Paradigmas de Programación

Sesión 2 Laboratorio

Cristian Tejedor García
Departamento de Informática
Universidad de Valladolid

Curso 2016-17

Grado en Ingeniería Informática
INDAT

Contenido

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

1. Objetivos

- Resolver dudas 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)**:
  - **while** True: # Ninguna indentación, bucle genérico
    <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, no sobre las posiciones como en Java o C.
    - Para iterara sobre posiciones: for i in range(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>

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. Funciones (I)

- No hay diferencia entre funciones y procedimientos:

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

- No se declara el tipo devuelto.
- Podemos devolver colecciones.
- **Argumentos mutables** se comportan como paso por referencia (afecta al original).
- **Argumentos inmutables**: paso por valor (copia).

3. Funciones (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)
    <inicializar el objeto con los attributos>

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

- **Atributos privados**: poner *self.__<att>* y no declararlo en la clase si no en los *getters* y *setters* directamente (al estilo Javascript).

4. Orientación a objetos (II)


```
class Fecha():
    def __init__(self):
        self.__dia = 1
    def getDia(self):
        return self.__dia
    def setDia(self, dia):
        if dia > 0 and dia < 31:
            self.__dia = dia
        else:
            print "Error 0-31“
```

```
# Constructor
mi_objeto = 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’.
  - f = open(“archivo.txt”, “w”)
  - completo = f.read() # Cadena con todo el contenido de f
  - Y las funciones que ya sabemos sobre ficheros…

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*, *controladores*, *interfaz*, *ejecución*, *utilidades*…

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): https://goo.gl/LgfLbG

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: 30 líneas en Python ☺ 100 líneas en Java ☹

Ejemplo de uso:

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

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://mundogeek.net/tutorial-python/
  - **Capítulos**: Estructuras de control, Funciones y Orientación a Objetos.