Ingeniería Informática en Sistemas de Información
Introducción al lenguaje de programación Python

Objetivos

El objetivo de este documento es introducir los conceptos básicos de programación en lenguaje Python (v3), centrándonos en el paradigma de programación estructurada.

Manuales interesantes online:

https://www.iaa.csic.es/python/curso-python-para-principiantes.pdf
http://docs.python.org.ar/tutorial/pdfs/TutorialPython3.pdf

Parte 1: Conceptos básicos

Las características del lenguaje de programación Python se resumen a continuación:

- Es un **lenguaje interpretado**, no compilado, usa **tipado dinámico, fuertemente tipado**.
- Es **multiplataforma**, lo cual es ventajoso para hacer ejecutable su código fuente entre varios sistema operativos (Windows, Linux y macOS).
- Es un lenguaje de programación **multiparadigma**, el cual soporta varios paradigmas de programación como **orientación a objetos, estructurada**, programación imperativa y, en menor medida, programación funcional.
- En Python, el formato del código (p. ej., la indentación) es estructural. **Esta indentación es obligatoria, pues define la estructura del programa.**
- Posee una biblioteca con numerosos módulos para realizar tareas como:
  - Programación web
  - Cómputos ciéntificos (numpy) y gráficas (Matplotlib).
  - Gestión de sockets TCP/IP (socket).
  - Redes convolucionales para aprendizaje automático (keras).
  - Desarrollo web (Django, Piramid, Flask)
  - Interfaces gráficas (pySimpleGUI) (https://github.com/PySimpleGUI/PySimpleGUI)
  - Robótica (ROS)

Parte 2: Tipos de datos básicos y compuestos

Nombre	**Ejemplo**
string	cadena = “Hola mundo”
int	edad = 35
float	Precio = 178.5
bool	verdadero = True falso = False

Tipos compuestos:

	Tuplas	Listas	Diccionarios
Descripción:	Agrupan valores no modificables. Tamaño fijo.	Agrupan elementos modificables. Tamaño variable (con append).	Almacenan pares clave/valor. Pueden ser modificables. Tamaño variable.
Declaración:	mi_tupla = ('cadena', 2.8, 15, 'd')	mi_lista = ['cad', 15, 2.8, 'dato']	mi_dic = {'clave_1': 22, 'clave_2': 15}
Uso:	print(mi_tupla[2:]) #Sale:(15, 'd')	print (mi_lista[-1]) # Sale: 'dato'	print (mi_dic['clave_2']) # Sale: 15

Para ver el número de elementos de un tipo compuesto usamos el operador *len*.

```
mi_tupla = ('cadena', 15, 2.8, 'dato')
print (len(mi_tupla))  # Sale: 4
```

Por último, podemos desempaquetar elementos de una tupla para poder procesarlos por separado. Esto es de especial utilidad cuando una función devuelve más de un parámetro.

```
mi_tupla = ('cadena', 15)
a, b = mi_tupla
print (a)    ; Sale:  "cadena"
```

Parte 3: Estructuras de control de flujo

En Python las estructuras de flujo están delimitadas según la indentación del código. Al usar una de ellas, las instrucciones que se ejecuten dentro de la misma estarán en una indentación superior a la actual. Normalmente se usan indentaciones de 4 espacios para delimitar la parte que está dentro de la estructura.

3.1 Estructura condicional

Indica un bloque de instrucciones que solo se ejecutará si se cumple una determinada condición.

Ejemplo:

```
if compra <= 100:
     print ("Pago en efectivo")
elif compra > 100 and compra < 300:
     print ("Pago con tarjeta de débito")
else:
     print ("Pago con tarjeta de crédito")
```

3.2 Estructura mientras

Indica un bloque de instrucciones que se repetirá mientras se cumpla una determinada condición.

Ejemplo:

```
num = 1
while num <= 10:
   print ("Num = ",num)
   num += 1   # Ojo! En Python no existe incremento
```

3.3 Bucle for

Itera todos los elementos de una lista o tupla.

Ejemplo:

```
mi_tupla = ('rosa', 'verde', 'celeste', 'amarillo')
for color in mi_tupla:
     print color
```

Para hacer un bucle for que itere una variable en un rango, usamos la instrucción *range*.

Ejemplo:

```
for num in range(1,10):
   print ("Num = ",num)
```

Parte 4: Definiendo funciones

En programación estructurada las funciones se usan para facilitar la reutilización de código. Las funciones aceptan parámetros y pueden devolver un valor, o varios agrupados en tuplas o listas (tipos compuestos).

Ejemplo:

```
def saludo(nombre):     # aquí el algoritmo
   return 'Hola ' + nombre

print (saludo('Juan'))  # Devuelve 'Hola Juan'
```

Parte 5: Introducción a la programación orientada a objetos con Python

Con Python podemos definir clases que contarán con sus atributos y métodos, y podrán ser instanciadas. Para ello usamos el comando class. Existe una palabra clave para indicar el contenido de una clase, dicho comando es self. Todos los métodos de la clase deben incluir dicha palabra clave como primer argumento. Ejemplo:

```
class Persona:
       def __init__(self, nombre, edad):
              self.nombre = nombre
              self.edad = edad    # Creamos dos atributos en self: nombre y edad

       def toString(self):
              return "Me llamo "+self.nombre+" y tengo "+ str(self.edad)+" años"

persona = Persona ("Juan", 26)   # persona es una instancia de la clase Persona
print (persona.toString())
```

La mayoría de las funciones que se proporcionan en los módulos de python se implementan como clases y tendremos que instanciarlas y llamar a sus métodos para usarlas.

Parte 6: Trabajando con ficheros

Para trabajar con ficheros, importaremos el módulo file. Además, con la directiva *with*, podemos crear un objeto fichero temporal que estará activo en un determinado bloque de código y se borrará de forma limpia al finalizar dicho bloque. Ejemplo:

```
with open('my_fichero.txt', 'r') as f:
text = f.readline()
print (text)
```

En este caso hemos abierto el fichero "mi_fichero.txt' en modo lectura. De forma similar, podremos abrir en modo escritura o de anexado (append) el archivo, e incluso en modo binario.

```
with open('prueba.txt', 'w') as f:
text = f.write('Esto es una prueba')
```

Parte 7: Importando funcionalidades nuevas

En Python existe un vasto número de módulos para realizar funcionalidades muy diversas entre sí. Para usar una de ellas usamos la directiva *import.* Por ejemplo usaremos el módulo sys para acceder a los argumentos de línea de comandos y el módulo *argparse* para poder interpretarlos de forma sencilla.

7.1 Módulo pip

Python tiene incorporado un módulo para instalar nuevos paquetes, llamado *pip*. Si no está instalado, para instalarlo deberemos:

1. Descargar https://bootstrap.pypa.io/get-pip.py
2. Ejecutarlo con python. Para ello, en una línea de comandos ejecutaremos:
   `python get-pip.py`
3. Comprobamos que está correctamente instalado. Para ello, en la línea de comandos:

```
pip -V
```

Una vez instalado, podremos agregar nuevos módulos dentro de la línea de comandos de python.

```
pip install pySimpleGUI
```

Parte 8: Gestión de excepciones

En contraste con la gestión de errores de los lenguajes de programación de bajo/medio nivel (sobre todo C), muchos lenguajes de programación de alto nivel suelen usar excepciones para indicar que una función no ha podido completarse de forma normal. Un ejemplo sería la imposibilidad de abrir un fichero para su lectura.

Si dichas excepciones no son capturadas, terminarían la ejecución del programa en el momento en el que se produzcan mostrando un mensaje en la salida de errores del sistema. Para evitar esto, las excepciones pueden capturarse usando la secuencia de instrucciones *try* y *except.*

Por ejemplo, el siguiente código se asegura de que el usuario introduzca un tipo entero de datos.

```
while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try again...")
```