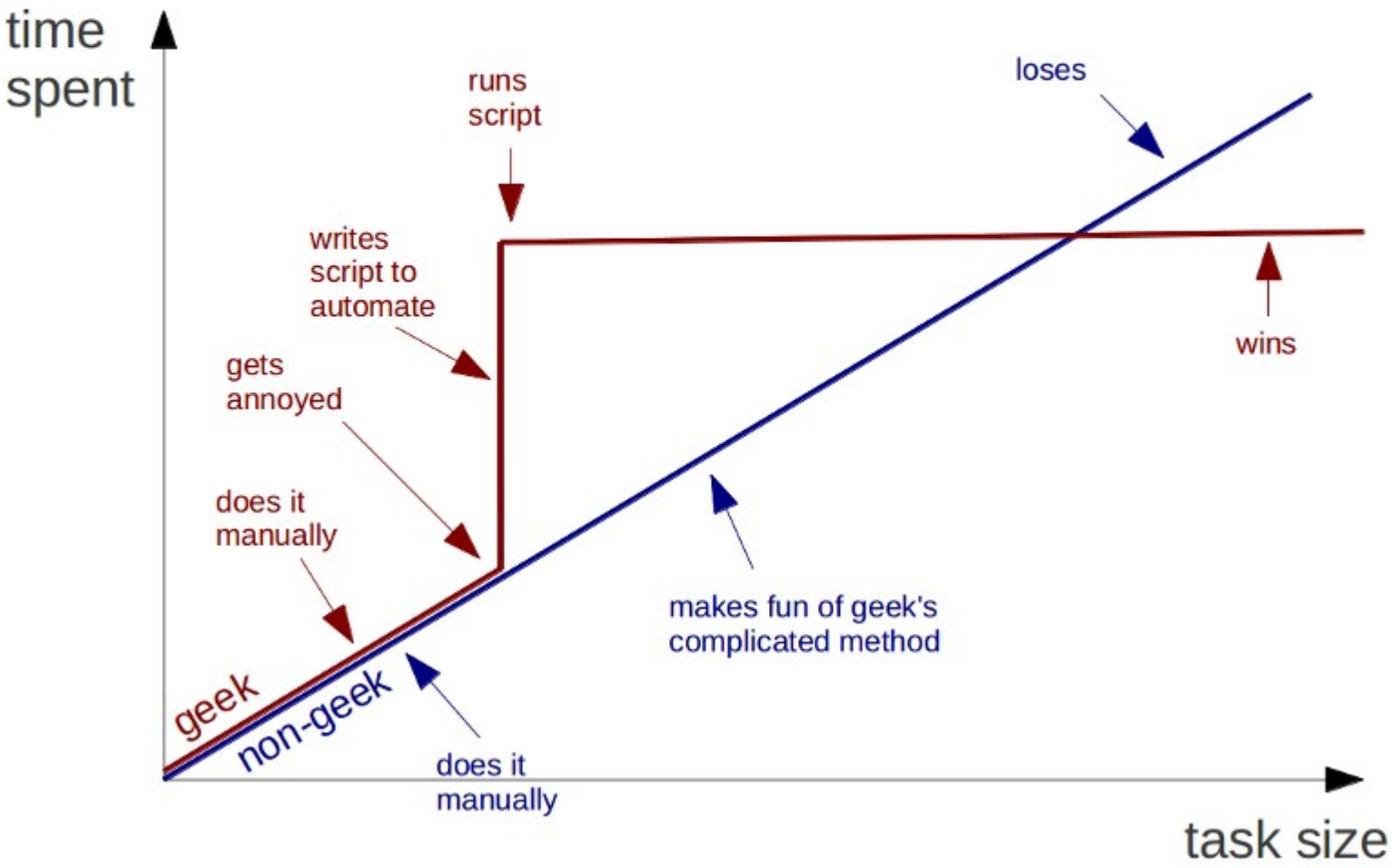


Bucles

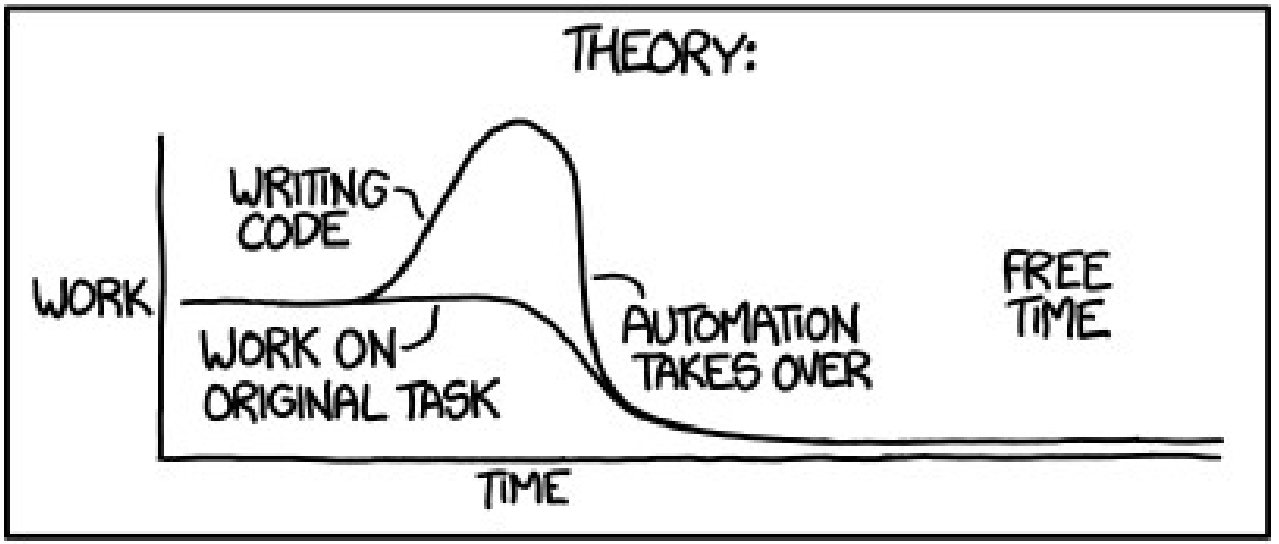
Bucles

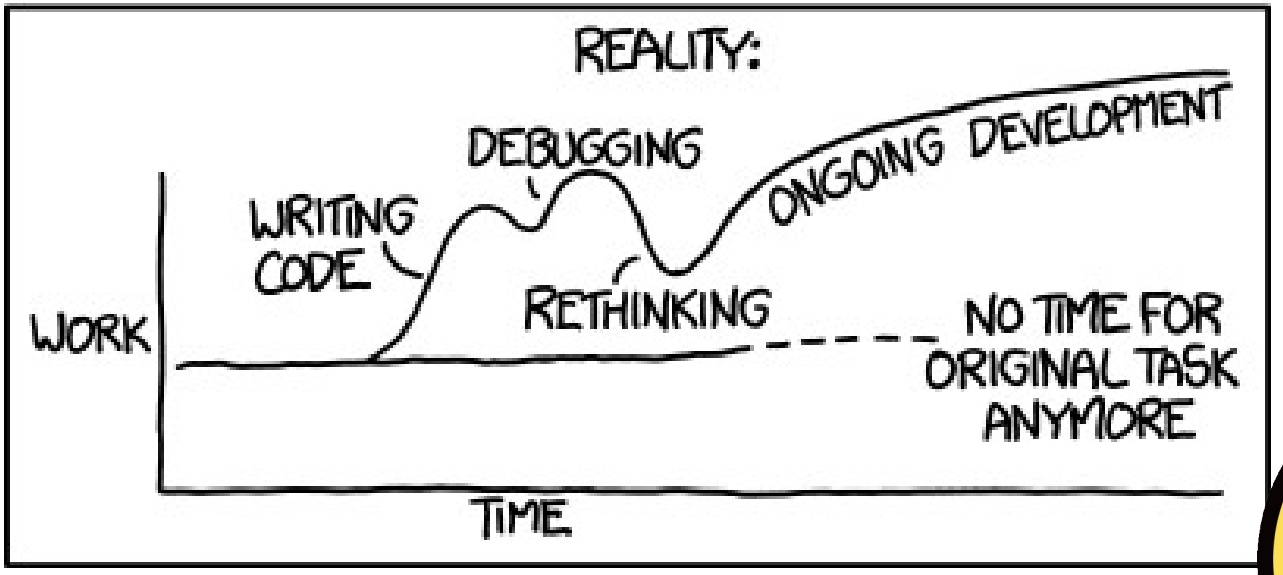
- Hacer tareas **repetitivas** es la base de la computación.
- Diferentes formas de hacerlo:
  - **for**: de 0 a N veces, sabemos **cuántas**.
  - **while**: de 0 a N veces, a veces **no sabemos** cuántas.

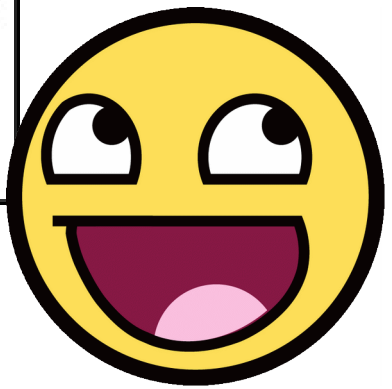
Geeks and repetitive tasks



"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"







while

```
while (condición):
    # código-while
```

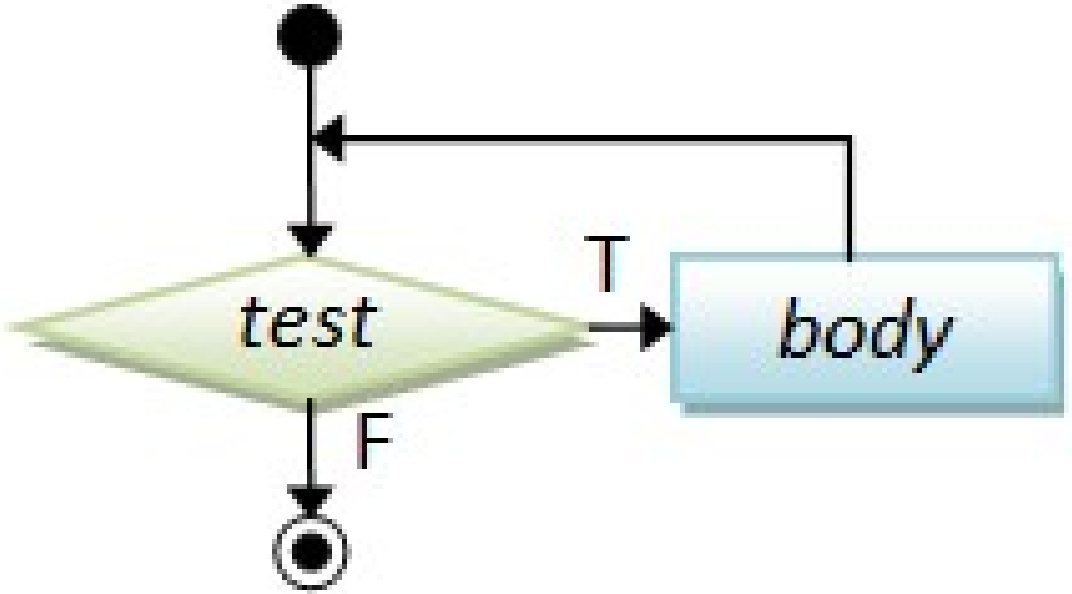
- Si se cumple la condición, ejecutamos el bloque de código dentro del while.
- Si no, salimos.
  - Es habitual que dentro del código-while haya algo que modifique el valor de la condición.
    - Si no → **bucle infinito**.

while

- Ejemplo:

```
i = 10
while (i > 0):
   print(i)
   i = i - 1
```

while



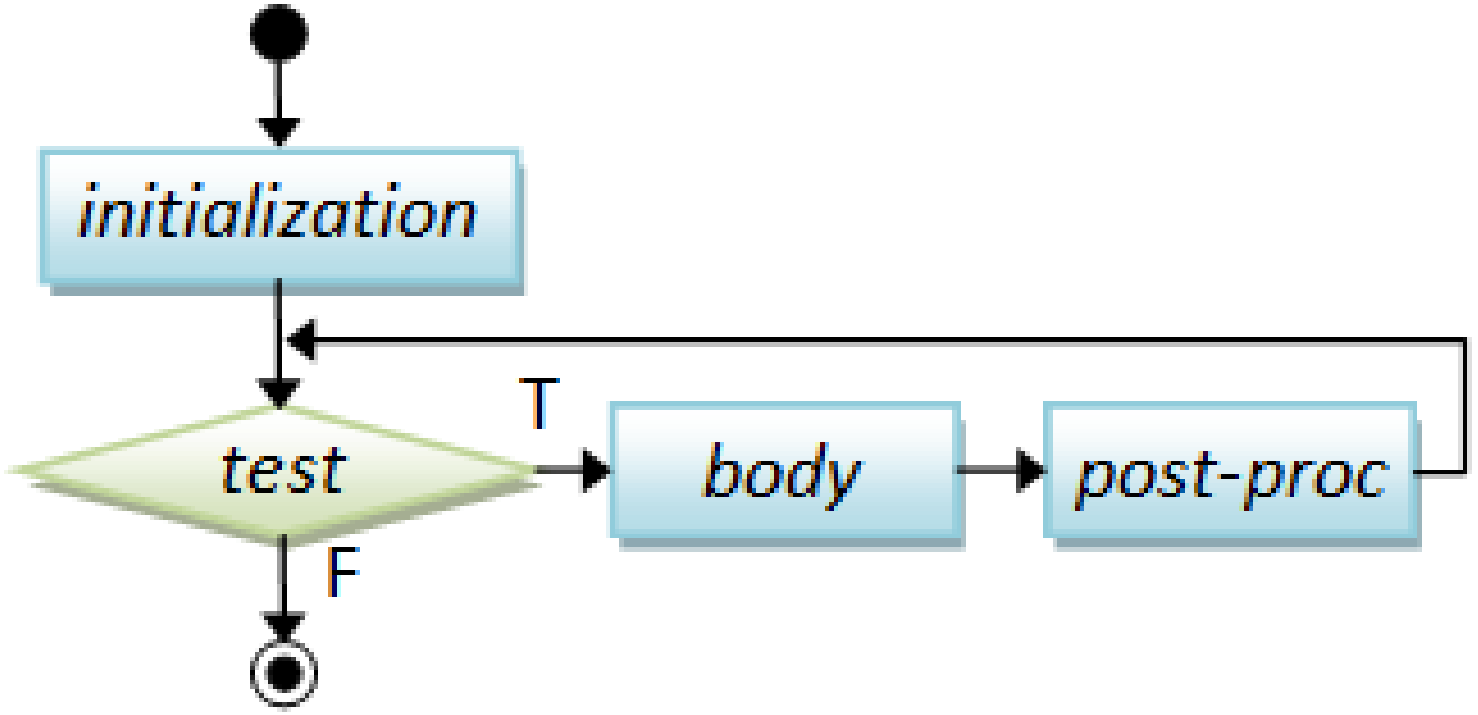
for

```
for variable in lista:
    # código que usa “variable”
```

- Ejemplos:

```
for numero in [0, 1, 2, 3, 4, 5]:
    print(numero)
for numero in range(0, 6):
    print(numero)
```

for



break y continue

- break: **rompe** el bucle, aunque no se cumpla la condición de salida.
- continue: deja de ejecutar esta vuelta del bucle, pero **continúa** con la siguiente.

break y else

- Si vamos a romper un **for** o **while** con un break, podemos poner un **else** para cubrir el caso en el que no se ha salido por break.
- Ejemplo: busco números pares en una lista

```
for numero in [23, 3, 67, 12, 55]:
    if (numero % 2 == 0):
        print(f'{numero} es par')
        break
else:
    print('No hay números pares en la lista')
```

Referencias

- John Sturtz, Real Python.
- John M. Zelle. Python Programming: An Introduction to Computer Science.
- Wikipedia.

Referencias

- Imágenes:
  - Wikipedia
  - Chua Hock-Chuan, Yet another insignificant… programming notes.
  - XKCD
  - https://www.flickr.com/photos/ruudhilgeman/6144979641
  - https://www.flickr.com/photos/string_bass_dave/263222263
  - https://www.flickr.com/photos/laenulfean/5943132296
  - https://www.flickr.com/photos/pablo_javier/10127935845