
python™

Contenido

- Importaciones Lib/modulos
- Instalar Librerias (PIP)
- PEP-8
- Clases y Objetos
- Comentarios
- Interpretación de errores
- Debug de código
- Lectura y Escritura de archivos

Importaciones Lib/modulos

```
import datetime.datetime
```

```
from datetime import datetime, date
```

```
from .calculadora import suma,
resta
```

calculadora.py

```
def suma(x: int, y: int) -> int:
    return x + y

def resta(x: int, y: int) -> int:
    return x - y
```

Instalar Librerias (PIP)

```
$ pip install <paquete>==<version>
```

```
$ pip install <paquete_1> <paquete_2>
```

```
$ pip install -r requirements.txt
```

requirements.txt

```
# Comentario

flask>=3.2
django==3
pewee
```

PEPs

- Son documentos que contiene las propuestas de mejora de Python, conocidas como PEP.
- Los editores de PEP asignan los números de PEP y una vez asignados, nunca se cambian.

Link: https://www.python.org/dev/peps/

PEP 8 - Introducción

Este documento proporciona las convenciones de codificación para el código Python que comprende la biblioteca estándar en la distribución principal de Python.

PEP 8 - Herramientas

En este curso principalmente vamos a usar flake8 para verificar el estado del código.

Instalación:

```
$ pip install flake8
```

```
$ pip install pylama
```

Ejecución:

```
$ flake8 <archivo>.py
```

```
$ pylama <archivo>.py
```

PEP 8 - Disposicion del codigo - Indentación

Use 4 espacios por nivel de indentación.

SI

```
# Aligned with opening delimiter.
foo = long_fun_name(var_one, var_two,
                    var_three,
var_four)

# Add 4 spaces (an extra level of
indentation) to distinguish arguments
from the rest.
def long_fun_name(
        var_one, var_two, var_three,
        var_four):
    print(lista)
```

NO

```
# Arguments on first line forbidden
when not using vertical alignment.
foo = long_fun_name(var_one, var_two,
    var_three, var_four)

# Further indentation required as
indentation is not distinguishable.
def long_fun_name(
    var_one, var_two, var_three,
    var_four):
    print(lista)
```

PEP 8 - Disposicion del codigo - Espacios

Tabs o espacios?: Siempre debe usarse espacios, si bien python permite usar tabs, pero es preferible siempre usar espacios

Largo máximo de las líneas: Se deben limitar el largo total de las líneas a 79 caracteres.

Líneas en blanco: Se deben dejar 2 líneas en blanco por arriba de la definición de una clase. Los métodos dentro de una clase llevan una sola línea en blanco.

PEP 8 - Disposicion del codigo - Espacios

Tabs o espacios?: Siempre debe usarse espacios, si bien python permite usar tabs, pero es preferible siempre usar espacios

Largo máximo de las líneas: Se deben limitar el largo total de las líneas a 79 caracteres.

Líneas en blanco: Se deben dejar 2 líneas en blanco por arriba de la definición de una clase. Los métodos dentro de una clase llevan una sola línea en blanco.

PEP 8 - Nombrado

Los nombres que son visibles para el usuario deben seguir las convenciones que reflejan el uso en lugar de la implementación.

Módulos: Deben tener nombres cortos en minúscula.

Clases: Usan la convención de CapWords.

Funciones y Variables: Se escriben en minúsculas y las palabras se separan por guiones bajos.

Constantes: Se escriben en mayúscula separadas por guiones bajos.

PEP 8 - Disposicion del codigo - Importaciones

Las importaciones siempre se colocan en la parte superior del archivo, justo después de los comentarios y las cadenas de documentación del módulo, y antes de las globales y constantes del módulo.

Las importaciones deben agruparse en el siguiente orden, separados por una línea en blanco:

1. Biblioteca estándar.
2. Importaciones de terceros relacionados.
3. Importaciones específicas de aplicaciones / bibliotecas locales.

PEP 8 - Espacios en Blanco

SI	NO
`spam(ham[1], {eggs: 2})`	`spam( ham[ 1 ], { eggs: 2 } )`
`foo = (0,)`	`bar = (0, )`
`if x == 4: print x, y; x, y = y, x`	`if x == 4 : print x , y ; x , y = y , x`
`ham[1:9], ham[1:9:3], ham[:9:3], ham[1::3], ham[1:9:]`	`ham[1: 9], ham[1 :9], ham[1:9 :3]`
`spam(1)`	`spam (1)`
`dct['key'] = lst[index]`	`dct ['key'] = lst [index]`

`x = 1`
`y = 2`
`long_variable = 3`
	`x             = 1`
	`y             = 2`
	`long_variable = 3`

`i = i + 1`
`submitted += 1`
`x = x*2 - 1`
`hypot2 = x*x + y*y`
`c = (a+b) * (a-b)`
	`i=i+1`
	`submitted +=1`
	`x = x * 2 - 1`
	`hypot2 = x * x + y * y`
	`c = (a + b) * (a - b)`

Clases

Las clases proveen una forma de empaquetar datos y funcionalidad juntos. Al crear una nueva clase, se crea un nuevo tipo de objeto, permitiendo crear nuevas instancias de ese tipo. Cada instancia de clase puede tener atributos adjuntos para mantener su estado. Las instancias de clase también pueden tener métodos para modificar su estado.

Clases - Definición y uso

Definición de una clase

```
class Rectangulos(object):
    def __init__(self, alto, ancho):
        self.alto = alto
         self.ancho = ancho
```

Uso

```
rectangulo = Rectangulos(20, 30)
```

Clases - Métodos y atributos

Definición

```
class Rectangulos(object):
    self.alto = 0
    self.ancho = 0
    def __init__(self, alto, ancho):
        self.alto = alto
        self.ancho = ancho

    def area(self):
        return self.alto * self.ancho

    def perimetro(self):
         return 2 * self.alto + 2 * self.ancho
```

Uso

```
rectangulo = Rectangulos(20, 30)
area = rectangulo.area()
```

Clases - Herencia

Definición

```
class Cuadrado(Rectangulos):
    def __init__(self, alto):
        super().__init__(alto, alto)
```

Uso

```
cuadrado = Cuadrado(20)
area = cuadrado.area()
```

Comment

Definición

```
# Este es un comentario de 1 linea
# Este es otro comentario de 1 linea

class Cuadrado(Rectangulos):

    # Comentario.
    def __init__(self, alto):
        super().__init__(alto, alto)
```

Uso

```
"""
Este es un comentario
multi-lineas.
"""
```

Pep-8: https://www.python.org/dev/peps/pep-0008/#id28

Interpretación de errores

En el momento de programar la interpretación de los errores es muy importante, ahora llega el punto de empezar a interpretar estos errores.

```python
class Rectangulos(object):
    def __init__(self, alto, ancho):
        self.alto = alto
        self.ancho = ancho

    def area(self):
        return self.alto * self.anchos

    def perimetro(self):
        return 2 * self.alto + 2 * self.ancho


class Cuadrado(Rectangulos):
    def __init__(self, alto):
        super().__init__(alto, alto)


if __name__ == '__main__':
    print("="*30)
    print("Cuadrado")
    cuadrado = Cuadrado(20)
    print("Area: {0}".format(cuadrado.area()))
    print("Perimetro: {0}".format(cuadrado.perimetro()))
```

```
AR0FVFYH14SHV2J:dataset pdalmasso$ python3 test.py
==============================
Cuadrado
Traceback (most recent call last):
  File "test.py", line 21, in <module>
    print("Area: {0}".format(cuadrado.area()))
  File "test.py", line 7, in area
    return self.alto * self.anchos
AttributeError: 'Cuadrado' object has no attribute 'anchos'
AR0FVFYH14SHV2J:dataset pdalmasso$
```

Debug de código

Para realizar una corrida interactiva en nuestros codigo muchas veces caemos en el uso de print, imprimir variables. para evitar esto tenemos la existencia de pdb, ipdb, etc. otras herramientas.

En este caso vamos a usar ipdb (https://pypi.org/project/ipdb/)

Instalar ipdb

```
$ pip install ipdb
```

Debug de código

Uso de ipdb

```python
import ipdb

def foo():
    # codigo.
    ipdb.set_trace()
    # codigo.
```

```
(venv) AR0FVFYH14SHV2J:presentations pdalmasso$ python ../../kaggle/projects/titanic/dataset/test.py
==============================
> /Users/pdalmasso/Dev/Projects/otros/kaggle/projects/titanic/dataset/test.py(21)<module>()
     20     import ipdb; ipdb.set_trace()
---> 21     print("Cuadrado")
     22     cuadrado = Cuadrado(20)

ipdb>
```

Comandos Básicos

n -> Ejecutar línea y pasar a siguiente.

c -> Continuar hasta el próximo breackpoint.

q -> Salir del debug

Lectura y Escritura de archivos

La función open() tiene por objeto interactuar con el sistema de archivos local para crear, sobreescribir, leer o desplazarse dentro de un archivo ya sea de texto o binario.

```
<nombre> = open(<ruta del archivo>,
<modo>)
```

Lectura y Escritura de archivos

Modos de apertura

Según Tipo	Modo	Descripción
Archivo	't'	Se trata de un archivo de texto.
Archivo	'b'	Permite escritura en modo binario
Archivo	'U'	Define saltos de línea universales para el modo de lectura.
Acceso	'r'	Es el modo de lectura.
Acceso	'w'	Es un modo de escritura. En caso de existir un archivo, éste es sobreescrito.
Acceso	'a'	Es un modo de escritura. En caso de existir un archivo, comienza a escribir al final de éste.
Acceso	'x'	Es un modo de escritura para crear un nuevo archivo.
Acceso	'+'	es un modo de escritura/lectura.

Lectura y Escritura de archivos

Escritura de texto.

```
with open("prueba.txt", "w") as archivo:
    archivo.write("\nBienvenido al curso.\nEspero que sea una agradable experiencia.")
    print(archivo.tell())
    print(type(archivo))
```

Lectura de texto.

```
archivo = open("prueba.txt", "r")
print(archivo.readline())
archivo.close()
```

<COMPLETAR>

<COMPLETAR>