Programación estructurada

Ricardo Pérez López

IES Doñana, curso 2019/2020

Índice general

1. Funciones definidas por el usuario **1**
1.1. Definición de funciones con nombre . . . 2
1.2. Paso de argumentos . . . 2
1.3. La sentencia `return` . . . 3
1.4. Ámbito de variables . . . 4
1.4.1. Variables globales . . . 4
1.4.2. Variables locales . . . 4
1.5. Declaraciones de tipos . . . 4
1.5.1. Declaraciones de tipo de argumento . . . 4
1.5.2. Declaraciones de tipo de devolución . . . 4
1.6. Funciones locales a funciones . . . 4
1.6.1. `nonlocal` . . . 4
1.7. *Docstrings* . . . 4

2. Teorema de Böhm-Jacopini **4**

3. Estructuras básicas de control **4**
3.1. Concepto de estructura . . . 4
3.2. Secuencia . . . 4
3.3. Selección . . . 4
3.4. Iteración . . . 4

4. Metodología de la programación estructurada **4**
4.1. Recursos abstractos . . . 5
4.2. Diseño descendente . . . 5
4.3. Refinamiento sucesivo . . . 5

5. Captura de excepciones **5**

1. Funciones definidas por el usuario

1.1. Definición de funciones con nombre

- En programación imperativa también podemos definir funciones.
- Al igual que ocurre en programación funcional, una función en programación imperativa es una construcción sintáctica que acepta argumentos y produce un resultado.
- Pero a diferencia de lo que ocurre en programación funcional, una función en programación imperativa es una **secuencia de sentencias**.
- Las funciones en programación imperativa conforman los bloques básicos que nos permiten **descomponer un programa en partes** que se combinan entre sí.
- Todavía podemos construir funciones mediante expresiones lambda, pero Python nos proporciona otro mecanismo para definir funciones en estilo imperativo: las **funciones con nombre**.
- La sintaxis para definir una función con nombre es:

```
def <nombre>([<parámetros>]):
    <sentencia>+
```

- Por ejemplo:

```
def saluda(persona):
    print('Hola', persona)
    print('Encantado de saludarte')

def despide():
    print('Hasta luego, Lucas')
```

- Notas importantes:
  - Tiene que haber, al menos, *una* sentencia.
  - Las sentencias van **indentadas** (o *sangradas*) dentro de la definición de la función, con el mismo nivel de indentación.
  - El final de la función se deduce al encontrarse una sentencia con un **nivel de indentación superior** (en el caso de arriba, otro `def`).

Conclusión:

En Python, la **estructura** del programa viene definida por la **indentación** del código.

1.2. Paso de argumentos

- Existen distintos mecanismos de paso de argumentos, dependiendo del lenguaje de programación utilizado.
- Los más conocidos son los llamados **paso de argumentos *por valor*** y **paso de argumentos *por referencia***.
- En Python existe un único mecanismo de paso de argumentos llamado **paso de argumentos *por asignación*** o también, a veces, **paso de argumentos *por nombre***.

- En la práctica resulta bastante sencillo.
- Consiste en suponer que **el argumento *se asigna* al parámetro** correspondiente, con toda la semántica relacionada con los *alias* de variables, inmutabilidad, mutabilidad, etcétera.
- Por ejemplo:

```
1 def saluda(persona):
2     print('Hola', persona)
3     print('Encantado de saludarte')
4
5 saluda('Manolo')  # Saluda a Manolo
6 x = 'Juan'
7 saluda(x)         # Saluda a Juan
```

- En la línea 5 se asigna a `persona` el valor `Manolo` (como si se hiciera `persona = Manolo`).
- En la línea 7 se asigna a `persona` el valor de `x`, como si se hiciera `persona = x`, lo que sabemos que crea un *alias* (que no afectaría ya que el valor pasado es una cadena, y por tanto inmutable).
- En caso de pasar un argumento mutable:

```
1 def cambia(lista):
2     print(lista)
3     lista.append(99)
4
5 lista = [1, 2, 3]
6 cambia(lista)     # Imprime [1, 2, 3]
7 print(lista)      # Imprime [1, 2, 3, 99]
```

1.3. La sentencia `return`

- Para devolver el resultado de la función al código que la llamó, hay que usar una sentencia `return`.
- Cuando el intérprete encuentra una sentencia `return` dentro de una función:
  - se finaliza la ejecución de la función,
  - se devuelve el control al punto del programa en el que se llamó a la función y
  - se declara que la función devuelve como resultado el valor de retorno definido en la sentencia `return`.
- Por ejemplo:

```
1 def suma(x, y):
2     return x + y
3
4 a = input('Introduce el primer número: ')
5 b = input('Introduce el segundo número: ')
6 resultado = suma(a, b)
7 print('El resultado es:', resultado)
```

- La función se define en las líneas 1–2. El intérprete lee la definición de la función pero no ejecuta sus sentencias en ese momento (lo hará cuando se *llame* a la función).

- En la línea 6 se llama a la función `suma`

1.4. Ámbito de variables

1.4.1. Variables globales

1.4.1.1. `global`

1.4.1.2. Efectos laterales

1.4.2. Variables locales

1.5. Declaraciones de tipos

1.5.1. Declaraciones de tipo de argumento

1.5.2. Declaraciones de tipo de devolución

1.6. Funciones locales a funciones

1.6.1. `nonlocal`

1.7. *Docstrings*

2. Teorema de Böhm-Jacopini

3. Estructuras básicas de control

3.1. Concepto de estructura

3.2. Secuencia

3.3. Selección

3.4. Iteración

4. Metodología de la programación estructurada

4.1. Recursos abstractos

4.2. Diseño descendente

4.3. Refinamiento sucesivo

5. Captura de excepciones