Diseño e implementación de metaheurísticas con Python

Sergio Pérez Peló

Jesús Sánchez-Oro

Universidad
Rey Juan Carlos

Python

- Lenguaje creado por **Guido van Rossum** a principios de los 90.
- **Interpretado**, no compilado.
- Utilizaremos la versión **Python 3**.

Python

- Su filosofía es que el código sea **legible**.
- Utilizado para desarrollar **cualquier tipo de aplicación**.
  - Instagram o Netflix, entre otros.
- Multiparadigma, pero nos centraremos en la **programación imperativa**.

Estructura de un programa en Python

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

Comentarios

- Permiten **documentar** el programa para facilitar el mantenimiento del código.
- Se **recomienda** su uso:
  - en expresiones que no sean obvias.
  - para comentar aspectos importantes del código.
- Son **ignorados** por el compilador.
- Cualquier línea que **comience** por **#** es un comentario.

Identificadores

- **Nombran** o hacen referencia a datos o a subprogramas.
- Es el nombre con el que se hace referencia a una **dirección de memoria** donde está almacenado aquello que se representa.

Identificadores

- Cada lenguaje tiene sus propias normas para los identificadores.
- En Python, deben ser una **cadena de caracteres** compuesta por:
  - Letras mayúsculas y minúsculas.
  - Dígitos.
  - El carácter '_' (guion bajo).
- **No** pueden aparecer espacios ni otros signos de puntuación.

Identificadores

- El **primer carácter** debe ser una letra
- **Case-sensitive**
- Utiliza identificadores **significativos** para mejorar la legibilidad y el mantenimiento.

Tipos de datos

- En Python existen tres tipos de datos **básicos**:
  - Numéricos
  - Booleanos
  - Cadenas de caracteres

Booleanos

- Puede recibir dos valores: `True` o `False`.
- Las colecciones devuelven `False` si están vacías y True en caso contrario.
- El valor 0 también se considera `False`.
- Operadores lógicos: `and`, `or`, y `not`.
- Operadores de relación: `!=`, `==`, `<`, `<=`, `>`, `>=`.

Numéricos

- Existen tres tipos: enteros, coma flotante y complejos.
- Los enteros ocupan menos memoria y se opera más rápido que con los flotantes.

Numéricos

Operación	Operador	Aridad
Exponenciación	`**`	Binario
Identidad, cambio de signo	`+, -`	Unario
Multiplicación, división	`*, /`	Binario
División entera, módulo	`//, %`	Binario
Suma, resta	`+, -`	Binario
Distinto, igual que	`!=, ==`	Binario
Menor, menor o igual que	`<, <=`	Binario
Mayor, mayor o igual que	`>, >=`	Binario
Negación	`not`	Unario
Conjunción	`and`	Binario
Disyunción	`or`	Binario

Cadenas de caracteres

- Una cadena es una secuencia de caracteres.
- Se expresan entre comillas simples o dobles.
- Operaciones básicas:
  - Concatenación: +
  - Repetición: *

Ejercicio

- Implementa un programa de conversión de euros a bitcoins.
- El cambio está a 28.296,98 € el bitcoin.

Ejercicio

- Implementa un programa que calcule el cuadrado de un número.

Ejercicio

- Implementa un programa que calcule el área de un triángulo dada la base y la altura.

Funciones

- `abs(n)`: calcula el valor absoluto de un número.
- `float(n)`: convierte la entrada en un valor flotante.
- `int(n)`: convierte la entrada en un valor entero.
- `str(n)`: convierte la entrada en una cadena.
- `round(n, d)`: redondea al entero más próximo o con el número de decimales que indiquemos como segundo parámetro.

Módulos

- Python proporciona muchas funciones adicionales: trigonométricas, logaritmos, etc.
- Para utilizarlas, debemos importar el módulo correspondiente.
  - `from modulo import funcion`

Estructuras de selección

- Sentencia condicional `if`

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

- Las acciones se deben escribir con un sangrado mayor que la línea de la condición

Estructuras de selección

- Sentencia condicional `if..else`

```
if condicion:
    instrucciones
else:
    otras
instrucciones
```

- Las acciones se deben escribir con un sangrado mayor que la línea de la condición.

Estructuras de selección

- Sentencia `elif`

```
if condicion:
    instrucciones
elif condicion :
    otras instrucciones
```

- Más compacto y cómodo que escribir varios if-else anidados.

Estructuras de repetición

`while`

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

- Mientras se cumpla una condición, repite las instrucciones internas.

Estructuras de repetición

`for...in`

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

- Para cada elemento de la colección, ejecuta las instrucciones internas

Función range

- A veces escribir todos los valores es incómodo o incluso imposible.

- La función range recibe el inicio y el final del rango, y genera el rango de números entre los dos límites (sin incluir el último).

Función range

- Por ejemplo, `range(1,11)` genera la secuencia de enteros desde el 1 hasta el 10, ambos incluidos.
- Se puede utilizar con 1, 2 o 3 argumentos:
  - Con 1: indicamos el final del rango, comenzando en 0.
  - Con 2: indicamos el inicio y el final del rango.
  - Con 3: indicamos inicio y final, y el incremento:
    - `range(1,10,2)` → `1,3,4,5,9`
    - `range(1,10,3)` → `1,4,7`
    - `range(0,10,3)` → `0,3,6,9`

Función range

- Las series pueden ser decrecientes si indicamos un crecimiento negativo:
  - range(10,1,-1) → 10,9,8,7,6,5,4,3,2
  - range(20,-11,-5) → 20, 15,10,5,0,-5,-10
- En este caso, el inicio debe ser mayor que el fin.

Funciones

- Para definir una función:

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

- def: indica que estamos definiendo una función
- return: indica el valor de retorno de la función

Ámbito

Ámbito local

- Cada función define un ámbito local: su cuerpo.
- Los identificadores de las variables locales solo son visibles dentro de su ámbito.

Ámbito global

- Son las líneas de código que no forman parte de una función.
- Las variables globales son visibles en el ámbito global y en todos los locales.

Módulos

- Las funciones ayudan a que nuestro código sea más legible y evitemos repetir los mismos cálculos.
- Pero terminaremos repitiendo las mismas funciones.
- Para evitarlo, podemos incluir nuestras funciones en módulos y reutilizarlos.

Operaciones con cadenas

- En Python podemos realizar muchas operaciones con cadenas de caracteres:
  - Concatenación: +
  - Repetición: *
  - Paso a minúsculas: `lower()`
  - Paso a mayúsculas: `upper()`
  - Longitud: `len(cadena)`
- Las cadenas empiezan en 0, y podemos acceder a una posición con `cadena[i]`.

Operaciones con cadenas

Operador de corte:

- Permite cortar cadenas
- La sentencia:

```
a = "Python mola"
print(a[2:5])
```

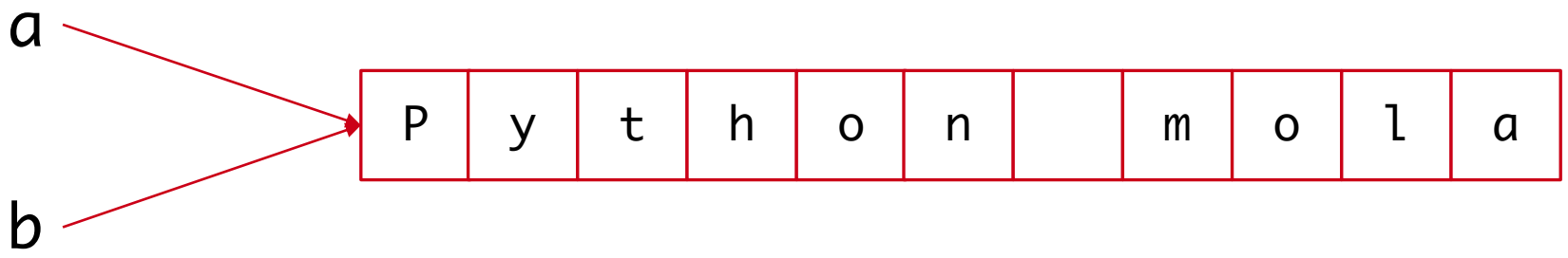
- Produce la salita “tho”

Operaciones con cadenas

- Si hacemos la siguiente asignación:

```
a = "Python mola"
b = a
```

- En memoria tendremos esto:



Operaciones con cadenas

- Las dos variables apuntan a la misma cadena
- Si modificamos una, la otra también se verá modificada

Ejercicio

- Implementa un programa que calcule si un número es par o impar.

Ejercicio

- Implementa un programa que calcule el máximo de tres números.

Ejercicio

- Implementa un programa que calcule la media de una serie de números introducidos por consola.

Ejercicio

- Implementa un programa que imprima la tabla de multiplicar de un número.

Ejercicio

- Implementa un programa que cuente cuántas vocales hay en una frase.

Ejercicio

- Supón que, para un juego, tienes que calcular el desglose mínimo en billetes y monedas de una cantidad exacta de euros.
- Hay billetes de 500, 200, 100, 50, 20, 10 y 5 € y monedas de 2 y 1 €.

Diseño e implementación de metaheurísticas con Python

Sergio Pérez Peló

Jesús Sánchez-Oro

Universidad
Rey Juan Carlos