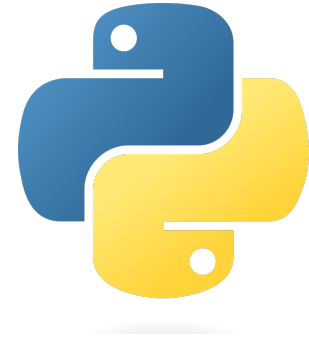


Implementación de metaheurísticas en Python

Día I: Introducción al lenguaje

Jesús Sánchez-Oro Calvo

¿Por qué Python?



Simplicidad

- Código compacto y limpio

IDE

- Pycharm: entorno muy completo para desarrollo y depuración

Multiparadigma

- Orientación a objetos, imperativa, funcional, etc.

Estructuras de datos

- Colección de las estructuras más utilizadas ya implementada

¿Por qué Python?

Prototipado rápido

- Muy similar a pseudocódigo, tiempos de desarrollo reducidos.


Extensiones

- Gran cantidad de módulos que amplían la funcionalidad del lenguaje.

Multiplataforma

- Windows, macOS, Linux, ...

Muy extendido

Oct 2022	Oct 2021	Change	Programming Language		Ratings	Change
1	1			Python	17.08%	+5.81%
2	2			C	15.21%	+4.05%
3	3			Java	12.84%	+2.38%
4	4			C++	9.92%	+2.42%
5	5			C#	4.42%	-0.84%
6	6			Visual Basic	3.95%	-1.29%

Mi primer programa en Python

- ¿Cómo implemento el clásico Hello, World! En Python?



```
print("Hello, World!")
```

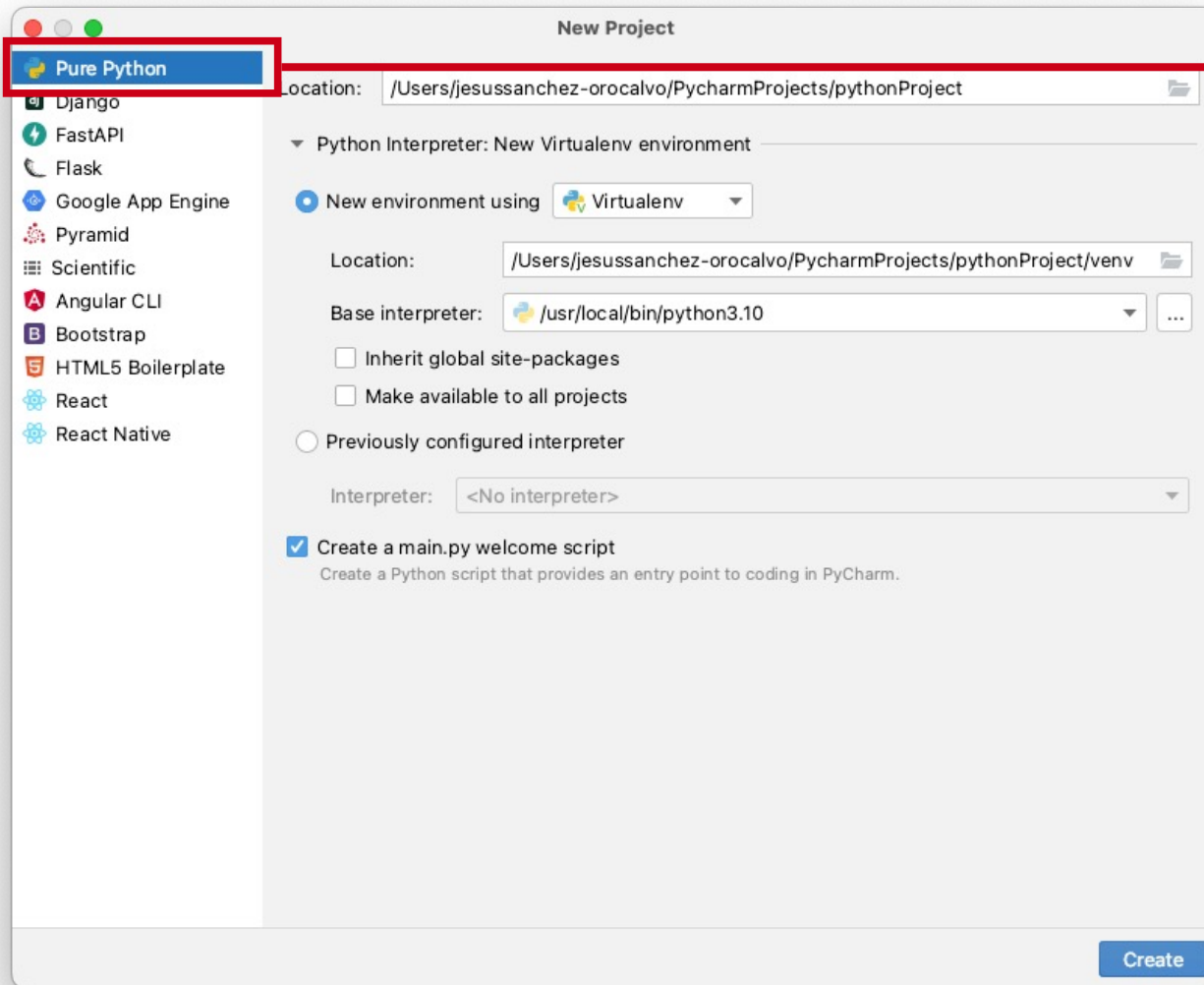
- Nada más, ni incluir librerías externas, ni creación de un main, ni clases, ni nada parecido, solo imprimir por pantalla.

¿Dónde programo?



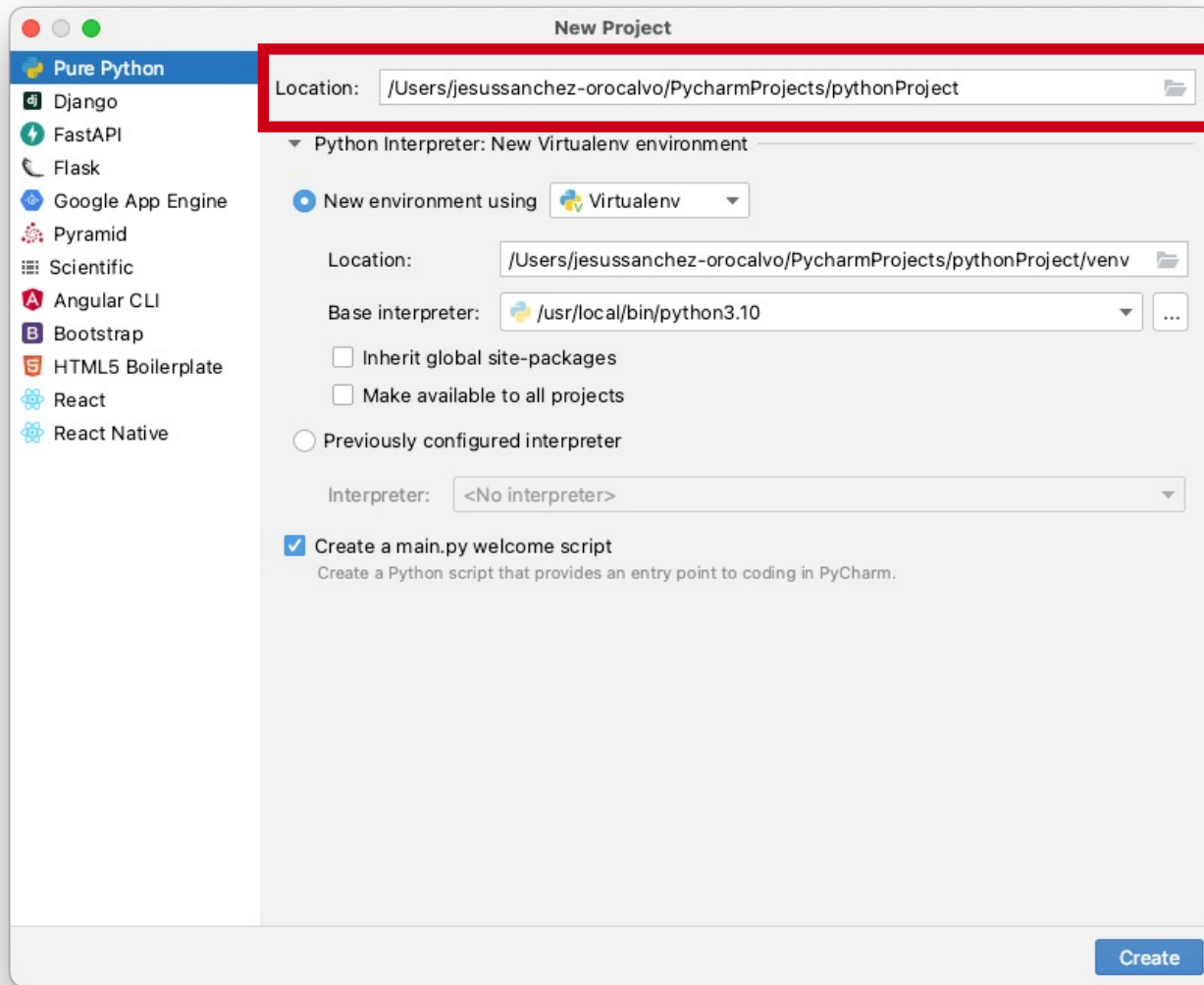
- IDE – Integrated Development Environment
 - Entorno de desarrollo que nos facilita la vida a la hora de programar, depurar, ejecutar, etc.
- PyCharm
 - IDE para Python, versión *community* gratuita para todos, profesional para profesores y estudiantes universitarios.

Creación de un proyecto



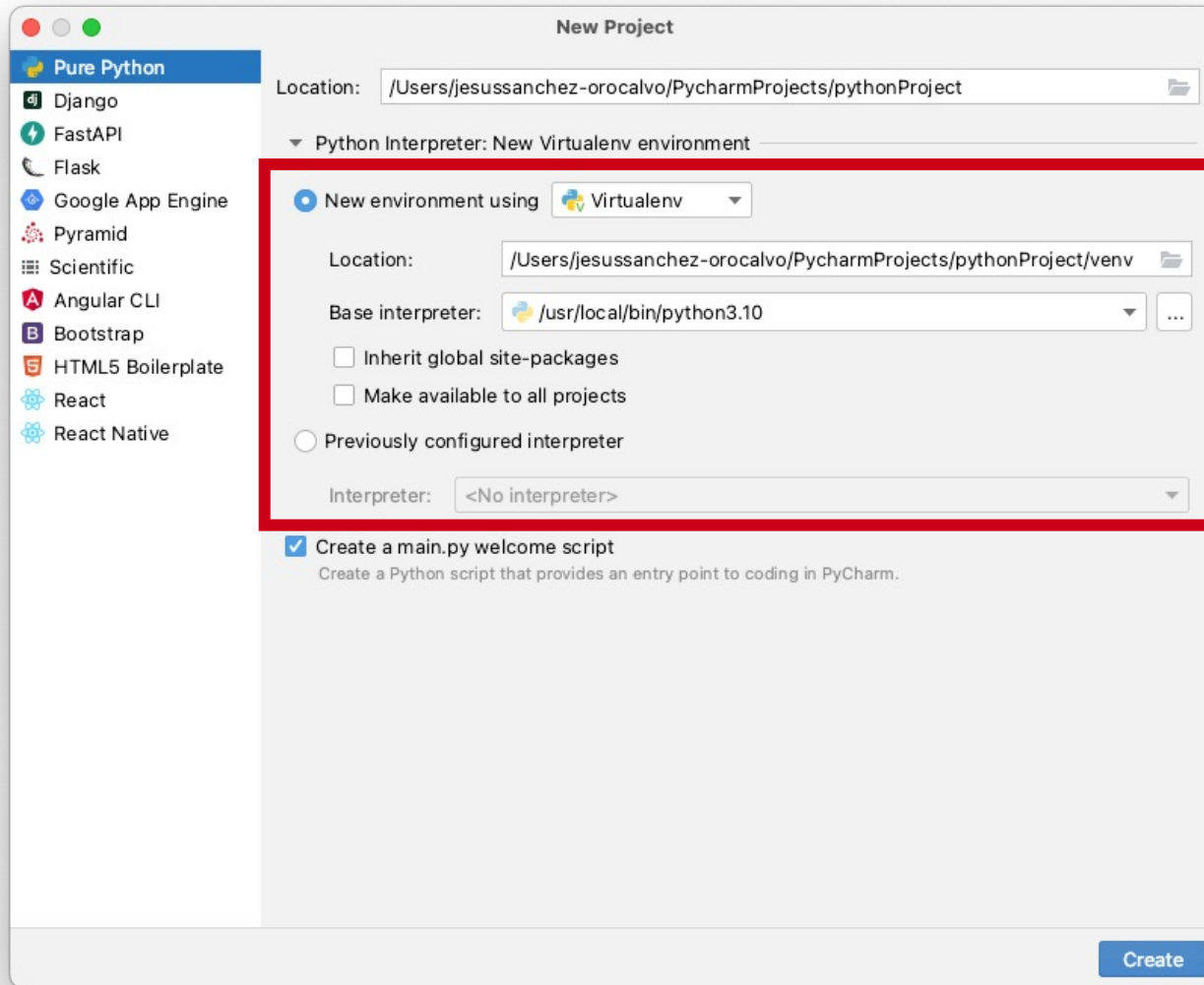
Nos aseguramos de que es un proyecto de Python

Creación de un proyecto



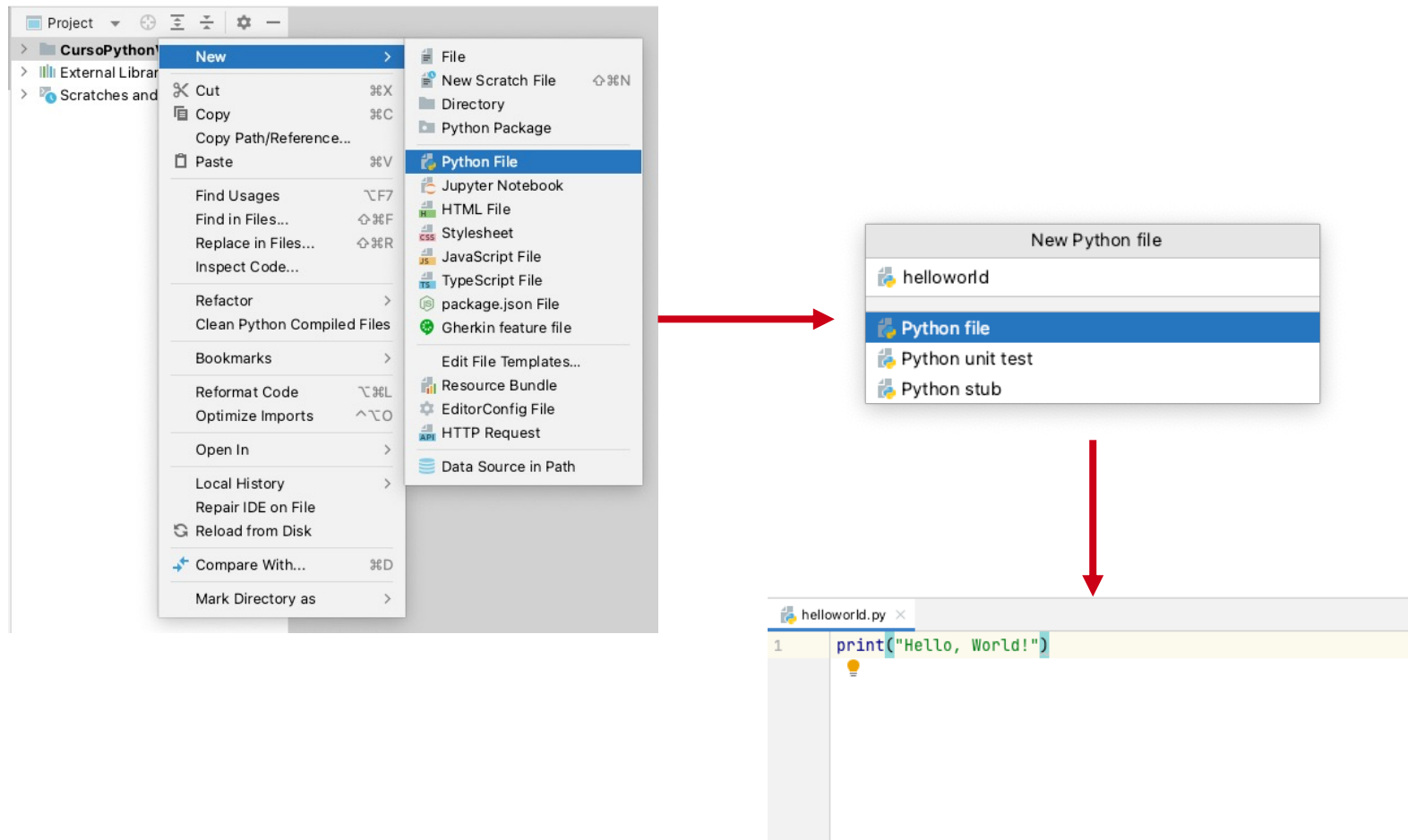
Dónde se va a crear

Creación de un proyecto

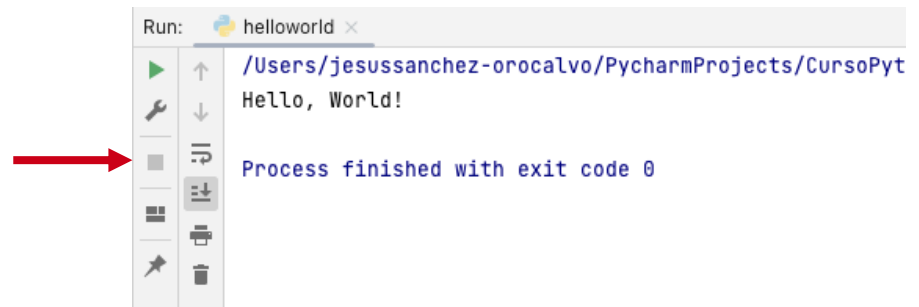
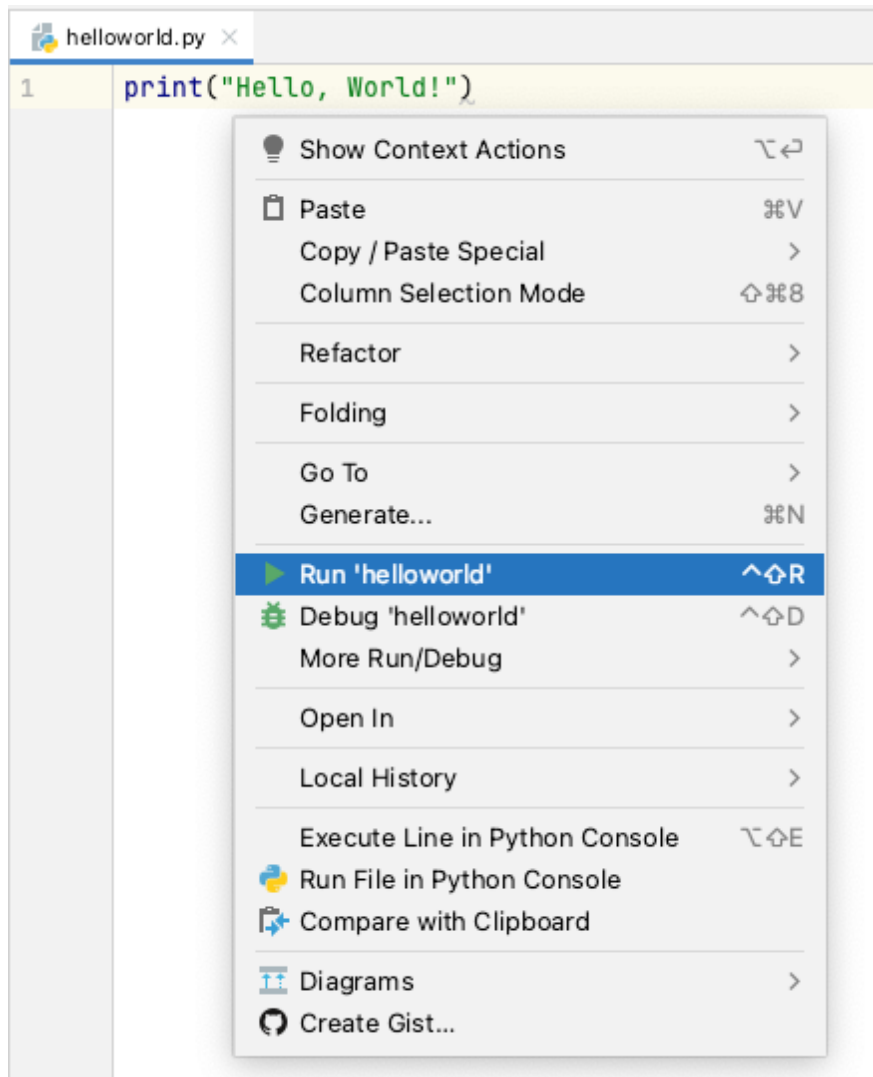


Elección del
intérprete de Python

Creación de un fichero



¿Cómo ejecuto el código?



Operadores y expresiones

Operación	Operador	Aridad	Asociatividad	Preced.
Exponenciación	**	Binario	Por la dcha	1
Identidad, cambio de signo	+, -	Unario	-	2
Multip, div	*, /	Binario	Por la izq	3
Div ent, mód	//, %	Binario	Por la izq	3
Suma, resta	+, -	Binario	Por la izq	4
Distinto, igual que	!=, ==	Binario	-	5
Menor, menor o igual que	<, <=	Binario	-	5
Mayor, mayor o igual que	>, >=	Binario	-	5
Negación	not	Unario	-	6
Conjunción	and	Binario	Por la izq	7
Disyunción	or	Binario	Por la izq	8

Tipos de datos

Enteros

- Ocupan menos en memoria
- Sus operaciones son más rápidas que con números reales

Reales

- Se pueden utilizar diferentes notaciones: $3.2E-3$, $.01$, $2.$, etc.

Booleano

- Pueden tomar valor `True` o `False`
- Operadores `and`, `or` y `not`

Cadenas de caracteres

- Se escriben entre comillas simples o dobles
- Operadores: concatenación (+) y repetición (*)

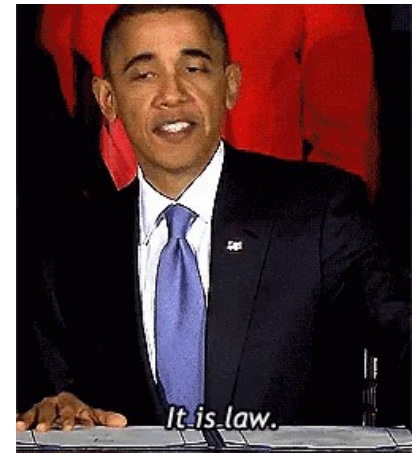
Tipos de datos

- Operadores de relación

operador	comparación
\neq	distinto que
$=$	igual que
$<$	es menor que
\leq	es menor o igual que
$>$	es mayor que
\geq	es mayor o igual que

Identificadores

- Los **identificadores** deben cumplir ciertas normas:
 - Combinación de letras, dígitos y/o el carácter guion bajo ‘_’
 - El primer carácter no puede ser un dígito
 - No puede coincidir con una palabra reservada
 - Case-sensitive: distingue entre mayúsculas y minúsculas



Funciones predefinidas

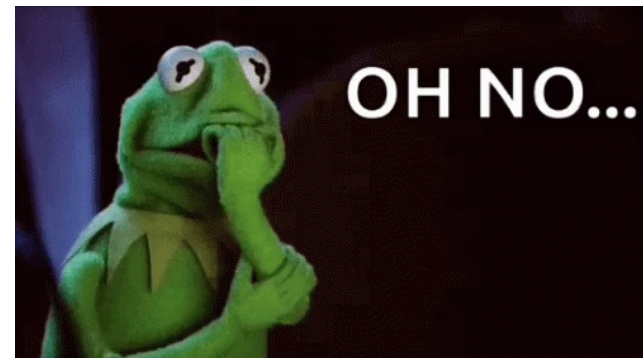
- Python dispone de algunas funciones predefinidas
 - `abs()`: valor absoluto
 - `float()`: devuelve el valor real de un entero o cadena de caracteres
 - `int()`: devuelve el valor entero de un real (trunca) o cadena de caracteres
 - `str()`: devuelve un real o entero en forma de cadena de caracteres
 - `round()`: se puede usar con uno (entero más cercano) o dos argumentos (cuántos decimales queremos conservar).

Módulos

- Los **módulos** son librerías que amplían la funcionalidad del lenguaje
- No están disponibles directamente, tenemos que **importarlos** para usarlos.
 - `from math import sin`: solo importa la función seno
 - `from math import *`: importa todo el módulo math



- **No** se recomienda **importar módulos completos**
 - Si importamos elementos individuales, sabremos su procedencia, favoreciendo **legibilidad** y **mantenibilidad**
 - Si definimos una variable que coincide con el nombre de una función del módulo, podríamos tener **conflictos**



Módulos

- Para evitar problemas, se **recomienda** importarlo de la siguiente manera:

```
import MODULO
```

- El **acceso a las funciones** del módulo se hace a través del nombre del módulo:

```
import math  
print(math.sin(0))
```

Entrada / Salida por consola

- La sentencia `input` recibe **datos por teclado** en formato **cadena de caracteres**
 - Podemos convertirlo al tipo de datos que necesitemos
- La sentencia `print` **imprime** cadenas de caracteres por **consola**.
 - Por defecto, añade un salto de línea al final, pero podemos modificarlo mediante el parámetro `end`

```
name = input("Introduce tu nombre: ")
print("Hola "+name)
age = int(input("Introduce tu edad: "))
print("Tienes "+str(age)+" años")
```

Entrada / Salida por fichero

- La función `open` nos permite **abrir un fichero**, y recibe dos parámetros:
 - `file`: ruta hasta el fichero que queremos abrir
 - `mode`: modo de apertura, 'r' para leer, 'w' para escribir
- Para definir el **entorno** en el que está abierto el fichero, se usa la sentencia `with`
- Todo lo que esté **tabulado** dentro de ella considera que el fichero está abierto, y una vez salimos de ese bloque, el fichero se **cierra**:

```
with open("fichero.txt", "w") as f:
```

Entrada / Salida por fichero

- El **identificador** del fichero será el nombre que indiquemos en la sentencia with anterior
- Para escribir texto, utilizaremos la función write:

```
f.write("Esto es una prueba")
```
- Para leer líneas, utilizaremos la función readline:

```
line = f.readline()
```

Sentencia condicional - if

```
if condicion:  
    instruccion 1  
    instruccion 2  
    ...  
    instruccion N
```

- **Si se cumple la condición**, se ejecuta el bloque de instrucciones delimitado por el tabulado

Sentencia condicional – if-else

```
if condicion:  
    instrucciones  
else:  
    otras_instrucciones
```

- Si se cumple la condición se ejecutan unas instrucciones, y si no se cumple las otras.
- **Evaluación perezosa**
 - Si evaluando la primera parte sabemos el resultado, no se evalúa el resto

Sentencia condicional – if-else-if

```
if condicion:  
    instrucciones  
elif:  
    mas_instrucciones  
else:  
    otras_instrucciones
```

- Si tenemos **más de una condición**, podemos utilizar la sentencia if-elif

Sentencia iterativa – while

```
while condicion:  
    instruccion 1  
    instruccion 2  
    ...  
    instruccion N
```

- Las instrucciones internas se ejecutan **mientras se cumpla** la condición indicada

Sentencia iterativa – for

```
for variable in rango_valores:  
    instruccion 1  
    instruccion 2  
    ...  
    instruccion N
```

- Las instrucciones se ejecutan **por cada valor del rango** de valores indicado.

¿Y si no tenemos un rango de valores?

- La función `range` permite crear un **rango de valores** dado un valor inicial y uno final
 - El inicio se incluye pero el final no
- Se puede utilizar con **varios argumentos**:
 - Con un argumento, se especifica solo el final (el inicio es 0)
 - Con dos argumentos, se especifica inicio y final
 - Con tres argumentos, se especifica inicio, final y el incremento

```
for i in range(10):  
    print(i)
```

```
for i in range(1,10):  
    print(i)
```

```
for i in range(1,10,2):  
    print(i)
```

¿Y si queremos una serie decreciente?

- Podemos establecer un **incremento negativo**
- El rango inicial y final deben **invertirse**

```
for i in range(10,0,-1):  
    print(i)
```

Funciones

```
def nombreFuncion(a, b, c, ...):  
    instruccion 1  
    instruccion 2  
    ...  
    return ...
```

- def es la palabra reservada para definir una **función**
- return indica lo que devuelve la función (**opcional**)

Listas

- En Python podemos definir **listas** de cualquier tipo de datos
- Los valores se indican entre **corchetes** y separados por comas

```
l = [1, 2, 3]  
l = ['a', 'b', 'c']
```

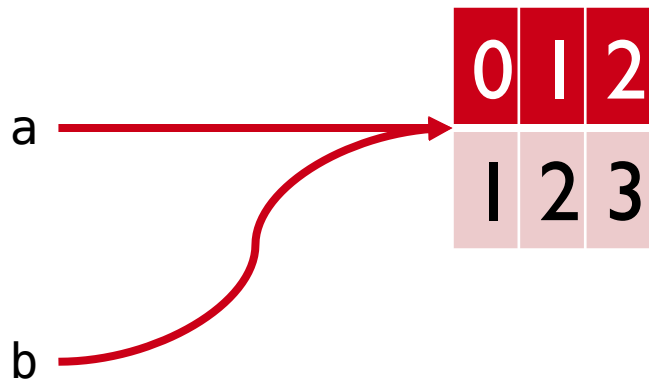
- Podemos crear listas de elementos de **diferentes tipos**

```
l = ['a', 1, 3.14]
```

Listas

- ¡OJO! Las listas en Python son **punteros**

```
a = [1, 2, 3]  
b = a
```



Listas

- Longitud de una lista: `len(lista)`
- Concatenación de listas: operador `+`
- Repetición de listas: operador `*`
- Fragmentos: operador corte, `lista[i:j]`
- Añadir elemento: concatenar una lista o función `append`
- Eliminar posición: operador del `lista[i]`
- Saber si un elemento está en la lista: operador `in`

Matrices

- En Python **no existe** el concepto de **matriz**, utilizamos listas de listas:

```
m = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

- El acceso se realiza de la misma manera:

```
m[0][1]
```

- Creación de una matriz de $n \times n$:

```
matrix = []  
for i in range(n):  
    matrix.append([0] * n)
```

Diccionarios

- Correspondencia entre **pares clave-valor**
- Creación

```
d = {}
```

- Añadir entradas al diccionario:

```
d['uno'] = 1  
d['dos'] = 2
```

Implementación de metaheurísticas en Python

Día I: Introducción al lenguaje

Jesús Sánchez-Oro Calvo
