

Paradigmas de Programación

Bucles, métodos, funciones, procedimientos, orientación a objetos, módulos



Cristian Tejedor García
Departamento de Informática
Universidad de Valladolid

Curso 2025-26

Grado en Ingeniería Informática
Grado en Estadística
INDAT



Contenido



1. Objetivos
2. Estructuras de control
3. Funciones
4. Orientación a objetos
5. Reto
6. Compilador online



1. Objetivos

- Resolver dudas de la instalación, teoría y ejercicios de la sesión 1.
- Explicar las estructuras de control.
- Explicar funciones.
- Introducir orientación a objetos.





2. Estructuras de control (I)

- **Secuencia:**

- Elegir **tabuladores** o **espacios**.

- ☐ Y aplicarlo siempre (al menos en el mismo fichero).

- **Dos puntos :** para todas las estructuras.



2. Estructuras de control (II)

- **Alternativa:**

- **Normal:** # No existe switch-case como tal

```
if <cond>:  
    <sentencia>  
elif <cond>:  
    <sentencia>  
else:  
    <sentencia>
```

- **En línea (*ternario*):** <algo> if <cond> else <otro>
En java <cond> ? <algo> : <otro>



2. Estructuras de control (III)

- **Iteración (I):**

- `i = 0` # Ninguna indentación
- `while <cond>:` # Ninguna indentación # condición al principio
- `<do_something()>` # Una indentación
- `i += 1` # Una indentación

- `while True:` # Ninguna indentación, imita el **do-while** de Java
- `<do_something()>` # Una indentación
- `if <cond>:` # Una indentación # condición al final
- `break` # Dos indentaciones, break acaba while

- `lista = ["uno", "dos", "tres"]`
- `for elemento in lista:` # Ninguna indentación
- `print(elemento)` # Una indentación

- ☐ Se itera sobre los elementos (sobre posiciones en **Java** o **C**).
- ☐ Para iterar sobre las posiciones hay que hacer: `for i in range(len(X)):`



2. Estructuras de control (IV)

- **Iteración (II):**

```
for num in range(2, 10):  
    if num % 2 == 0:  
        print("Encontrado número par", num)  
        continue # Salta a la siguiente iteración  
    print("Encontrado número impar", num)
```

➤ **for** <cond>:
 else:
 <sentencia>

while <cond>:
 else:
 <sentencia>

✓ La parte del **else** se ejecuta siempre que acabe el **for** o el **while**, siempre y cuando no haya terminado por un **break**.



2. Estructuras de control (V)

- **Excepciones:**

try:

<sentencias>

except <tipoError>: # ValueError, ZeroDivisionError, NameError, TypeError

print(*"Estoy en la parte de error"*)

finally: # Siempre se ejecuta, da igual si viene de try o de except

print(*"Estoy en la parte finally"*)

Programación defensiva (throw new Exception() en Java)

if <cond>:

raise(Exception,'Defino la excepción')



3. Métodos (I)

- **No hay diferencia entre funciones y procedimientos:**

```
def <nombre>(<params>, <param>=<valor_por_defecto>):  
    """ Explicación de la función """  
    return <expresión> # acaba la función, OBLIGATORIO siempre  
    return None # Importante, si no queremos devolver nada
```

- No se declara el tipo devuelto ni el de los argumentos.
- Podemos devolver tipos complejos (listas, tuplas, ...).
- **Argumentos mutables** se comportan como paso por referencia (afecta al original y fuera del método). *i.e.: listas.*
 - *Mucho cuidado con utilizar el símbolo = dentro del método con valores mutables pasados como parámetro: aunque sean mutables, se crea un nuevo objeto en memoria y fuera del método no cambia si no lo devolvemos con `return`.*
- **Argumentos inmutables:** paso por valor (copia). *i.e.: tuplas.*



3. Métodos (II)

```
def recorrer_parametros_arbitrarios(parametro_fijo, *arbitrarios):
```

```
    print parametro_fijo
```

```
    # Los parámetros arbitrarios se recorren como listas o tuplas:
```

```
    for argumento in arbitrarios:
```

```
        print(argumento)
```

```
    return None
```

```
# Llamada a la función
```

```
recorrer_parametros_arbitrarios('Fixed', 'arbitrario 1', 'arbitrario 2',  
                                'arbitrario 3')
```



4. Orientación a objetos (I)

`class <nombre> (<herencia>):`

"""Explicación de la clase."""

`def __init__(self):` # **Constructor**: dos guiones bajos delante y atrás
self es el propio objeto

`<PadreClase.__init__(self, atts)>` # **super** (en Java) precursor
<inicializar el objeto con los atributos privados>
`return None`

`def metodos(self, args):` # Obligatorio el **self**, siempre ponerlo
`self.__<atributo_privado> += 1`
`return None`

- **Atributos privados**: poner `self.__<att>` y **SOLO** declararlos en el *constructor* y *getters* y *setters* directamente (al estilo **Javascript**).



4. Orientación a objetos (II)

```
## modulo fecha.py
class Fecha():
    def __init__(self):
        self.__dia = 1
        return None

    def getDia(self):
        return self.__dia

    def setDia(self, dia):
        if dia > 0 and dia < 31:
            self.__dia = dia
        else:
            print("Error 0-31")
            return None
```

```
## Importar la clase en otro módulo
import fecha

# Constructor
mi_objeto = fecha.Fecha()
print(mi_objeto.getDia())

# Cambiamos el día
mi_objeto.setDia(4)
print(mi_objeto.getDia())

# Error al cambiar el día
mi_objeto.setDia(40)
```



4. Orientación a objetos (III)

- **Caso de uso:** Fichero = objeto de tipo 'file'.

A. Opción corta, siempre cierra automáticamente

```
with open('file.txt', 'w') as f: # with abre (open) y cierra (close) el fichero como si fuera un finally
    f.write("Hola amigos!")
```

B. Opción larga, se abre y ...

try:

```
f = open('file.txt', 'w')
```

```
f.write("Hola amigos!") # f.flush() # para forzar el guardado en ese instante
```

except:

Nuestras líneas de código para actuar en caso de excepciones

finally:

... haya o no excepción, se cierra

```
f.close()
```



4. Orientación a objetos (IV)

- **Módulo:** entidad que permite una organización y división lógica del código.
 - Cada fichero Python (.py) es un módulo.
 - Para utilizar funciones en otro módulo:
 - `import modulo1` # Se puede utilizar: `import modulo1 as mo`
 - `modulo1.mi_funcion()` # `mo.mi_funcion()`
 - `from time import asctime` # Ahorramos escribir el módulo delante
 - `print asctime()`
- **Paquete:** entidad que organiza módulos relacionados.
 - Estructura lógica de programas: *entidades*, *interfaz*, *controladores*, *ejecución*, *utilidades*...



4. Orientación a objetos (V)

- La manera estándar de empezar nuestros programas en Python es con `__name__ == "__main__":`:

```
def menu():  
    print("Estoy en el menu")  
    return None
```

```
if __name__ == "__main__":  
    print("Estoy en el main")  
    menu()
```

```
# Estoy en el main  
# Estoy en el menu
```

- Se escribe abajo del todo de nuestro código .py, pero **será ejecutado lo primero.**

5. Reto



Disponemos de un **fichero** de texto que contiene la información de la matrícula de alumnos de primer curso. **Cada línea** del fichero representa un par **asignatura-alumno** (la asignatura se representa por una abreviatura de 2,3 o 4 letras, le sigue un espacio y el NIA del alumno): [Enlace de descarga](#)

Crear una **función reto**, que lea el fichero y lo procese de forma que devuelva un diccionario cuyo índice sea una tupla de dos asignaturas y que almacene el número de alumnos que están matriculados simultáneamente de ambas asignaturas.

Aproximadamente: 15 líneas en Python (o menos) 😊 100 líneas en Java ☹️

Ejemplo de uso:

```
dic = reto("reto.txt")  
dic[("PAR","AMAT")] → 113 # Devuelve el número de alumnos matriculados  
simultáneamente en Paradigmas y Ampliación de Matemáticas
```

6. Intérprete *on-line*



Se puede trabajar *on-line* desde:

https://www.tutorialspoint.com/execute_python3_online.php



Para afianzar...

- Ejercicios de la asignatura (sesión 2) ~2horas:
 - <https://www.infor.uva.es/~cvaca/asigs/docpar/sesion2.pdf>
- Reto propuesto sesión 2.
- "Python para todos", Raúl González Duque
 - http://www.utic.edu.py/citil/images/Manuales/Python_para_todos.pdf
 - **Capítulos:** Estructuras de control, Funciones, Orientación a Objetos y Módulos y Paquetes.