Python Fundamentals for Data Science

November 5, 2024

[1]:
```
#Texto Justificado

from IPython.core.display import HTML
HTML("""
<style>
p {
text-align: justify;
}
</style>
""")
```

[1]:
```
<IPython.core.display.HTML object>
```

En Python, la unidad básica del trabajo son los **módulos**.

Dentro de los módulos, tenemos estructuras ejecutables ("callable") denominadas **clases**. Las clases incluyen *atributos* (variables) y *métodos* (funciones).

Finalmente, un conjunto de módulos conforma las ***librerías***.

Algunas de las librerías más comunes en Python son:

```
- Numpy y Pandas (basado en Numpy): para operaciones aritméticas y de análisis de datos
- Matplotlib y Seaborn (su alternativa): para visualización de datos
```

Adicionalmente, está bien tener en cuenta:

```
- Os, Sys (para configuraciones internas, caso de establecer directorio o fijar hora)
- Random (esta librería es importante en Data Science a efectos de reproducibilidad,
que se fija con random.seed () )
```

Finalmente, tenemos algunas librerías más avanzadas:

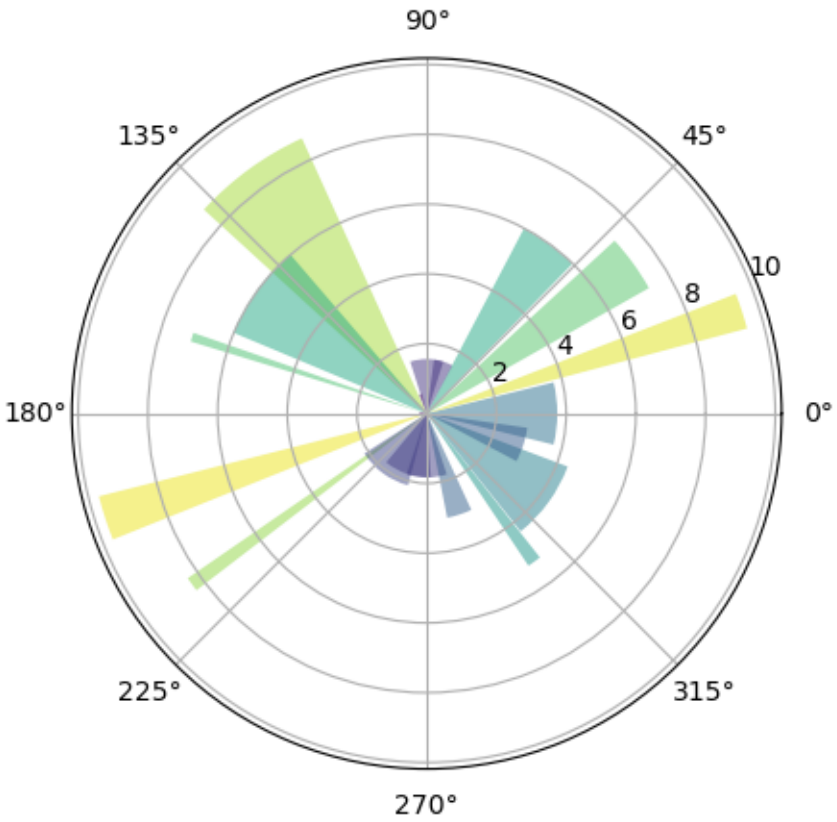
```
- Scikit-Learn para Machine Learning
- Keras, TensorFlow o PyTorch para Deep Learning
```

Nota: hemos marcado los nombres de las librerías en mayúscula, pero por convención escribimos en Python en minúscula.

Para ejecutar el código, sólo tenemos que hacer click en "Run". In[*] nos muestra su ejecución.

[2]:
```
#Veamos lo que podemos hacer en Python con unas pocas líneas de código en Numpy␣
↪(análisis de datos) y Matplotlib (visualización):
```

```
import matplotlib.pyplot as plt
import numpy as np

#Reproducibilidad
np.random.seed(42)

#Parametrización
N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
colors = plt.cm.viridis(radii / 10.) #Añadimos la paleta de colores "viridis",␣
 ↪muy popular en programación

#Visualización
ax = plt.subplot(projection='polar')
ax.bar(theta, radii, width=width, bottom=0.0, color=colors, alpha=0.5)

plt.show()

#Como vemos, los comandos siempre van acompañados de (), con el argumento dentro

#Adicionalmente, vemos que los comandos únicos son resaltados en color, en␣
 ↪tanto que directamente ejecutables
```



En Python, tenemos 3 tipos de “cells”:

- Markdown, en las que sólo escribimos texto y fórmulas (como en esta "cell")

- Code, para código, en las que -como hemos visto en la "cell" anterior- las anotaciones de texto y/o fórmulas van con "#" para distinguirlas del código

- Raw NB Convert, que no se usa mucho en Data Science,
pero que sirve para convertir a otros formatos (HTML o LaTeX)

[3]:
```
#Las operaciones más sencillas vienen incorporadas por "default"

#Recordemos que trabajamos con estructuras "callable". Para ejecutarlas, está␣
 ↪el comando "print":

any_sum = 5 + 39
print(any_sum)

any_rest = 79 - 23
print(any_rest)
```

```
any_multiplication = 4 * 32
print(any_multiplication)

any_division = 100 / 4
print(any_division)

any_exponent = 3 ** 4
print(any_exponent)

any_remainder = 20 // 3
print(any_remainder)
```

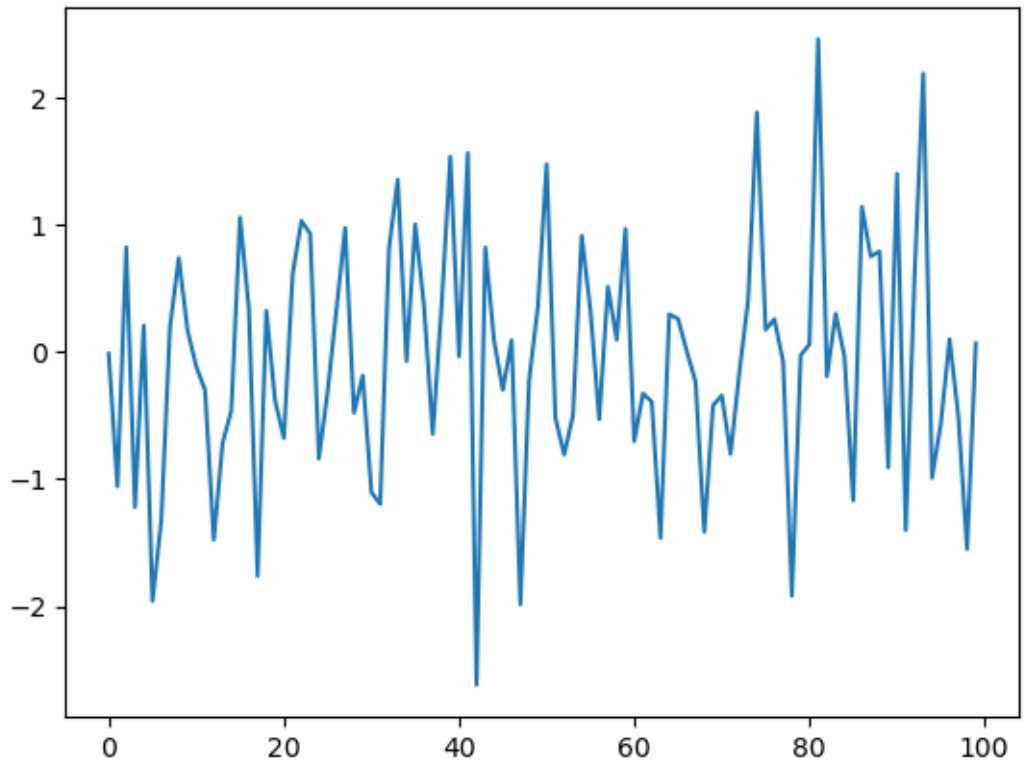
```
44
56
128
25.0
81
6
```

[4]:
```
#A partir del código anterior, en lo referente a los números, vemos dos tipos␣
 ↪diferentes: "integers" (números enteros) y "floats" (números decimales)

#Podemos convertir "integers" en "floats" y al revés:

any_integer = 5
converted_to_float = float(any_integer)
print(converted_to_float)

converted_to_integer = int(converted_to_float)
print(converted_to_integer)

#Adicionalmente, podemos por supuesto incluir palabras (las letras a secas son␣
 ↪más para variables):
any_word = "Hello, World!"

print(any_word)
```

```
5.0
5
Hello, World!
```

[5]:
```
#Para operaciones un poco más complejas (raíces cuadradas, logaritmos,␣
 ↪sequencias de números aleatorios), necesitamos librerías:
#(Es decir, estas operaciones no vienen por "default")

#Usemos, por ejemplo, Numpy para programar una secuencia de 100 valores␣
 ↪aleatores que siguen una Distribución Normal Estándar y Matplotlib para␣
 ↪visualizarlos:
```

```
import numpy as np
import matplotlib.pyplot as plt

epsilon_values = np.random.randn(100)
plt.plot(epsilon_values)
plt.show()

#Si desconocemos algún comando, podemos consultarlo de la forma siguiente:

?np.random.randn

#Vemos que nos aparece un cuadro externo con la explicación

#Por último, tener en cuenta que si no importamos la librería "random" y no␣
 ↪usamos el comando "random.seed()", la ejecución de la secuencia aleatoria no␣
 ↪será reproducible
```



[6]:
```
#Veamos ahora, con el ejemplo de una raíz cuadrada, las 2 posibles formas de␣
 ↪importar librerías y módulos:

import numpy as np
```

```
np.sqrt(4)

#En este caso, importamos el conjunto de la librería y utilizamos un módulo␣
 ↪concreto
```

[6]: 2.0

[7]:
```
#La otra forma es importar un módulo por cada operación que queramos hacer:

from numpy import sqrt
sqrt(4)

#Esto es más eficiente en términos computacionales, pero menos práctico para␣
 ↪propósitos de Data Science

#Sin embargo, esta forma es bastante común en tareas más avanzadas de Machine␣
 ↪Learning y Deep Learning
```

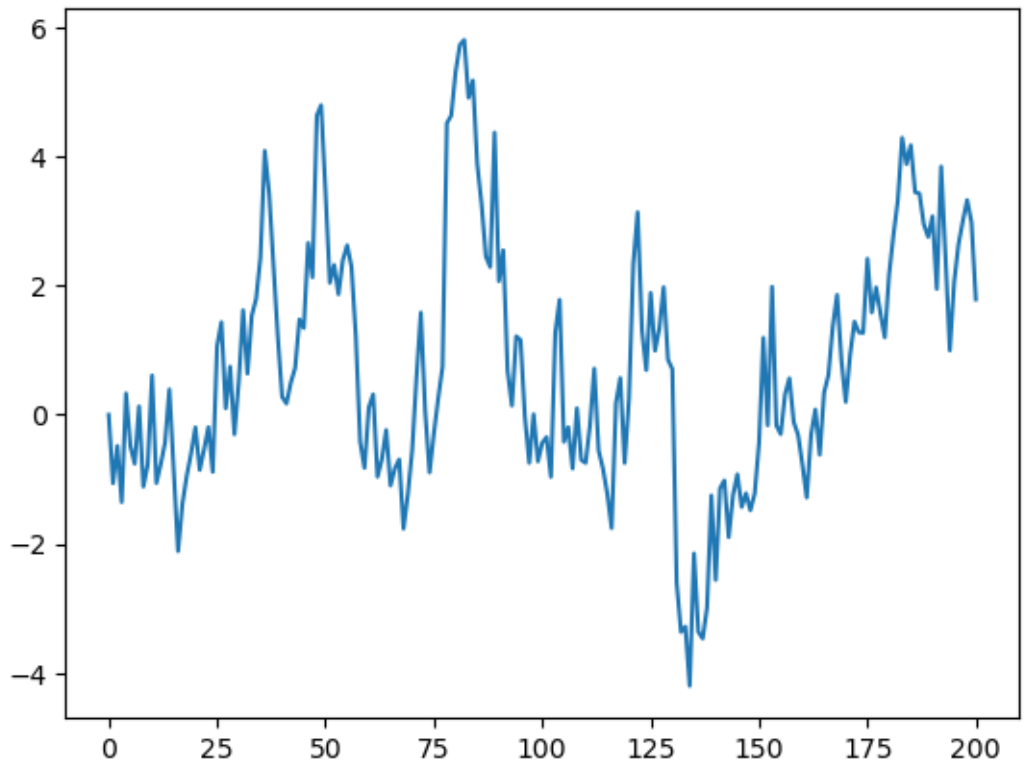
[7]: 2.0

[8]:
```
#En Python, tenemos 4 tipos o estructuras de datos básicas:

#1. LISTS, entre []

animals = ['dog', 'cat', 'bird']
print(animals)

#2. TUPLES, entre () - Como Lists, pero con la diferencia de que no podemos␣
 ↪permutar los valores
#Debido a esta falta de flexibilidad, no es una estructura que use mucho, pero␣
 ↪está bien saber que existe

animals_tuple = ('dog', 'cat', 'bird')
print(animals_tuple)

#3. DICTIONARIES, entre {} - Muy útil para asignación de variables

#Adicionalmente, veamos otra forma de agregar variables

phonebook = {}

phonebook["John"] = {"Phone": "012 794 794", "Email": "john@email.com"}

phonebook["Jill"] = {"Phone": "012 345 345", "Email": "jill@email.com"}

phonebook["Joss"] = {"Phone": "012 321 321", "Email": "joss@email.com"}
```

```
print(phonebook)

#Vemos, por cierto, que podemos utilizar tanto '...' como "..."

#4. SETS - LISTS Sin Registros Duplicados

animals_set = set(["dog", "cat", "cat", "cat", "rabbit"])
print(animals_set)
```

```
['dog', 'cat', 'bird']
('dog', 'cat', 'bird')
{'John': {'Phone': '012 794 794', 'Email': 'john@email.com'}, 'Jill': {'Phone':
'012 345 345', 'Email': 'jill@email.com'}, 'Joss': {'Phone': '012 321 321',
'Email': 'joss@email.com'}}
{'dog', 'rabbit', 'cat'}
```

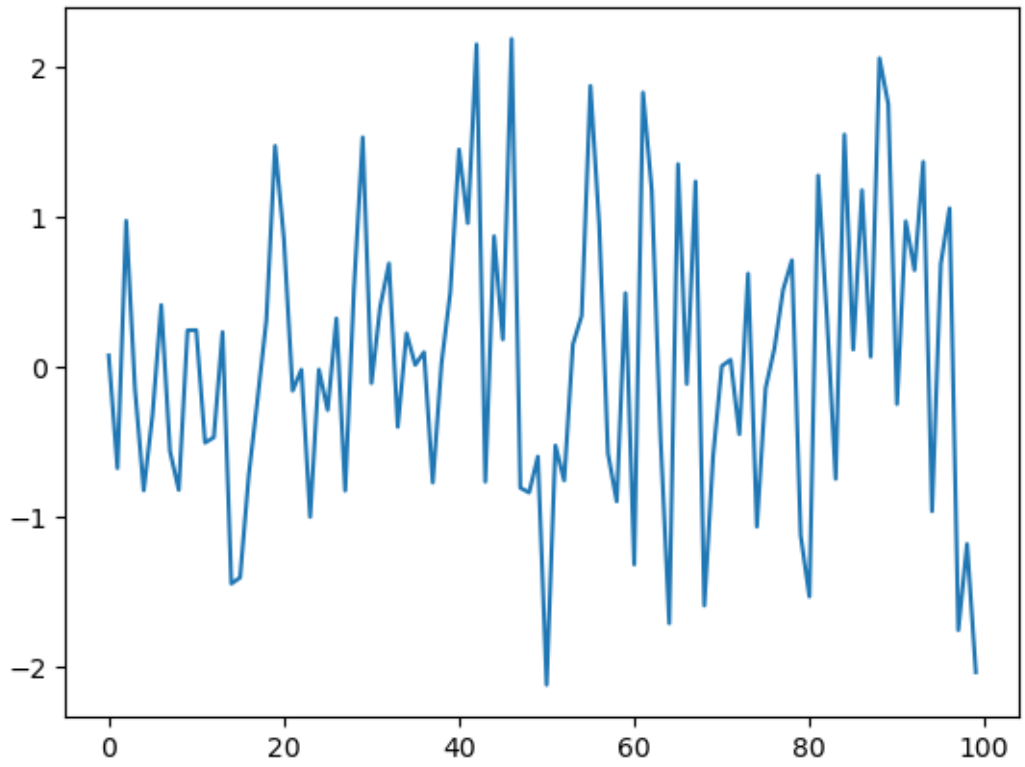
[9]:
```
#LOOPS

#Tenemos 2 tipos de Loops principales: "For" y "While"

#Ejemplo de un "For" Loop:

import numpy as np
import matplotlib.pyplot as plt

T = 200
x = np.empty(T + 1)
x[0] = 0
alpha = 0.9

for t in range(T):
    x[t + 1] = alpha * x[t] + np.random.randn()

plt.plot(x)
plt.show()

#En este Jupyter Notebook de iniciación a Python, estamos poniendo las␣
 ↪librerías en cada "cell"

#Por convención en programación, las librerías y los módulos se ponen todos␣
 ↪juntos al inicio - lo haremos en trabajos futuros más amplios
```



[10]:
```
#Ejemplo del Loop "While":

import numpy as np
import matplotlib.pyplot as plt

ts_length = 100

epsilon_values = []

i = 0

while i < ts_length:
    e = np.random.randn()
    epsilon_values.append(e)
    i = i + 1

plt.plot(epsilon_values)

plt.show

#Fíjemos, adicionalmente, la atención en las "indentations": es la forma de␣
 ↪estructurar lógicamente lo que queremos hacer
```

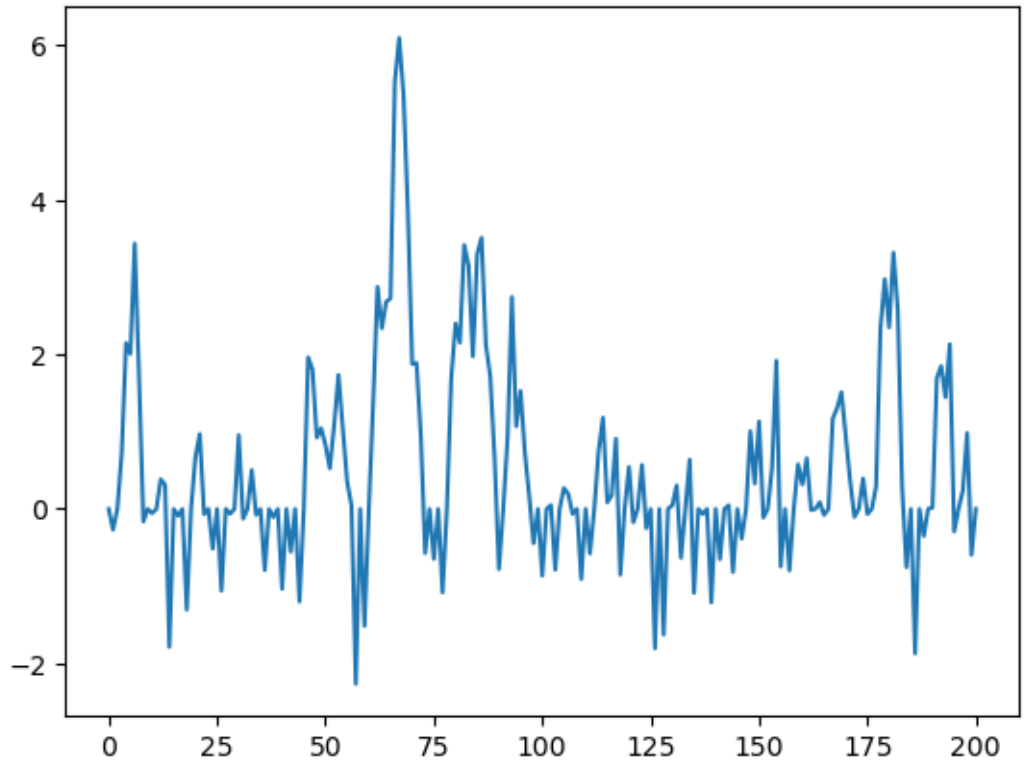
```
#Vemos que nos aparece un mensaje "<function matplotlib.pyplot.show(close=None,␣
 ↪block=None)>"

#Esto es porque hemos usado un comando muy simple ("plt.plot") a secas en el␣
 ↪que no hemos definido todos los argumentos
```

[10]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[11]:
```
#STATEMENTS - "If" y "Else"

#Adicionalmente, tenemos Statements de carácter más implícito (que se usan␣
 ↪dentro de los argumentos), caso de "True" o "False"

#Se pueden utilizar con Loops y otros tipos de objetos en Python

#Fijémonos en que cada vez vamos teniendo código más largo. Para ello, es␣
 ↪importante respetar la lógica de las indentaciones.

import numpy as np
import matplotlib.pyplot as plt
```

```
a = 0.9
T = 200
x = np.empty(T+1)
x[0] = 0

for t in range(T):
    if x[t] < 0:
        abs_x = - x[t]
    else:
        abs_x = x[t]
        x[t+1] = a * abs_x + np.random.randn()

plt.plot(x)
plt.show()
```



[12]:
```
#Todo lo que hemos visto anteriormente se puede utilizar "al mismo tiempo" para␣
 ↪el propósito que deseemos

#Veamos, por ejemplo, una forma "manual" de aproximar el valor de "pi":

n = 100000
count = 0
```

```
for i in range(n):
    u, v = np.random.uniform(), np.random.uniform()
    d = np.sqrt((u - 0.5)**2 + (v - 0.5)**2)
    if d < 0.5:
        count += 1

area_estimate = count / n

print(area_estimate * 4)

#Adicionalmente, fijemos la atención en que en esta "cell", si bien trabajamos␣
 ↪con Numpy, no hemos cargado la librería en esta "cell"

#Esto es porque la hemos cargado en las "cells" anteriores.

#Por esta razón, en trabajos de Data Science amplios, tiene sentido cargarla␣
 ↪sólo una vez en la "cell" del inicio

#Como hemos dicho anteriormente, los operadores aritméticos básicos vienen por␣
 ↪"default". (Al igual que Loops y Statements.)

#Así, hemos usado Numpy únicamente para la distribución uniforme aleatoria y la␣
 ↪raíz cuadrada
```

```
3.14672
```

[13]:
```
#FUNCIONES

#El último tipo de objeto fundamental que vamos a ver son las Funciones

#Su sentido es análago al que se les da en las matemáticas

#Utilizamos los comandos "def" (para definir) y "return" (la operación a␣
 ↪realizar), y ejecutamos la función -una vez definida- con los argumentos que␣
 ↪deseemos:

def sum_two_numbers(x, y):
    return x + y

sum_two_numbers(5, 10)
```

[13]:
```
15
```

[14]:
```
#Al principio, hemos dicho que tanto las variables como las funciones conforman␣
 ↪las "clases"
```

```python
#Las clases, a su vez, conforman los módulos; y los módulos, librerías

#Veamos un tipo de clases para una problemática muy común: los errores

#Vamos a definir nosotros mismos un error y analizar por qué se produce

class ZeroArgError(Exception):
    pass

def check_zero(old_function):
    """
    Check the argument passed to the function to ensure it is not zero.
    """
    def new_function(arg):
        if arg == 0:
            raise ZeroArgError ("Zero is passed to the argument")
        old_function(arg)
    return new_function

@check_zero
def print_num(num):
    return(num)

print_num(57)
print_num(0)

#Así, vemos que el número 57 no da ningún error, pero sí el número 0 (en tanto␣
 ↪que definimos previamente que así fuese)

#Como hemos dicho, los errores son muy comunes y se trata de entender solamente␣
 ↪a qué se deben para poder subsanarlos

#En este caso en concreto, basta con que el número a retornar sea cualquiera␣
 ↪distinto de 0
```

```
---------------------------------------------------------------------------
ZeroArgError                              Traceback (most recent call last)
Cell In[14], line 27
     24     return(num)
     26 print_num(57)
---> 27 print_num(0)

Cell In[14], line 18, in check_zero.<locals>.new_function(arg)
     16 def new_function(arg):
     17     if arg == 0:
---> 18         raise ZeroArgError ("Zero is passed to the argument")
     19     old_function(arg)
```

```
ZeroArgError: Zero is passed to the argument
```

[15]:
```
#A efectos de que quede más claro todavía lo que son las "clases", veamos el siguiente ejemplo de conversión de divisa:

exchange = {"SGD":{"Euro":1.44}}

class Money:

    def __init__(self, amount, currency):
        self.amt = amount
        self.currency = currency

    def __add__(self, money):
        if isinstance(money, self.__class__):
            if self.currency == money.currency:
                return Money(self.amount + money.amt, self.currency)
            else:
                converted_rate = exchange[self.currency][money.currency]*money.amt
                return Money(self.amt + converted_rate, self.currency)

    def __sub__(self, money):
        money.amt*=-1
        return self.__add__(money)

    def convert(self, currency):
        converted_rate = exchange[self.currency][currency]*self.amt
        return Money(converted_rate, currency)

    def __repr__(self):
        return "The amount is {} in {}".format(round(self.amt,2),self.currency)

Money(20,'SGD') + Money(40,'Euro')

#Vemos que una única clase encapsula todas las funciones

#Éste es el motivo (eficiencia computacional) por el que en trabajos avanzados de Data Science, elegimos importar sólo unos pocos módulos concretos
```

[15]:
```
The amount is 77.6 in SGD
```

[ ]:
```
#Habiendo visto todo esto, importemos y analicemos nuestro primer documento en formato, por ejemplo, Excel:
```

```
import os #Como comentamos al inicio, esto nos permite establecer directorio

new_directory = 'C:/Users/.../.../...' #Introducir en ... Nombre de Usuario,␣
 ↪Escritorio, Carpeta

os.chdir(new_directory) #Fijamos directorio

dataset = read_csv('....csv', sep=';', decimal='.') #Nombre del Dataset en ....
 ↪csv

print(dataset.info()) #Python nos permite observar el Dataset en un bonito␣
 ↪formato visual
```

Recomendaciones de lectura para profundizar:

- Teik Toe Teoh & Rong Zhen. (2022). "Artificial Intelligence with Python". Springer.

(Nota: algunos ejemplos han sido tomados de este libro.)

- Teik Toe Teoh & Yu Jin Goh (2023). "Artificial Intelligence in Business Management". Springer.

Autor del curso: Artem Urlapov Sedova (Universidad Autónoma de Madrid).