Programación en Python

Programación como herramienta para la ingeniería

Tomado casi en su totalidad de material desarrollado por Ivania Donoso y Antonio Ossa

Que versión de Python vamos a usar

- (Recomendación) Descargar desde https://www.continuum.io/downloads la última versión de la plataforma Anaconda (versión 3.6 de Python).
- Contiene múltiples herramientas que utilizaremos (Jupyter, R, etc.).
- (Otra opción) Descargar Python desde https://www.python.org/downloads

PEP 8

Guía de estilo

PEP8

- Python Enhancement Proposal 8 es la guía de estilo de Python
- Se usa para hacer más legible y consistente el código
- https://www.python.org/dev/peps/pep-0008/

PEP8

- Imports al comienzo del módulo
- Nombres de variables descriptivos
- Espacios entre líneas
  - 2 líneas después de los imports
  - 2 líneas alrededor de las clases y funciones
  - 1 línea entre métodos de clase
  - 1 espacio después de “,” y a cada lado de los operadores
- Líneas de máximo 80 caracteres (incluyendo espacios)
- NO usar tabs. Solo usar espacios.

CamelCase y snake_case

```
CONST_PI = 3.1415
class ClaseDeEjemplo:

   def __init__(self, hola):
      self.variable_de_ejemplo = hola

   def metodo_de_ejemplo(self):
      return 1 + 1 == 2
```

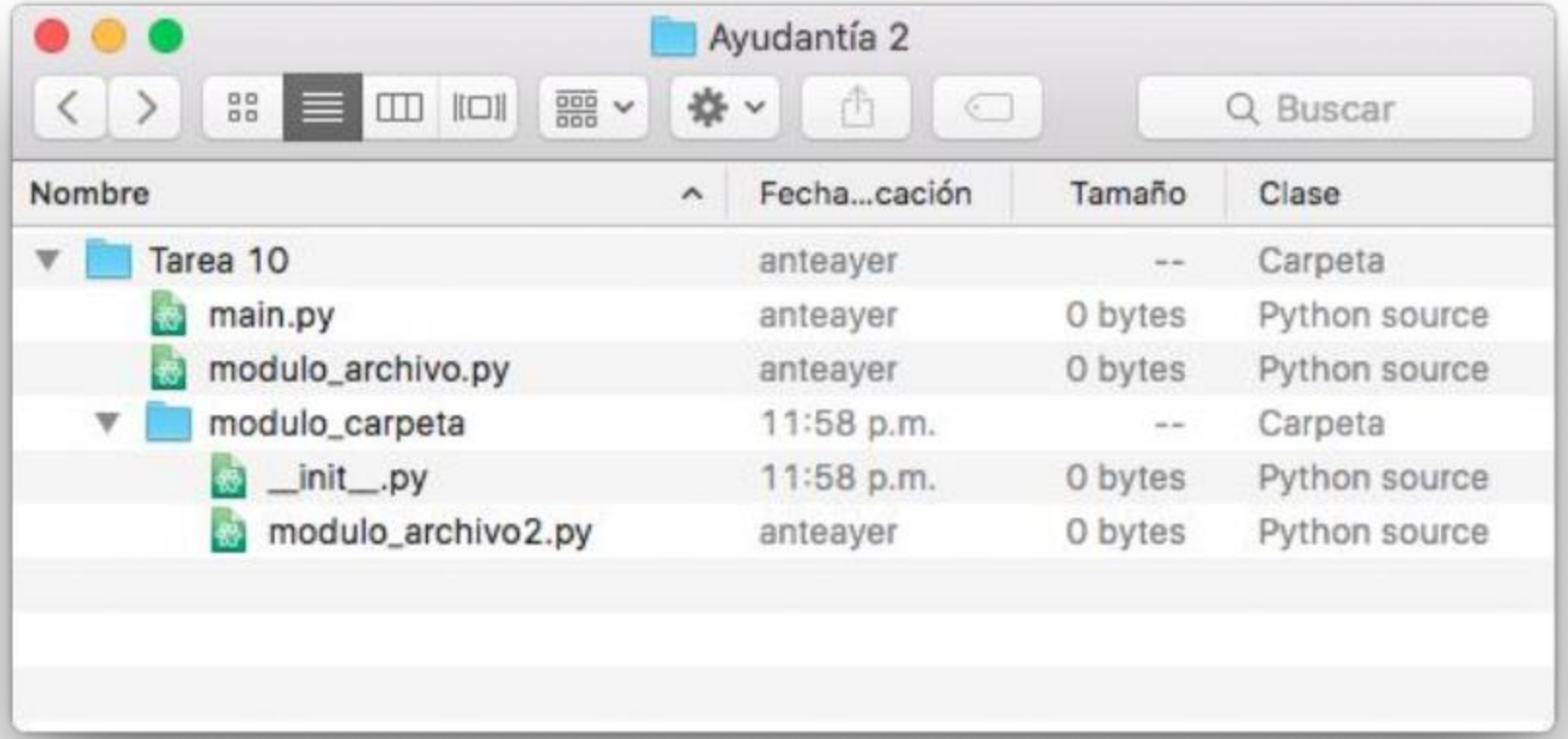
Siempre recuerda que el código se lee más veces de lo que se escribe y que otro lo va a leer.

Modularización

Modularización: ¿Por qué?

- Cuando un programa crece, se hace inviable mantenerlo en un solo archivo:
  - El mantenimiento es difícil
  - El trabajo en equipo es difícil
  - Es desordenado
- Como es un archivo de python normal puede tener:
  - Variables
  - Métodos
  - Clases

Modularización



Cómo usar módulos

Importándolo entero

```
import modulo_archivo


if __name__ == “__main__”:
    variable_tipica = modulo_archivo.VALOR_FIJO
    objeto_tipico = modulo_archivo.Clase()
    modulo_archivo.funcion()
```

Cómo usar módulos

Importándolo entero con pseudónimo

```
import modulo_archivo as ma


if __name__ == “__main__”:
    variable_tipica = ma.VALOR_FIJO
    objeto_tipico = ma.Clase()
    ma.funcion()
```

Cómo usar módulos

Importando lo necesario

```
from modulo_archivo import VALOR_FIJO, Clase, funcion


if __name__ == “__main__”:
    variable_tipica = VALOR_FIJO
    objeto_tipico = Clase()
    funcion()
```

Cómo usar módulos

Importando un package

Un package es una carpeta que tiene tener el archivo vacío “__init__.py”

```
import modulo_carpeta as mc


if __name__ == “__main__”:
    variable_tipica = mc.modulo_archivo2.VALOR_FIJO
    objeto_tipico = mc.modulo_archivo2.Clase()
    mc.modulo_archivo2.funcion()
```

Cómo usar módulos

- Cuando se importa un módulo se ejecuta todo el código en él
- Para evitar que se ejecute código de un módulo al ser importado se utiliza el siguiente if:

```
# Código del módulo

if __name__ == “__main__”:

    # Mucho código escrito
```

__name__?

bar.py

```
import foo

# Código

if __name__ == “__main__”:
    # Código
```

foo.py

```
# Código
def method():
    pass
```

Cómo NO usar módulos

Importando todo sin referencia al módulo

```
from modulo_archivo import *

if __name__ == “__main__”:
    variable_tipica = VALOR_FIJO
    objeto_tipico = Clase()
    funcion()
```

Cómo NO usar módulos

- Evita crear módulos que se llamen igual a los que vienen incluidos en python
- ¿Cómo busca los módulos python?:
  - Módulo de la librería estándar
  - Módulo en la misma carpeta
  - Módulo en el directorio de instalación