# Curso de Python Básico

Sergio Heredia Carmona

Octubre 2023

# Índice

# 1 Introducción

## 1.1 ¿Qué es Python?

Python es un lenguaje de programación de uso genérico, orientado a objetos, de alto nivel, fuertemente tipado, dinámico e interpretado.

- **Uso genérico** porque se puede usar para cualquier campo en cualquier máquina
- **Alto nivel** porque es muy parecido al lenguaje natural (a más nivel más parecido al humano)
- **Fuertemente tipado** significa que el tipo de dato de una variable no cambia de forma mágica si no de forma explícita.
- **Dinámico** significa que no necesitamos definir previamente con qué dato vamos a trabajar. Directamente trabajamos con lo que nos venga.
- **Interpretado** significa que al mismo tiempo que el intérprete lee una parte del código, la traduce a 0s y 1s y la ejecuta. Esto hace que sea más lento porque en un lenguaje compilado, el compilador traduce directamente todo a 0s y 1s y ejecuta del tirón sin tener que estar leyendo, traduciendo y ejecutando a cada rato.

Este fue creado por Guido van Rossum porque los lenguajes de programación de entonces no hacían frente a algunas de las necesidades que este necesitaba.

Su nombre no se debe a la serpiente sino a los Monty Python.

## 1.2 ¿Qué es un Código o Programa?

Un código, está constituido por una serie de sentencias ordenadas una debajo de la otra (de la misma forma que escribimos en un folio) las cuales son ejecutadas por una máquina.

Un sentencia es una línea de código que da una orden a la máquina. Dentro de las sentencias tenemos un tipo de especial que se llama comentario. Esta sentencia no hace nada. Solo está para explicar partes del código o aclaraciones necesarias.

[1]:
```
# Comentario de linea
"""
Comentario de bloque
"""

1 + 1
```

[1]:
```
2
```

## 1.3 Archivos de Python

En Python tenemos al menos 2 tipos de archivos con los que podemos programar

- Los *.py o archivo de Python
- Los *.ipynb o Jupyter Notebooks

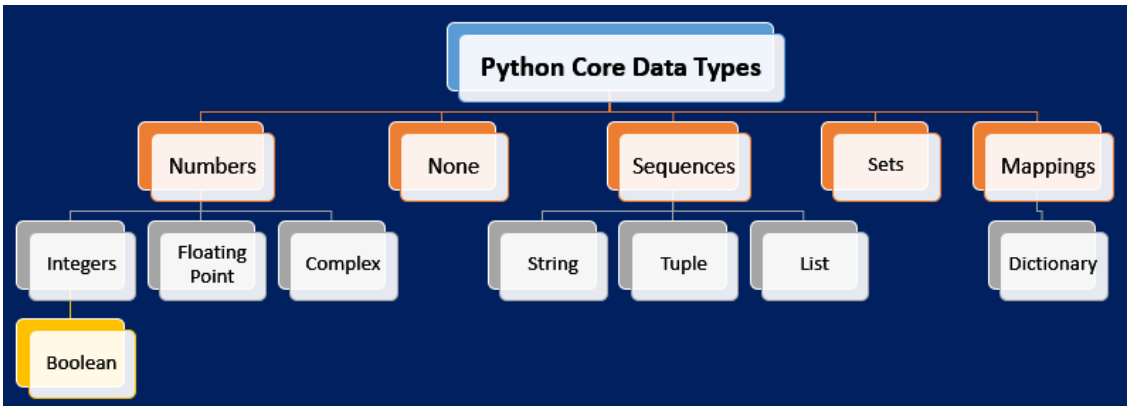
Los archivos de Python (*.py) se suelen usar para programación más compleja y estructurada. A veces puede ser necesario descomponer un programa en varios archivos y para ello suele ser más cómodo usar varios archivos* .py

Por contra, aunque los Jupyter Notebooks tambien se pueden utilizar para computación compleja (Ej: Machine Learning), se suelen usar para programas más interactivos o didácticos en donde podemos dividir nuestro código en lo que se llama **celdas**.

Entre las celdas destacamos:

- **Las celdas de texto** donde podríamos entenderlas como la parte donde pondremos comentarios, explicaciones en un formato más rico como tablas, enlaces a webs, ...
- **Las celdas de código** donde realmente escribimos las sentencias que nuestra máquina ejecutará

# 2 Tipos de datos




## 2.1 Int

Representa los números enteros. . . . , -1, 0, 1, . . .

[1]:
```
1
```

[1]:
```
1
```

[2]:
```
-1
```

[2]:
```
-1
```

[3]:
```
-3000000
```

[3]:
```
-3000000
```

## 2.2 Float

Representa los números reales. Cualquier número con o sin decimales que no sea complejo.

[4]:
```
1.0
```

[4]:
```
1.0
```

[5]:
```
.5
```

[5]:
```
0.5
```

[6]:
```
-30.0
```

[6]:
```
-30.0
```

## 2.3 Bool

Se basa en el Algebra de Bool. Sirve para indicar la veracidad o falsedad de algo.

[7]:
```
True
```

[7]:
```
True
```

[8]:
```
False
```

[8]:
```
False
```

## 2.4 Secuencias

### 2.4.1 String

Nos permite definir texto. Letras, palabras, oraciones, ...

[9]:
```
'hola mundo'
```

[9]:
```
'hola mundo'
```

[10]:
```
"hola mundo"
```

[10]:
```
'hola mundo'
```

[11]:
```
"it's"
```

[11]:
```
"it's"
```

### 2.4.2 List

Permite organizar datos ordenados según el orden en el que se añadieron a la lista.

Cada elemento de la lista tiene un índice que empieza en 0 e indica el orden de entrada en la misma.

Sintaxis:

- variable = [dato, dato, ..]

Ej:

- list : [‘a’,‘b’,‘c’,‘d’,‘e’] || índices: [0, 1, 2, 3, 4]

[12]:
```
[] # vacía
```

[12]:
```
[]
```

[13]:
```
[1, 2, 3]
```

[13]:
```
[1, 2, 3]
```

[14]:
```
[1, [12, 13, 14], 'palabra']
```

[14]:
```
[1, [12, 13, 14], 'palabra']
```

### 2.4.3 Tuple

Es lo mismo que una lista. Con la diferencia de que es **immutable**
Es decir, para **cambiar** un **elemento inmutable** de la tupla, debemos **redefinir** la **tupla**.

La tupla realmente tiene dos usos:

- Como elemento inmutable
- Como registro de datos

Dado que no permite la modificación de datos durante la ejecución de nuestro código (runtime), podemos almacenar datos que sabemos que no serán modificados a lo largo del tiempo:

Ejemplo: persona = ('Fran', 12) # (nombre, edad)

Sintaxis:

- variable = (dato, dato, ..)

[15]:
```
() # vacía
```

[15]:
```
()
```

[16]:
```
(1, ) # las tuplas con un único elemento deben tener una coma después
```

[16]:
```
(1,)
```

[17]:
```
(1, 2, 4)
```

[17]:
```
(1, 2, 4)
```

[18]:
```
(1, 'a', [1, 2,], (1, (1, 2)))
```

[18]:
```
(1, 'a', [1, 2], (1, (1, 2)))
```

## 2.5 Set

Representa el concepto matemático de conjunto. Lista elementos (ordenada o no) que no pueden repetirse.

Sintaxis: - variable = {dato, dato, ..}

[19]:
```
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3}
```

[19]:
```
{1, 2, 3}
```

[20]:
```
{1, -1, 'a', (1, -1)}
```

[20]:
```
{(1, -1), -1, 1, 'a'}
```

[21]:
```
{[1, 2], [3, 4]} # solo elementos inmutables
```

```
TypeError: unhashable type: 'list'
```

### 2.6 Dict

Es una lista de pares **clave - valor**. Se puede interpretar como una lista cuyos datos guardados son los valores y los indices son las claves.

**Nota:** Las claves se guardan como un conjunto. Es decir, en un diccionario no hay 2 claves iguales.

[22]:
```
{}
```

[22]:
```
{}
```

[23]:
```
{
    '12345678A' :  {
        'Nombre' : 'Sergio',
        'Edad' : 13,
        'Trabajo' : '????????',
    },
    '91234567B' :  {
        'Nombre' : 'Fran',
        'Edad' : 45,
        'Trabajo' : '????????',
    },
}
```

[23]:
```
{'12345678A': {'Nombre': 'Sergio', 'Edad': 13, 'Trabajo': '????????'},
 '91234567B': {'Nombre': 'Fran', 'Edad': 45, 'Trabajo': '????????'}}
```

[24]:
```
{ 'clave' : 1, 1 : [1, 2, 3] }
```

[24]:
```
{'clave': 1, 1: [1, 2, 3]}
```

[25]:
```
{ (1, 2) : 2 }
```

[25]:
```
{(1, 2): 2}
```

[26]:
```
{ [1] : 1 }
```

```
TypeError: unhashable type: 'list'
```

### 2.7 None

Representa la nada. La ausencia de datos.

[ ]:
```
none1 = None

none1
```

### 2.8 Tipos de datos mutables y no mutables:

Mutables:

- List
- Set
- Dict

Inmutables:

- Int
- Float
- Bool
- String
- Tuple

# 3 Operadores

## 3.1 Definición de variable

Una **variable** es un espacio dentro de nuestro código que nos permite **almacenar** y **acceder** a ciertos **datos** en el momento que lo necesitemos

Los nombres de variables en Python se definen de la siguiente forma:

- Cualquier cantidad de letras (A-Z, a-z), dígitos (0-9) y el símbolo _ (guión bajo)
- Una variable no puede empezar con un dígito
- Se sigue la norma snake_case

Ejemplos válidos:

- fecha_matriculacion, nombre_usuario

Ejemplos no válidos:

- 1aaaa, _*?

### 3.2 Operador de asignación

Se corresponde con el símbolo =. Sirve para asignar datos a variables.

En la actualizad, cuando definimos una variable (no siempre hay que hacerlo) aplicamos un concepto llamado **Type Hint**. Este nos permite indicar qué dato **debe** guardar una variable concreta.

Recordemos que **Python es dinámicamente tipado**, por lo que en una variable podemos meter lo que queramos, el **Type Hinting** simplemente es una buena práctica para hacer nuestro código más legible. No obliga a que una variable tenga un dato concreto.

[1]:
```
entero : int = 1
entero
```

[1]:
```
1
```

[2]:
```
decimal : float = 1.0
decimal
```

[2]:
```
1.0
```

[3]:
```
cierto : bool = True
cierto
```

[3]:
```
True
```

[4]:
```
cadena_texto : str = 'cadena'
cadena_texto
```

[4]:
```
'cadena'
```

[5]:
```
lista : list = [1, 2, '1']
lista
```

[5]:
```
[1, 2, '1']
```

[6]:
```
tupla : tuple = (1, 2, 3)
```

[7]:
```
conjunto : set = {1, 2, 3, 3, 3, 3, 3}
conjunto
```

[7]:
```
{1, 2, 3}
```

[8]:
```
diccionario : dict = {'persona' : 23}
diccionario
```

[8]:
```
{'persona': 23}
```

[9]:
```
nada : None = None
nada
```

### 3.3 Operadores aritméticos

En python existen 7 operaciones aritméticas a nuestra disposición por defecto.

Operaciones:

- Suma: +
- Resta: -
- Producto: *
- División: /
- Potencia: **
- División Entera: //
- Módulo o resto: %

#### 3.3.1 Números (Int y Float)

[10]:
```
1 + 42
```

[10]:
```
43
```

[11]:
```
4 - 10
```

[11]:
```
-6
```

[12]:
```
3 * 4 * 2
```

[12]:
```
24
```

[13]:
```
6 / 2
```

[13]:
```
3.0
```

[14]:
```
2 ** 3
```

[14]:
```
8
```

[15]:
```
6 // 2
```

[15]:
```
3
```

[16]:
```
3 % 2
```

[16]:
```
1
```

Podemos combinarlas como queramos
Recordemos que la prioridad es PEMDAS (Paréntesis, Exponente, Multiplicación, División, Adicion, Subtracción)

[17]:
```
1 + 2 * 3 // 4
```

[17]:
```
2
```

[18]:
```
1 + (2 * 3) // 4
```

[18]:
```
2
```

[19]:
```
(2 + 2) * (3 + 4)
```

[19]:
```
28
```

[20]:
```
2 ** 3 / 4
```

[20]:
```
2.0
```

[21]:
```
2 ** (3 / 4)
```

[21]:
```
1.681792830507429
```

### 3.3.2 Bool

[22]:
```
True + True
```

[22]:
```
2
```

[23]:
```
True / False
```

```
ZeroDivisionError: division by zero
```

[24]:
```
False * True
```

[24]:
```
0
```

### 3.3.3 Secuencias (String, List, Tuple)

- La suma de **sencuencias** concatena **secuencias**
- **secuencia** * n genera una nueva **secuencia** replicada n veces
- El resto de operaciones da error

[25]:
```
'hola ' + 'mundo'
```

[25]:
```
'hola mundo'
```

[26]:
```
'palabra ' * 3
```

[26]:
```
'palabra palabra palabra '
```

[27]:
```
[1] + [3]
```

[27]:
```
[1, 3]
```

[28]:
```
[1] * 4
```

[28]:
```
[1, 1, 1, 1]
```

[29]:
```
(1, ) + ('a', (1, ))
```

[29]:
```
(1, 'a', (1,))
```

[30]:
```
(1, [1, 2]) * 2
```

[30]:
```
(1, [1, 2], 1, [1, 2])
```

### 3.3.4 Set

Solo funciona la resta como diferencia asimétrica

[31]:
```
{1, 2} - {1, 2, 3}
```

[31]:
```
set()
```

[32]:
```
{1, 2, 3} - {1, 2}
```

[32]:
```
{3}
```

### 3.3.5 Dict y None

Da error cualquier operación

[33]:
```
{ 'a' : 1 } + {'a' : 1}
```

```
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

[34]:
```
None + None
```

```
TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

### 3.3.6 Ejercicios

- Sumar varios números en una misma sentencia
- Combinar operadores y paréntesis para formar cálculos largos
- Usar variables y operadores

## 3.4 Operadores bit a bit (bitwise)

Lo único que entienden las computadoras son 0s y 1s.

Todo está representado en binario.

Las siguientes operaciones se realizan normalmente sobre los números enteros.

Operaciones:

- AND: **&**
- OR: |
- XOR: ^
- NOT: ~
- SHIFT LEFT: «
- SHIFT RIGHT: »

### 3.4.1 AND

La sintaxis es **int & int**.

Cuando aplicamos esta operación lo que hacemos es ejecutar bit a bit la **puerta lógica AND**. Solo obtenemos 1 cuando ambos valores sean 1.

AND:

- 0 & 0 -> 0
- 0 & 1 -> 0
- 1 & 0 -> 0
- 1 & 1 -> 1

[35]:
```
"""

1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

2 & 5 = 010 & 101 = 000 = 0
1 & 5 = 001 & 101 = 001 = 1

"""

2 & 5, 1 & 5
```

[35]: (0, 1)

### 3.4.2 OR

La sintaxis es **int | int**.

Cuando aplicamos esta operación lo que hacemos es ejecutar bit a bit la **puerta lógica OR**. Cuando alguno de los valores sea 1, obtenemos 1.

OR:

- 0 | 0 -> 0
- 0 | 1 -> 1
- 1 | 0 -> 1
- 1 | 1 -> 1

[36]:
```
"""

1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

2 | 5 = 010 | 101 = 111 = 7
1 | 5 = 001 | 101 = 101 = 5

"""

2 | 5, 1 | 5
```

[36]: (7, 5)

### 3.4.3 XOR

La sintaxis es **int ^ int**.

Cuando aplicamos esta operación lo que hacemos es ejecutar bit a bit la **puerta lógica XOR**. Se conoce como **o exclusivo** porque solo obtenemos 1 cuando los valores son distintos.

XOR:

- 0 ^ 0 -> 0
- 0 ^ 1 -> 1
- 1 ^ 0 -> 1
- 1 ^ 1 -> 0

[37]:
```
"""

1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

2 ^ 5 = 010 ^ 101 = 111 = 7
1 ^ 5 = 001 ^ 101 = 100 = 4

"""

2 ^ 5, 1 ^ 5
```

[37]:
```
(7, 4)
```

### 3.4.4 NOT

La sintaxis **~int**.

Cuando aplicamos esta operación lo que hacemos es ejecutar bit a bit la **puerta lógica NOT**. Lo que es 1 se hace 0 y lo que es 0 se hace 1.

NOT:

- ~0 -> 1
- ~1 -> 0

[38]:
```
"""

1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

~2 = ~010 = 101 = 5
~1 = ~001 = 110 = 6

"""

~0, ~1
```

[38]:
```
(-1, -2)
```

### 3.4.5 SHIFT LEFT

La sintaxis **int « int**.

Cuando aplicamos esta operación lo que hacemos es desplazar los bit del número n veces a la izquierda.

[39]:
```
"""

1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

1 << 1 = 001 << 1 = 010 = 2

"""

1 << 1
```

[39]: 2

### 3.4.6 SHIFT RIGHT

La sintaxis **int » int**.

Cuando aplicamos esta operación lo que hacemos es desplazar los bit del número n veces a la derecha.

[40]:
```
"""

1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

7 >> 1 = 111 >> 1 = 011 = 3

"""

7 >> 1
```

[40]: 3

### 3.4.7 Ejercicios

- Extraer el tercer y quinto bit (a la vez) de los números 255 y 250

## 3.5 Operadores lógicos

Estas operaciones trabajan bajo el concepto de verdad y mentira. En otras palabras:

- En las operaciones aritméticas trabajamos con números y obtenemos números como resultado (Salvo en Secuencias y Sets que obtenemos Secuencias y Sets respectivamente)
- En las operaciones bit a bit trabajamos con bits y obtenemos números representados con los bits modificados como resultado.

De igual manera, en las operaciones lógicas trabajamos con **Bools** que representan los valores True y False y obtenemos **Bools** como resultado (Salvo en el caso de AND y OR pero eso ya lo dejo como investigación para quien tenga curiosidad).

Operaciones:

- AND: **and**
- OR: **or**
- NOT: **not**
- IS: **is**
- IN: **in**
- EQUALS: ==
- NOT EQUALS: **!**=
- GREATER: >
- LESSER: <
- GREATER OR EQUALS: >=
- LESSER OR EQUALS: <=

### 3.5.1 AND

La sintaxis es **bool and bool**.

Funciona igual que vimos en las operaciones bit a bit. La diferencia es que trabajamos con bools.

AND:

- False and False -> False
- False and True -> False
- True and False -> False
- True and True -> True

[41]:
```
False and False
```

[41]:
```
False
```

[42]:
```
False and True
```

[42]: False

```
[43]: True and False
```

[43]: False

```
[44]: True and True
```

[44]: True

### 3.5.2 OR

La sintaxis es **bool or bool**.

Funciona igual que vimos en las operaciones bit a bit. La diferencia es que trabajamos con bools.

OR:

- False or False -> False
- False or True -> True
- True or False -> True
- True or True -> True

```
[45]: False or False
```

[45]: False

```
[46]: False or True
```

[46]: True

```
[47]: True or False
```

[47]: True

```
[48]: True or True
```

[48]: True

### 3.5.3 NOT

La sintaxis es **not bool**.

Funciona igual que vimos en las operaciones bit a bit. La diferencia es que trabajamos con bools.

NOT:

- not False -> True
- not True -> False

[49]:
```
not False
```

[49]: True

[50]:
```
not True
```

[50]: False

### 3.5.4 IS

La sintaxis es **data is data**.

Compara si 2 datos son idénticos. Es decir, están guardados exactamente en la misma posicion de memoria.

[51]:
```
[1, 2] is [1, 2]
```

[51]: False

[52]:
```
lista_1 : list = [1, 2]
lista_2 : list = lista_1

lista_1 is lista_2
```

[52]: True

### 3.5.5 IN

La sintaxis es **data in data**.

Estudia si un dato está en otro dato que agrupe otros datos como los strings, lists, tuples, sets, and dicts.

[53]:
```
'a' in 'abcdefghijklmnopqrstuvwxyz'
```

[53]: True

[54]:
```
1 in [2, 3, 4]
```

[54]: False

En el caso de los diccionarios, se comprueba si data está entre las claves. No entre los valores.

[55]:
```
'a' in {'a' : 1}
```

[55]: True

[56]:
```
'a' in {1 : 'a'}
```

[56]: False

### 3.5.6 EQUALS

La sintaxis es **data == data**.

Compara si dos variables guardan el mismo valor. No comparamos si guardan el mismo valor en la misma posicion de memoria. Estamos viendo si los valores que contienen coinciden.

[57]:
```
[1, 2] == [1, 2]
```

[57]:
```
True
```

[58]:
```
1 == 'a'
```

[58]:
```
False
```

[59]:
```
1 == 1
```

[59]:
```
True
```

### 3.5.7 NOT EQUALS

La sintaxis es **data != data**.

Compara si dos variables **no** guardan el mismo valor.

[60]:
```
1 != 0
```

[60]:
```
True
```

[61]:
```
'a' != 'a'
```

[61]:
```
False
```

### 3.5.8 GREATER

La sintaxis es **data > data**.

Compara si un dato es mayor que otro.

**Nota:** Se puede escribir dato1 > dato2 > dato3 en una misma sentencia.

[62]:
```
1 > 2
```

[62]:
```
False
```

[63]:
```
2 > 1
```

[63]:
```
True
```

### 3.5.9 LESSER

La sintaxis es **data** < **data**.

Compara si un dato es menor que otro.

**Nota:** Se puede escribir dato1 < dato2 < dato3 en una misma sentencia.

[64]:
```
1 < 2
```

[64]:
```
True
```

[65]:
```
3 < 2
```

[65]:
```
False
```

### 3.5.10 GREATER OR EQUALS

La sintaxis es **data** >= **data**.

Compara si un dato es mayor o igual que otro.

**Nota:** Se puede escribir dato1 >= dato2 >= dato3 en una misma sentencia.

[66]:
```
1 >= -1
```

[66]:
```
True
```

[67]:
```
0 >= -1 >= -2
```

[67]:
```
True
```

### 3.5.11 LESSER OR EQUALS

La sintaxis es **data** <= **data**.

Compara si un dato es menor o igual que otro.

**Nota:** Se puede escribir dato1 <= dato2 <= dato3 en una misma sentencia.

[68]:
```
0 <= 1 <= 10
```

[68]:
```
True
```

### 3.5.12 Extra

Se pueden combinar las operaciones <, >, <=, >= de la forma A >= B < C, ...

[69]:
```
1 <= 2 > 0
```

[69]:
```
True
```

### 3.5.13 Ejercicios

- Comparar números con listas
- Comparar strings entre sí
- Usa los operadores **and**, **or** y **not** con otros tipos de datos como listas, sets, ...

# 4 Control de flujo

## 4.1 Estructura if-elif-else

Normalmente, nuestro código se ejecuta de arriba a abajo linea por linea o sentencia por sentencia. Sin embargo, a veces es necesario ejecutar partes de nuestro código bajo ciertas condiciones. De esta forma conseguimos que algunas partes que no deben de ejecutarse siempre no se ejecuten.

La sintaxis es:

- if condition:
  - code
- elif condition:
  - code
- else:
  - code

Algunos ejemplos son:

- No ejecutar un código si los datos introducidos no son válidos.
- Si un dato tiene un valor concreto, ejecutar una sección del código concreta.

### 4.1.1 Ejemplo 1 | if

Si un dato está entre 0 y 10 lo elevamos al cuadrado. En otro caso nada.

[1]:
```
n : int = -1

if 0 <= n <= 10:
    n = n ** 2

n
```

[1]:
```
-1
```

[2]:
```
n : int = 3

if 0 <= n <= 10:
    n = n ** 2

n
```

[2]:
```
9
```

### 4.1.2 Ejemplo 2 | if-else

Si un dato está entre 0 y 10 lo elevamos al cuadrado. En otro caso al cubo.

[3]:
```
n : int = -2

if 0 <= n <= 10:
    n = n ** 2
else:
    n **= 3

n
```

[3]: -8

[4]:
```
n : int = 3

if 0 <= n <= 10:
    n = n ** 2
else:
    n **= 3

n
```

[4]: 9

### 4.1.3 Ejemplo 3 | if-elif-else

Si una lista está vacía, creamos una lista con un elemento cualquiera. Si no esta vacía y contiene solo el numero 1, vaciamos la lista. En otro caso creamos una lista con los numeros desde el 0 hasta el 9.

[5]:
```
lista : list = []

if lista == []:
    lista = [1]
elif lista == [1]:
    lista = []
else:
    lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

lista
```

[5]: [1]

[6]:
```
lista : list = [1]

if lista == []:
    lista = [1]
elif lista == [1]:
    lista = []
else:
```

```
    lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

lista
```

[6]: []

[7]:
```
lista : list = [2]

if lista == []:
    lista = [1]
elif lista == [1]:
    lista = []
else:
    lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

lista
```

[7]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#### 4.1.4 Ejemplo 4 | Combinación de condiciones

Este ejempo muetra un caso en el que necesitamos **anidar** varias condiciones (estructura if-elif-else). Además, debemos usar operaciones (and, or, not) cuando tenemos 2 o más condiciones dentro de un if o de un elif.

- Si un número es menor que 100
  - Si es par **y** multiplo de 5 entonces lo elevamos al cuadrado.
  - Si es multiplo de 3 le restamos 3.
  - En otro caso le cambiamos el signo.
- Si es mayor que 100 pero menor que 1000
  - Si es impar lo multiplicamos por 2
  - En otro caso no hacemos nada
- En otro caso no hacemos nada

[8]:
```
numero : int = 11

if numero < 100:
    if numero % 2 == 0 and numero % 5 == 0:
        numero = numero ** 2
    elif numero % 3:
        numero -= 3
    else:
        numero = -numero
elif 100 < numero < 1000:
    if numero % 2 == 0:
        numero *= 2

numero
```

[8]: 8

[9]:
```
numero : int = 12

if numero < 100:
    if numero % 2 == 0 and numero % 5 == 0:
        numero = numero ** 2
    elif numero % 3:
        numero -= 3
    else:
        numero = -numero
elif 100 < numero < 1000:
    if numero % 2 == 0:
        numero *= 2

numero
```

[9]: -12

[10]:
```
numero : int = 10

if numero < 100:
    if numero % 2 == 0 and numero % 5 == 0:
        numero = numero ** 2
    elif numero % 3:
        numero -= 3
    else:
        numero = -numero
elif 100 < numero < 1000:
    if numero % 2 == 0:
        numero *= 2

numero
```

[10]: 100

### 4.1.5 Ejercicios

- Usa la estructura if-elif-else con otros tipos de datos con y sin operadores de comparación

## 4.2 Estructuras while y for

En la sección anterior comentamos que nuestro código normalmente se ejecuta en secuencial de arriba a abajo. Decíamos que con la estructura if-elif-else podemos controlar qué partes de nuestro código se ejecutan según una serie de condiciones.

En otras ocasiones necesitamos ejecutar varias veces un mismo bloque de código. Sin embargo, es muy engorroso escribir una y otra vez lo mismo.

Cuando lo que nuestro código hace no cambia, es decir, las líneas de código son las mismas pero tenemos que ejecutarlas varias veces usamos los **bucles**.

Cuando sabemos cuántas veces ejecutaremos una misma sección de código, usamos el **bucle for**.

Este en python funciona recorriendo una variable **iterable**. Es decir, podemos leer cada elemento de la misma sin necesidad de indicar qué elemento queremos leer.

El bucle for indica por nosotros que queremos leer el siguiente elemento en caso de haber más. Independientemente de cual sea ese elemento.

Sintaxis:

- for element in iterable:
  - code
- else:
  - code

Cuando no sabemos cuántas veces ejecutaremos una misma sección de código, usamos el **bucle while**.

En este indicamos una condición y mientras se cumpla dicha condición, el bucle ejecutará lo que haya dentro del mismo.

Sintaxis:

- while condition:
  - code
- else:
  - code

**NOTA:** En la sintaxis vemos que hay una parte adicional **else**. Esta parte es opcional y sirve para ejecutar una porción de código después de que el bucle termine. Solo se ejecuta esta parte si el bucle finalizó sin problemas. Es decir, no ocurrió ningún error que pare la ejecución del mismo de forma inesperada.

### 4.2.1 Ejemplo 1 | for

Obtener la suma de los elementos de una lista

[11]:
```
suma : int = 0

for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    suma += i

suma
```

[11]:
```
45
```

Obtener claves de un diccionario

[12]:
```
claves : list = []

for key in {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e' : 5}:
    claves += [key]

claves
```

[12]:
```
['a', 'b', 'c', 'd', 'e']
```

### 4.2.2 Ejemplo 2 | for con condicionales

Obtener la suma de los elementos pares, los múltiplos de 3 y los impares en 3 variables separadas, de una lista.

[13]:
```
suma_pares : int = 0
suma_multiplos_3 : int = 0
suma_impares : int = 0

for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    if i % 2 == 0:
        suma_pares += i
    else:
        suma_impares += i
    if i % 3 == 0:
        suma_multiplos_3 += i

suma_pares, suma_multiplos_3, suma_impares
```

[13]:
```
(20, 18, 25)
```

### 4.2.3 Ejemplo 3 | for con bucles anidados

Dada una lista de palabras, generar una lista indicando el número de letras tiene cada palabra

[14]:
```
numero_letras_por_palabra : list = []

for palabra in ['hola', 'mundo', 'arbol', 'cigüeña']:
    numero_caracteres = 0
    for caracter in palabra:
        numero_caracteres += 1
    else:
        numero_letras_por_palabra += [(palabra, numero_caracteres)]

numero_letras_por_palabra
```

[14]:
```
[('hola', 4), ('mundo', 5), ('arbol', 5), ('cigüeña', 7)]
```

### 4.2.4 Ejemplo 4 | while

Dado un número:

- Si es par, lo elevamos al cuadrado y sumamos 1
- Si es impar, lo dividimos entre 3 y multiplicamos por 2

Repetir la operación mientras el número sea menor que 100 y el número de veces que se realiza la operación de arriba es menor que 10

[15]:
```
numero : int = 2
iteraciones : int = 0

while numero < 100 and iteraciones < 10:
    if numero % 2 == 0:
        numero = (numero ** 2) + 1
    else:
        numero = (numero // 3) * 2

    iteraciones += 1
else:
    iteraciones = 0

numero
```

[15]:
```
2
```

[16]:
```
numero : int = 3
iteraciones : int = 0

while numero < 100 and iteraciones < 10:
    if numero % 2 == 0:
        numero = (numero ** 2) + 1
    else:
        numero = (numero // 3) * 2
```

```
    iteraciones += 1
else:
    iteraciones = 0

numero
```

[16]: 5

### 4.2.5 Compresión de listas (Avanzado)

De normal, escribir un bucle for es una estructura lenta. A veces interesa más usar compresión de listas que equivale a un bucle for.

La sintaxis normal es:

- [item for item in iterable] -> Genera una lista
- {key : value for item in iterable} -> Genera un diccionario
- (item for item in iterable) -> Podemos pensar que esto genera una tupla pero NO!!!. Genera un tipo de dato especial llamado **generador**. Es un elemento iterable, pero no está no está específicamente almacenado en memoria por lo que no podemos acceder directamente a un dato concreto. Tenemos que recorrer y cada dato que se irá generando.

Sin compresión listas

[17]:
```
lista : list = []
for i in range(10):
    lista.append(i ** 2)

lista
```

[17]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

[18]:
```
diccionario : dict = {}
for i in lista:
    diccionario[i] = i ** 3

diccionario
```

[18]:
```
{0: 0,
 1: 1,
 4: 64,
 9: 729,
 16: 4096,
 25: 15625,
 36: 46656,
 49: 117649,
 64: 262144,
```

```
 81: 531441}
```

Con compresión de listas

[19]:
```
lista : list = [i ** 2 for i in range(10)]

lista
```

[19]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

[20]:
```
diccionario : dict = {i : i ** 3 for i in lista}

diccionario
```

[20]:
```
{0: 0,
 1: 1,
 4: 64,
 9: 729,
 16: 4096,
 25: 15625,
 36: 46656,
 49: 117649,
 64: 262144,
 81: 531441}
```

#### 4.2.5.1 Rizando el rizo

Como todo, mejor no abusar, pero al igual que los bucles se pueden anidar y escribir un bucle dentro de un bucle, podemos hacer los mismo con la compresión de listas.

Sin compresión de listas

[21]:
```
matriz : list = []

for i in range(3):
    row = []
    for j in range(3):
        row += [0]
    matriz += [row]

matriz
```

[21]:
```
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Con compresión de listas

[22]:
```
matriz : list = [[j + 1 + i * 3 for j in range(3)] for i in range(3)]

matriz
```

[22]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

### 4.2.6 Ejercicios

- Dada una lista de números cualquiera
  - Crea una lista nueva con los números impares de la misma
  - Crea una lista nueva con los números pares de la misma
  - Crea una lista con todos números que sea distintos (que no se repitan)
- Dado un string, sustituye las minúsculas por mayúsculas

## 4.3 Estructura try-except-else-finally

En informática existen al menos 3 tipos de errores básicos:

- **Error sintáxtico**: No hemos escrito bien alguna parte del código según lo requiere nuestro lenguaje
- **Error lógico**: A la hora de programar hay algo que se nos pasó por alto y aunque el código esté bien escrito, el resultado que obtendremos no tiene que ser el deseado
- **Error en tiempo de ejecución**: Durante la ejecución de nuestro código, algo inesperado ocurre y nuestro código termina la ejecución de manera forzada

Sea cual sea el tipo de error al que nos enfrentemos, a veces necesitamos que nuestro código siga funcionando.

Un ejemplo sería abrir 2 archivos y que uno de ellos no exista. Si el otro si existe quizás nos interese procesar el que si existe y ya veremos que pasa con el que falta.

Para tener un mayor control sobre qué ocurre en nuestro código, existe la estructura **try-except-else-finally**.

Sintaxis:

- try:
  - code
- except:
  - code
- else:
  - code
- finally:
  - code

No necesariamente tienen que estar las 4 partes presentes en un código. Normalmente se suele escribir más la estructura:

- try:
  - code
- except:
  - code

El significado de cada parte es el siguiente:

- **try**:
  - Encierra el código que queremos ejecutar. Si algo falla, se salta a **except** en caso de que esté escrita esa parte.
- **except**:
  - Captura lo que se conoce como **excepción**.

  - Podemos escribir simplemente **except:** y escribir un código que gestione el error como un mensaje indicando que algo falló o escribir **except Exception as alias**.
  - **except ExceptionType as alias**:
    * Con esta sintaxis estamos indicando que solo queremos controlar la excepción que hemos indicado. Si ocurre otra que no hayamos indicado, el código seguirá fallando y podría pararse la ejecución. -

- **else**:
  - Permite definir una parte de código que se ejecutará en caso de la parte dentro de **try** se haya ejecutado correctamente.
- **finally**:
  - Permite definir una parte de código que sí o sí se ejecutará independientemente de si ocurre algún error o no.

### 4.3.1 Ejemplo 1 | Error descontrolado

[23]:
```
1 + 'a'

'Ejecución sin errores'
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

### 4.3.2 Ejemplo 2 | Manejo de la situación del ejemplo 1

Por normal general no podemos realizar operaciones entre distintos tipos de datos. En el caso de arriba vemos que no podemos sumar un número con una letra.

[24]:
```
try:
    1 + 'a'
except:
    pass

'Ejecución sin errores'
```

[24]:
```
'Ejecución sin errores'
```

### 4.3.3 Ejemplo 3 | Capturar una excepción específica

En este caso vamos a tratar solo de controlar la excepción **FileNotFoundError**. Esta ocurre cuando queremos abrir un archivo pero python no lo encuentra. Es normal que cuando se ejecute este código nos salte un error porque la excepción que ocurre es **TypeError** porque no estamos abriendo archivos, estamos realizando operaciones entre datos de distinto tipo.

[25]:
```
try:
    1 + 'a'
```

```
except FileNotFoundError as e:
    pass

'Ejecución sin errores'
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

### 4.3.4 Ejemplo 4 | Manejo de multiples excepciones de manera específica

[26]:
```
try:
    1 + 'a'
except (FileNotFoundError, TypeError) as e:
    pass

'Ejecución sin errores'
```

[26]:
```
'Ejecución sin errores'
```

### 4.3.5 Ejemplo 5 | Aplicación estructura try-except-else-finally

Caso (a)

[27]:
```
numero_else : int = 1
numero_finally : int = 1

try:
    1 + 'a'
except (FileNotFoundError, TypeError) as e:
    pass
else:
    numero_else = 2
finally:
    numero_finally = -1

'Ejecución sin errores', numero_else, numero_finally
```

[27]:
```
('Ejecución sin errores', 1, -1)
```

Caso (b)

[28]:
```
numero_else : int = 1
numero_finally : int = 1

try:
    1 + 1
except (FileNotFoundError, TypeError) as e:
    pass
```

```
else:
    numero_else = 2
finally:
    numero_finally = -1

'Ejecución sin errores', numero_else, numero_finally
```

[28]: ('Ejecución sin errores', 2, -1)

# 5 Funciones

A lo largo de las tres últimas secciones hemos ido dando la idea de que un programa no siempre tiene que ejecutarse de forma secuencial linea por linea o sentencia por sentencia.

Algunas veces parte del código que hemos escrito no se ejecutará salvo condiciones muy concretas. Otras veces parte del código se ejecutará varias veces antes de seguir con el resto del programa. Y otras veces tendremos una sección de código definida que podríamos necesitar ejecutar varias veces en un programa pero no sabemos cuantas veces ni donde debe de ejecutarse.

Esta última situación da lugar a un nuevo paradigma de la programación. **La programación funcional**.

Una función es una estructura que nos permite definir una sección de código parametrizada e identificada con un nombre. Con esta estructura, simplemente indicando el nombre y los parámetros necesarios, se ejecutará la sección de código que encierra la función allá donde se necesite.

Sintaxis:

- def nombre_funcion([parámetros]):
  - code - return (opcional)

## 5.1 Ejemplo 1 | Función sin parámetros

[1]:
```
numero : int = 1

def incremento() -> int:
    return numero + 1

incremento()
```

[1]:
```
2
```

## 5.2 Ejemplo 2 | Función con un parámetro

[2]:
```
numero : int = 1

def increment(amount : int) -> int:
    return numero + amount

increment(10)
```

[2]:
```
11
```

## 5.3 Ejemplo 3 | Función con varios parámetros

[3]:
```
def suma(a : int, b : int) -> int:
    return a + b
```

```
suma(1, 1)
```

[3]: 2

```
suma(2, 1)
```

[4]: 3

```
suma(3, -1)
```

[5]: 2

### 5.4 Ejemplo 4 | Función con parámetros por defecto

Caso (a) Sintáxis correcta

```
def suma(a : int, b : int = 1) -> int:
    return a + b

suma(1, 1), suma(10)
```

[6]: (2, 11)

Caso (b) Los parámetros por defecto deben ir al final

```
def suma(a : int = 0, b : int) -> int:
    return a + b
```

```
SyntaxError: non-default argument follows default argument
```

### 5.5 Ejemplo 5 | Llamar función indicando qué valor va a qué parámetro

Caso (a) Da igual el orden de asignación si se especifica el nombre del parámetro al que dar un valor

```
def suma(a : int = 0, b : int = 1) -> int:
    return a + b

suma()
```

[8]: 1

```
suma(b = -1, a = 1)
```

[9]: 0

```
suma(b = -10)
```

[10]: -10

[11]:
```
suma(30, -10)
```

[11]: 20

Caso (b) Especificar valor para los parámetros que no tienen valor por defecto

[12]:
```
def suma(a : int, b : int, c : int = 1) -> int:
    return a + b + c

suma(c = -10)
```

```
TypeError: suma() missing 2 required positional arguments: 'a' and 'b'
```

### 5.6 Ejemplo 6 | Programa de geometría

Sin funciones

[13]:
```
punto_1 : dict = {'x' : 1, 'y' : 1}
punto_2 : dict = {'x' : -1, 'y' : 1}
punto_3 : dict = {'x' : 10, 'y' : 11}
punto_4 : dict = {'x' : -21, 'y' : 9}
punto_5 : dict = {'x' : 0, 'y' : 0}

distancia_punto_1_punto_2 : float = ((punto_1['x'] - punto_2['x']) ** 2 +␣
 ↪(punto_1['y'] - punto_2['y']) ** 2) ** 0.5
distancia_punto_1_punto_3 : float = ((punto_1['x'] - punto_3['x']) ** 2 +␣
 ↪(punto_1['y'] - punto_3['y']) ** 2) ** 0.5
distancia_punto_1_punto_4 : float = ((punto_1['x'] - punto_4['x']) ** 2 +␣
 ↪(punto_1['y'] - punto_4['y']) ** 2) ** 0.5
distancia_punto_2_punto_4 : float = ((punto_2['x'] - punto_4['x']) ** 2 +␣
 ↪(punto_2['y'] - punto_4['y']) ** 2) ** 0.5

distancia_punto_1_punto_2, distancia_punto_1_punto_3, distancia_punto_1_punto_4,␣
 ↪distancia_punto_2_punto_4
```

[13]: (2.0, 13.45362404707371, 23.40939982143925, 21.540659228538015)

Con funciones

[14]:
```
def get_point(x : float, y : float) -> dict:
    return {'x' : x, 'y' : y}

def distance(p_1 : dict, p_2 : dict) -> float:
    squared_sum : float = 0

    for coordinate in p_1:
        squared_sum += (p_1[coordinate] - p_2[coordinate]) ** 2
```

```
    return squared_sum ** 0.5

punto_1 : dict = get_point(1, 1)
punto_2 : dict = get_point(-1, 1)
punto_3 : dict = get_point(10, 11)
punto_4 : dict = get_point(-21, 9)
punto_5 : dict = get_point(0, 0)

distancia_punto_1_punto_2 : float = distance(punto_1, punto_2)
distancia_punto_1_punto_3 : float = distance(punto_1, punto_3)
distancia_punto_1_punto_4 : float = distance(punto_1, punto_4)
distancia_punto_2_punto_4 : float = distance(punto_2, punto_4)

distancia_punto_1_punto_2, distancia_punto_1_punto_3, distancia_punto_1_punto_4,␣
↪distancia_punto_2_punto_4
```

[14]: (2.0, 13.45362404707371, 23.40939982143925, 21.540659228538015)

### 5.7 Ejemplo 7 | Función con número indefinido de parámetros

Caso (a)

[15]:
```
def suma(*args) -> int:
    suma : int = 0

    for item in args:
        suma += item

    return suma

suma()
```

[15]: 0

[16]:
```
suma(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
```

[16]: 78

Caso (b) *args no significa pasar una lista sino valores separados por comas

[17]:
```
def suma(*args) -> int:
    suma : int = 0

    for item in args:
        suma += item

    return suma
```

```
suma([1, 2, 3])
```

```
TypeError: unsupported operand type(s) for +=: 'int' and 'list'
```

### 5.8 Ejemplo 8 | Función con número indefinido de parámetros pero cada uno con un nombre

[18]:
```
def foo(**kwargs) -> dict:
    return kwargs

foo(a = 1, b = 2, c = 3, d = 4)
```

[18]:
```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

### 5.9 Extra 1 | Definir 2 funciones con mismo nombre implica que solo la última definición tiene efecto

[19]:
```
def suma(a : int = 0, b : int = 1) -> int:
    return a + b

def suma(a : int = 0, b : int = 1) -> int:
    return a * a + b * b

suma(2, 2)
```

[19]:
```
8
```

### 5.10 Extra 2 | Combinación *arg, **kwargs y parámetros con nombre específicos

[20]:
```
def foo(a : int, *args, **kwargs) -> tuple:
    return a, args, kwargs

foo(1, 2, b = 'a')
```

[20]:
```
(1, (2,), {'b': 'a'})
```

### 5.11 Extra 3 | Funciones anónimas

A veces tendremos funciones cuyo código encapsulado solo ocupa una sentencia de **return expresion**. Es decir, tendremos funciones que hacer un pequeño computo e inmediatamente devuelven el resultado.

Un ejemplo sería la función suma con la que hemos trabajado en muchos ejemplos. Esta recibe una serie de parámetros, los suma y devuelve esa suma.

- def suma(a, b):

- return a + b

Cuando tengamos este tipo de funciones sencillas, podemos directamente hacer uso de lo que se conoce como funciones lambda.

Sintaxis:

- **lambda param1, param2 : code**

Esta función se guarda en una variable, recibe los parametros especificados, realiza el cómputo especificado y devuelve el resultado de manera implícita.

Se usa de la misma forma que las funciones **def**. No deja de ser una funcion solo que con una sintaxis más compacta.

[21]:
```
# Función sin lambda
def suma_sin_lambda(a : int, b : int) -> int:
    return a + b

# Función con lambda
suma_con_lambda = lambda a, b : a + b

suma_sin_lambda(1, 2), suma_con_lambda(1, 2)
```

[21]:
```
(3, 3)
```

## 5.12 Extra 4 | Funciones built-in o funciones disponibles por defecto en cualquier archivo de python

- Built-in functions: https://docs.python.org/es/3.11/library/functions.

Por defecto, tenemos a nuestra disposición un gran variedad de funciones que podemos usar en cualquier momento.

Algunas de las que más se suelen usar son:

- abs(x)
- min(*args)
- max(*args)
- int(object)
- float(object)
- list(object)
- tuple(object)
- set(object)
- dict(object)
- str(object)
- bool(object)

- type(object)
- print(*objects, sep = ' ', ...)
- help(object = None)
- zip(*iterables)
- enumerate(iterable)
- range(start, stop, step = 1)
- len(iterable)

### 5.12.1 abs(), min(), max()

Son operaciones matématicas básicas:

- abs(x) Devuelve el valor absoluto de un númeno
- min(*args) Devuelve el valor mínimo de una serie de números pasados a la función separados por comas
- max(*args) Devuelve el valor máximo de una serie de números pasados a la función separados por comas

[22]:
```
abs(-1)
```

[22]:
```
1
```

[23]:
```
min(0, -1, -30, 20)
```

[23]:
```
-30
```

[24]:
```
max(0, -1, -30, 40)
```

[24]:
```
40
```

### 5.12.2 type(), int(), float(), bool(), str(), list(), tuple(), dict(), set()

Estas funciones nos permiten trabajar con los tipos de datos:

- type(): Devuelve el tipo de dato de una variable, ...
- int() Trata de convertir una variable en entero
- float() Trata de convertir una variable en float
- bool() Trata de convertir una variable en bool
- str() Trata de convertir una variable en string
- list() Trata de convertir una variable en list | Se suele usar con tuplas
- tuple() Trata de convertir una variable en tuple | Se suele usar con listas
- dict() Trata de convertir una variable en dict | Necesario lista de lista. [ [1, 2] ] -> {1 : 2}

- set() Trata de convertir una variable en set | Se suele usar con listas y tuplas

[25]:
```
type(1)
```

[25]:
```
int
```

[26]:
```
type([1,])
```

[26]:
```
list
```

[27]:
```
int(1.634)
```

[27]:
```
1
```

[28]:
```
float(-1)
```

[28]:
```
-1.0
```

[29]:
```
bool([1, 2])
```

[29]:
```
True
```

[30]:
```
str([{1, 3}, 1])
```

[30]:
```
'[{1, 3}, 1]'
```

[31]:
```
list((1, 2))
```

[31]:
```
[1, 2]
```

[32]:
```
tuple([1, 2])
```

[32]:
```
(1, 2)
```

[33]:
```
dict([['a', 2], ['b', 4]])
```

[33]:
```
{'a': 2, 'b': 4}
```

[34]:
```
set([1, 2, 3, 1])
```

[34]:
```
{1, 2, 3}
```

### 5.12.3 print()

No permite mostrar texto por pantalla o guardar texto en archivos

[35]:
```
print(1)
print(1, 2, 3, sep = '|||')
print('aa', end = '----------------------------------')
print([])
```

```
1
1|||2|||3
aa--------------------------------[]
```

### 5.12.4 help()

Normalmente las funciones y otros códigos tienen una documentación. Es decir, una parte del código de la función es una serie de comentarios explicando qué hace esa función. Con help() podemos acceder a esa documentación.

[36]:
```
def foo() -> None:
    """Función estúpida. No hace nada.
    """

help(foo)
```

```
Help on function foo in module __main__:

foo()
    Función estúpida. No hace nada.
```

### 5.12.5 zip()

Algunas veces suele pasar que tenemos 2 o más variables de tipo secuencia (Ej : lista) y necesitamos trabajar cada valor de en paralelo. Es decir, si trabajamos con el primer elemento, queremos trabajar con los primeros elementos de todas las secuencias a la vez y así con el resto de elementos.

[37]:
```
"""
Código que dadas dos listas, genera una con la suma de cada pareja
"""

lista1 : list = [1, 2, 3]
lista2 : list = [2, 3, 4]

suma_lista1_lista2 : list = []

for pareja in zip(lista1, lista2):
    suma_lista1_lista2.append(sum(pareja)) # append es una función de la␣
 ↪variable lista | la veremos en otras secciones

suma_lista1_lista2
```

[37]:
```
[3, 5, 7]
```

### 5.12.6 enumerate()

A veces, cuando trabajamos con secuencias, necesitamos saber cual en qué posición estamos. Es decir, ¿estamos trabajando con el primer elemento, el segundo, . . . ?

La función enumerate(sequence) devuele una pareja (índice, valor)

[38]:
```
lista : list = [1, 2, 3, 4, 5]

for indice, valor in enumerate(lista):
    print('Accediendo al valor', valor, 'en la posición', indice, 'de la lista',␣
 ↪lista)
```

```
Accediendo al valor 1 en la posición 0 de la lista [1, 2, 3, 4, 5]
Accediendo al valor 2 en la posición 1 de la lista [1, 2, 3, 4, 5]
Accediendo al valor 3 en la posición 2 de la lista [1, 2, 3, 4, 5]
Accediendo al valor 4 en la posición 3 de la lista [1, 2, 3, 4, 5]
Accediendo al valor 5 en la posición 4 de la lista [1, 2, 3, 4, 5]
```

[39]:
```
"""
Código que recorre una lista.
Calcula en cuadrado del valor actual y lo pone de nuevo en la lista.
"""

lista : list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

for indice, valor in enumerate(lista):
    lista[indice] = valor ** 2

lista
```

[39]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

### 5.12.7 range()

Hemos visto en muchos ejemplo que las lista numéricas suelen ser de la forma [1, 2, 3, . . . ]. Cuando tenemos una lista de número con un patrón concreto, en vez de crear la lista manualmente, podemos hacer uso de la función range.

[40]:
```
print(list(range(0, 10, 1)))
print(list(range(0, 10, 2)))
print(list(range(-10, 10, 2)))
print(list(range(10, -10, -2)))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]
[10, 8, 6, 4, 2, 0, -2, -4, -6, -8]
```

### 5.12.8 len(iterable)

Devuelve la cantidad de elementos de un dato iterable.

[41]:
```
print( len([1, 2, 3, 4, 5, 6, 7, 8, 9]) )
print( len({'a' : 1, 'b' : -1}) )
```

```
9
2
```

### 5.12.9 Ejercicios

- Crea una función que dada una lista de números enteros como parámetro
  - Devuelva una lista nueva con los números impares de la misma
- Crea una función que dada una lista de números enteros como parámetro
  - Devuelva una lista nueva con los números pares de la misma
- Crea una función que dada una lista de números enteros como parámetro
  - Devuelva una lista con todos números que sea distintos (que no se repitan)
- Crea una función que dada una lista de números enteros como parámetro
  - Devuelva el porcentaje de números que no están repetidos (solo aparecen una vez en la lista)

# 6 Orientación a Objetos

## 6.1 Definición

En informática hay mínimo 4 paradigmas o formas de programación:

- Imperativa: Indicar en cada momento a la máquina que debe hacer
- Procedural: Crear funciones para reducir dimensiónes de código, mejorar la escalabilidad y permitir un código más reusable
- Funcional: Las funciones se definen como si fueran funciones matemáticas. Lo que lleva a una programación generalista más potente
- Orientación a Objectos: Se define el concepto de **clase** como una representación general de algo específico permitiendo diseñar un código más expresivo

### 6.1.1 Programación imperativa

Todo lo que necesitamos lo programamos nosotros y todo lo que ocurre lo hemos programado nosotros.

[1]:
```
numero : int = 5
suma : int = 0
i : int = 1

while i <= numero:
    suma = suma + i
    i = i + 1

suma
```

[1]: 15

[2]:
```
numero : int = 7
suma : int = 0
i : int = 1

while i <= numero:
    suma = suma + i
    i = i + 1

suma
```

[2]: 28

### 6.1.2 Programación procedural

Cuando sea necesario, creamos funciones para no tener que estar a cada rato escribiendo lo mismo. Si lo necesitamos, haremos uso de funciones ya creadas por otras personas como las funciones built-in de python.

[3]:
```python
def calcular_suma(n : int) -> int:
    numeros : list = []

    for i in range(1, n + 1):
        numeros = numeros + [i]

    return sum(numeros)

calcular_suma(5), calcular_suma(7)
```

[3]:
```
(15, 28)
```

### 6.1.3 Programación funcional

En la programación funcional, las funciones se llevan a otro nivel partiendo del concepto matemático de composición de funciones. Además, las funciones pueden recibir otras funciones como parámetros.

Ejemplo:

- $g(x) = x + 1$
- $f(g(x)) = x - 1$
- $g(1) = 1 + 1 = 2$
- $f(g(1)) = g(1) - 1 = 2 - 1 = 1$

En python el ejemplo que hemos estado viendo (dado n, calcula suma 1, n + 1) podríamos calcularla combinado funciones.

[4]:
```python
n : int = 5
suma : int = sum(range(1, n + 1))

n : int = 7
suma1 : int = sum(range(1, n + 1))

def map(function, iterable) -> list:
    return [function(x) for x in iterable]

def power_of_two(x : int) -> int:
    return x ** 2

suma, suma1, map(power_of_two, [1, 2, 3])
```

[4]:
```
(15, 28, [1, 4, 9])
```

### 6.1.4 Programación Orientada a Objectos

Hasta ahora, los tipos de datos que hemos visto son básicamente números, letras, listas, ...; Lo básico.

Sin embargo, hay problemas en donde necesitamos tipos de datos más complejos. Necesitamos un código con más fuerza que nos permita hacer más cosas en pos de resolver el problema al que nos enfrentamos.

La Orientación a Objectos en resumen define 2 conceptos nuevos. **Clase** e **Instancia u objeto**.

El concepto **Clase** es una nueva estructura que tiene como objetivo representar datos y conceptos más complejos. A veces más cercanos a nuestro mundo y otros más cercanos a un formalismo matématico. Depende del problema.

La **representación** se plantea de forma **genérica**. Es decir, si queremos representar un arbol, según el problema necesitaremos más o menos detalle. Una posible representación sería:

- Arbol
  - tiene_hojas : bool
  - color_hojas : Tuple[int, int, int]
  - tipo : perenne | caduco
  - altura_tronco : float
  - color_tronco : Tuple[int, int, int]
  - ...

Si queremos representar una persona en una web. Es decir, un usuario de Facebook por ejemplo, una posible representación sería:

- Usuario
  - nombre : string
  - alias : string
  - edad : int
  - amigos : List[Usuario]
  - posts : List[Post]
  - ...

Estas clases, además de servir para representar algo y de tener datos (**atributos**) que definen ese concepto.
Tambien pueden ser elementos activos, que hagan cosas (**métodos**).

Ejemplos:

- Un usuario de Facebook puede **aceptar amigos**, lo que añade un Usuario a la lista de amigos
- **Eliminar amigos**, lo que elimina un Usuario de la lista de amigos
- Un usuario de Facebook puede **publicar** un **Post**, lo que añade un Post a la lista de posts
- Puede **reaccionar** a **Post**, lo que tiene unas implicaciones ...

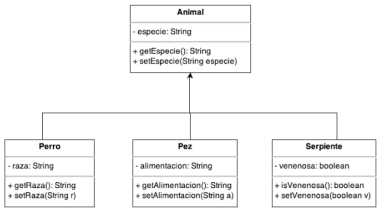
El concepto **Objeto** representa la clase llevada a lo específico.

Por ejemplo, en el caso del Usuario. Un objeto es un Usuario con valores concretos para cada carácterística especificada.

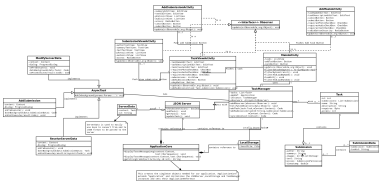
Según el tipo de clase puede haber 1 o varios objetos para dicha clase. Siguiendo con la clase Usuario, un objeto sería un Usuario con nombre Maria, ... y otro objeto sería un Usuario con nombre Pepe, ...

#### 6.1.4.1 Ejemplos conceptuales usando diagramas UML (Unified Modeling Language)

Modelo animales




WTF????



### 6.1.5 ¿Por qué esta chapa?

Porque **todo**, **absolutamente todo** en python **son objectos**.

Lo que hasta ahora hemos visto como los ints, floats, funciones, etc... Son objetos.

Para demostrarlo, introducimos una nueva función built-in **dir(object = None)**.
La lista que vemos muestra los métodos de un objeto.

Int no es solo un número entero

[5]:
```
print(dir( int ))

numero : int = 8
numero.numerator, numero.bit_length()
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio',
```

```
'bit_count', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
'numerator', 'real', 'to_bytes']
```

[5]:
```
(8, 4)
```

List no es solo un contenedor ordenado de datos

[6]:
```
print(dir( list ))

lista : list = []
lista.append(1)
lista.extend([1, 3, 1])
lista
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

[6]:
```
[1, 1, 3, 1]
```

Las funciones también son objetos

[7]:
```
def suma(a : int, b : int = 3) -> int:
    return a + b

print(dir(suma))
print('Nombre: ', suma.__name__, '\nValores por defecto: ', suma.__defaults__,␣
 ↪'\nClase de origen: ', suma.__class__, '\nTipos de datos anotados por␣
 ↪parámetro y retorno: ', suma.__annotations__)
```

```
['__annotations__', '__builtins__', '__call__', '__class__', '__closure__',
'__code__', '__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__globals__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__',
'__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__',
'__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__']
Nombre:  suma
Valores por defecto:  (3,)
Clase de origen:  <class 'function'>
Tipos de datos anotados por parámetro y retorno:  {'a': <class 'int'>, 'b':
<class 'int'>, 'return': <class 'int'>}
```

### 6.1.6 Resumen

- **Clase**: Definición general de un concepto. Modela algo que no existe por defecto en nuestro programa y que lo necesitamos para resolver el problema en cuestión.
  - **Atributos**: Datos representan caracteristicas del concepto a modelar
  - **Operaciones**: Indica qué puede hacer una clase.
    - Ejemplo: Un Animal se puede mover.
  - **Métodos**: Es la implementación de una operación. La operación define textualmente qué puede hacer una clase y el método es el código que hace realidad esa operación.
- **Objeto o Instancia**: Es una clase con valores concretos para los atributos definidos por la misma.

Si tenemos un Objeto y queremos acceder a un atributo o método. La forma es a través de la **notación punto**.

- Ejemplos:
  - Si tenemos un objeto que se llama gaviota1 y tiene un método que se llama volar(), la forma de hacer volar a la gaviota es **gaviota1.volar()**
  - Si tenemos un objeto que se llama satélite2 y tiene un atributo llama tipo_sensor, la forma de saber qué sensor tiene el sensor es **satélite2.sensor**
    - Aunque de normal se debe crear un método que devuelve el atributo a consultar. La forma correcta sería **satélite2.get_sensor()** Pero eso no es parte del curso.

Si alguien quiere profundizar en cómo se programan clases en python aquí dejo una web buena para empezar y en español.

https://j2logo.com/python/tutorial/programacion-orientada-a-objetos/

## 6.2 La Clase String

Como dijimos en la última sección, todo en python en un objeto. En esta sección enumeramos algunos de los métodos estándar de la clase String junto con ejemplos de uso.

Para más información es buena idea saber que existe la documentación oficial de python:

- https://docs.python.org/3/library/stdtypes.html#str

Pero normalmente está mejor usar alguna web para aprender porque simplifica mucho los conceptos de varios lenguajes:

- https://www.w3schools.com/python/python_ref_string.asp

### 6.2.1 Métodos más comunes

Listado de métodos

[8]:
```
for method in dir(str):
    print(method)
```

```
__add__
__class__
__contains__
__delattr__
__dir__
__doc__
__eq__
__format__
__ge__
__getattribute__
__getitem__
__getnewargs__
__gt__
__hash__
__init__
__init_subclass__
__iter__
__le__
__len__
__lt__
__mod__
__mul__
__ne__
__new__
__reduce__
__reduce_ex__
__repr__
__rmod__
__rmul__
__setattr__
```

```
__sizeof__
__str__
__subclasshook__
capitalize
casefold
center
count
encode
endswith
expandtabs
find
format
format_map
index
isalnum
isalpha
isascii
isdecimal
isdigit
isidentifier
islower
isnumeric
isprintable
isspace
istitle
isupper
join
ljust
lower
lstrip
maketrans
partition
removeprefix
removesuffix
replace
rfind
rindex
rjust
rpartition
rsplit
rstrip
split
splitlines
startswith
strip
swapcase
title
translate
```

```
upper
zfill
```

Uso de algunos a métodos

capitalize()

[9]:
```
string_1 : str = 'hola'
string_1.capitalize()
```

[9]:
```
'Hola'
```

count(sub[, start[, end]])

[10]:
```
string_1 : str = 'hola'
string_1.count('a')
```

[10]:
```
1
```

[11]:
```
string_1.count('a', 0, 2)
```

[11]:
```
0
```

endswith(suffix[, start[, end]])

[12]:
```
string_1 : str = 'hola'
string_1.endswith('')
```

[12]:
```
True
```

[13]:
```
string_1.endswith('b')
```

[13]:
```
False
```

startswith(suffix[, start[, end]])

[14]:
```
string_1 : str = 'hola'
string_1.startswith('-')
```

[14]:
```
False
```

[15]:
```
string_1.startswith('h')
```

[15]:
```
True
```

find(sub[, start[, end]])

[16]:
```
string_1 : str = 'hola'
string_1.find('h')
```

[16]:
```
0
```

[17]:
```
string_1.find('a', 1, 2)
```

[17]:
```
-1
```

format(*args, **kwargs)

[18]:
```
string_4 : str = 'Mi nombre es {} y mi edad es {}'
string_4.format('Sergio', 25)
```

[18]:
```
'Mi nombre es Sergio y mi edad es 25'
```

format_map(mapping)

[19]:
```
string_5 : str = 'Mi nombre es {nombre} y mi edad es {edad}'
string_5.format_map({'nombre' : 'Sergio', 'edad' : 25})
```

[19]:
```
'Mi nombre es Sergio y mi edad es 25'
```

lower()

[20]:
```
string_2 : str = 'AAA'
string_2.lower()
```

[20]:
```
'aaa'
```

upper()

[21]:
```
string_1 : str = 'hola'
string_1.upper()
```

[21]:
```
'HOLA'
```

split(sub)

[22]:
```
string_6 : str = 'A,B,C,D,E,F,G,H,I'
string_6.split(',')
```

[22]:
```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

join(iterable)

[23]:
```
string_7 : str = ','
string_7.join(['A', 'B', 'C', 'D', 'E'])
```

[23]:
```
'A,B,C,D,E'
```

Estos métodos tienen una peculiaridad interesante además de útil. No son **inplace**. Es decir, cuando hacemos str.método(), ese método devuelve un string con la modicación. Lo que permite concatenar varias llamadas a funciones en una misma línea sin necesidad de crear variables intermedias o cosas de ese estilo.

[24]:
```
string : str = 'aaaaA_bCsasddede'
string.lower().capitalize().replace('_', '').replace('a', '').replace('s', '')
```

[24]:
```
'Abcddede'
```

### 6.2.2 Consultar la documentación

No olvidemos que existe la función **help(obj)** que nos devuelve el docstring (documentación) de un obj, función, ...

[25]:
```
# Help __add__
help(str.__add__)

print('-' * 50 + '\n')

# Help capitalize
help(str.capitalize)

print('-' * 50 + '\n')

# Help __getitem__
help(str.__getitem__)

print('-' * 50 + '\n')

# Help format
help(str.format)
```

```
Help on wrapper_descriptor:

__add__(self, value, /)
    Return self+value.

--------------------------------------------------

Help on method_descriptor:

capitalize(self, /)
    Return a capitalized version of the string.

    More specifically, make the first character have upper case and the rest
lower
    case.

--------------------------------------------------

Help on wrapper_descriptor:
```

```
__getitem__(self, key, /)
    Return self[key].

------------------------------------------------

Help on method_descriptor:

format(...)
    S.format(*args, **kwargs) -> str

    Return a formatted version of S, using substitutions from args and kwargs.
    The substitutions are identified by braces ('{' and '}').
```

### 6.2.3 Prefijos

f-strings

Son una alternativa a la función format(*arg, **kwargs). Actualmente se recomienda más usar f-strings porque se crearon para hacer un código más potente y a su vez más corto

[26]:
```
variable : int = 1

print( f'{variable}' ) # Mostrar valor de una variable
print( f'{variable=}' ) # Mostrar valor de una variable y el nombre de la␣
 ↪variable
```

```
1
variable=1
```

r raw-strings

Raw (crudo), significa que no que cada caracter se interpreta como lo que es, un caracter. Es decir, la expresión
**n** genera una nueva línea dentro del string,**
t** genera el resultado de pulsar la tecla **tabulador**. Sin embargo, si ponemos una **r** delante del string, lo mencionado arriba no ocurre, cada caracter se interpreta como lo que es.

[27]:
```
print( '\ta' )
```

```
        a
```

[28]:
```
print( r'\ta' )
```

```
\ta
```

[29]:
```
print( '\na' )
```

```

a
```

[30]:
```
print( r'\na' )
```

```
\na
```

### 6.2.4 Ejercicios

- Dada una cadena de strings, calcula la frecuencia de aparación de todos los caracteres.
- Dada una cadena de strings escrita en **snake_case**
  - Transformarla a una cadena escrita en **PascalCase**
  - **snake_case**: hola_mundo, palabra1_palabra2_palabra3_palabra4
  - **PascalCase**: HolaMundo, Palabra1Palabra2Palabra3Palabra4
- Dada una cadena de strings, reemplaza las minúsculas por el símbolo *

## 6.3 La Clase List

Poco a poco vamos afianzando la idea de que en Python todo son objetos.

Lo importante es saber que un objeto tiene métodos y atributos. Los atributos son variables y los métodos son funciones que actuan sobre el objeto. Podemos acceder a los atributos y métodos a través de la notación punto. **objeto.método()**

Bien, tras recordar esto un poco, vamos a explicar cómo se trabajan con las listas y vamos a dar algunos nuevos conceptos sobre el tipo de dato **Secuencia**. Recordemos que los strings, las listas y las tuplas son Secuencias.

### 6.3.1 Métodos de las listas

append(obj)

[37]:
```
lista : list = []
lista.append(1)
lista
```

[37]:
```
[1]
```

count(obj)

[38]:
```
lista2 : list = [1, 2, 3, 4, 1, 1,  1, 2]
lista2.count(1)
```

[38]:
```
4
```

extend(iterable)

[39]:
```
lista : list = []
lista.extend([1, 2, 3])
lista
```

[39]:
```
[1, 2, 3]
```

copy()

[40]:
```
lista : list = []
lista_copia : list = lista

print('ID lista_copia:', id(lista_copia), 'ID lista:', id(lista), '¿Idénticos?',␣
 ↪id(lista_copia) == id(lista))
```

```
ID lista_copia: 2365957555648 ID lista: 2365957555648 ¿Idénticos? True
```

[41]:
```
lista_copia : list = lista.copy()
print('ID lista_copia:', id(lista_copia), 'ID lista:', id(lista), '¿Idénticos?',␣
 ↪id(lista_copia) == id(lista))
```

```
ID lista_copia: 2365958326016 ID lista: 2365957555648 ¿Idénticos? False
```

index(obj)

[42]:
```
lista2 : list = [1, 2, 3, 4, 1, 1,  1, 2]
lista2.index(1)
```

[42]:
```
0
```

[43]:
```
try:
    print(f'{lista2=}.index(obj):', lista2.index(-1))
except ValueError as exception:
    print('Si busco algo que no está en la lista:', f'"{exception}"')
```

```
Si busco algo que no está en la lista: "-1 is not in list"
```

insert(index, object)

[44]:
```
lista : list = []
lista.insert(0, -991)
lista.insert(0, -2)
lista
```

[44]:
```
[-2, -991]
```

pop(index)

[45]:
```
lista : list = [1, 2, 3]
lista.pop(0)
```

[45]:
```
1
```

[46]:
```
lista
```

[46]:
```
[2, 3]
```

[47]:
```
lista : list = [2, 3]
try:
    lista.pop(0)
    print(lista)
    lista.pop(0)
    print(lista)
    lista.pop(0)
except Exception as exception:
    print('Si intento eliminar datos de una lista vacía:', f'"{exception}"')
```

```
[3]
[]
Si intento eliminar datos de una lista vacía: "pop from empty list"
```

remove(obj)

[48]:
```
lista : list = [1, 2, 3]
lista.remove(1)
lista
```

[48]: [2, 3]

[49]:
```
lista : list = [1, 2, 3]
try:
    lista.remove(0)
except Exception as exception:
    print('Si elimino algo que no está en la lista:', f'"{exception}"')
```

```
Si elimino algo que no está en la lista: "list.remove(x): x not in list"
```

reverse()

[50]:
```
lista : list = [1, 2, 3]
lista.reverse()
lista
```

[50]: [3, 2, 1]

sort()

[51]:
```
lista : list = [3, 4, 1, 2, -3]
lista.sort()
lista
```

[51]: [-3, 1, 2, 3, 4]

clear()

[52]:
```
lista : list = [1, 2, 3]
lista.clear()
lista
```

[52]: []

Los métodos de las listas, a diferencia de los strings, de normal son **inplace**. Es decir, objeto.método(params) modifica el objeto dentro del código del método y devuelve None. Esto hace que no podamos concatenar varias funciones. Si queremos hacer varias operaciones debemos escribir una nueva linea por operacion.

[53]:
```
lista : list = []

lista.append(1)
lista.append(0)
lista.remove(0)

# No podemos hacer [].append(1).append(0).remove(0).
```

### 6.3.2 Acceso a los elementos de las secuencias

Acceso a un elemento concreto por índice

[54]:
```
lista : list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(lista[0]) # El primer elemento empieza en el cero
print(lista[1])
print(lista[-1]) # Si queremos acceder desde el último elemento pero no queremo␣
 ↪usar el índice exacto (o no lo conocemos), usamos -1, -2, ...
print(lista[-2]) # Si queremos acceder desde el último elemento pero no queremo␣
 ↪usar el índice exacto (o no lo conocemos), usamos -1, -2, ...
```

```
1
2
9
8
```

### 6.3.3 Slicing

Si tenemos una secuencia, podemos obtener una subsecuencia a usando la nomenclatura: secuencia[start : end : step]

Con esto, obtenemos una subsecuencia a partir de una patrón de índices.

Ejemplo:

- **[0 : 10 : 1]** -> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  El número **end**, en este caso 10, **nunca se incluye** en los índices generados
- **[0 : 10 : 2]** -> 0, 2, 4, 6, 8
  El número **end**, en este caso 10, **nunca se incluye** en los índices generados
- **[0 : None]** -> Devuelve todos los índices, la secuencia completa. end = None equivale a len(secuencia).
- **[:]** -> Devuelve todos los índices. Es una copia de la lista. Lo mismo que hacer lista.copy()
- **[::]** -> Devuelve todos los índices. Es una copia de la lista. Lo mismo que hacer lista.copy()
- **[::-1]** -> Devuelve la lista invertida. Ej [0, 1, 2][::-1] -> [2, 1, 0]

[55]:
```
lista : list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(lista[0 : 10 : 1])
print(lista[0 : 10 : 2])
print(lista[0 : None], '------', lista[0 : len(lista)])
print(lista[0 : -1]) # Si queremos incluir el último elemento usar None, nunca -1
print(lista[:]) # Si queremos incluir el último elemento usar None, nunca -1
print(lista[::]) # Si queremos incluir el último elemento usar None, nunca -1
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ------ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

### 6.3.4 Modificar un elemento

Depende del dato que almacena la secuencia. Si tenemos una lista de enteros, podemos sumar, restar, ... Si tenemos una lista de listas, podemos aplicar métodos de listas, ...

[56]:
```
lista_enteros : list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

lista_enteros[0] = -1 # Asignar un dato
lista_enteros[0] *= 90 # Hace una operacion y asignar
lista_enteros[1] * 0 # Si no ponemos el operacion de asignación, el dato de la␣
↪lista no se modifica || Al menos en este caso que el dato es inmmutable.
lista_enteros
```

[56]:
```
[-90, 2, 3, 4, 5, 6, 7, 8, 9]
```

[57]:
```
lista_listas : list = [[], []]

lista_listas[0].append(1)
lista_listas # En este caso al ser mutable, si un método modifica el dato en␣
↪cuestión pos listo, no es necesario usar el operador de asignación.
```

[57]:
```
[[1], []]
```

### 6.3.5 Consejos

#### No modificar e iterar sobre la misma lista

Si tenemos una lista y queremos eliminar o añadir elementos a la vez que recorremos dicha lista, lo correcto es una de dos:

- Recorrer una copia y modificar la lista original
- Recorrer la lista original y modificar la copia

[58]:
```
lista : list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

for i in lista:
    if i < 5:
        lista.remove(i)

lista
```

[58]:
```
[2, 4, 5, 6, 7, 8, 9]
```

[59]:
```
lista : list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

for i in lista[:]:
    if i < 5:
        lista.remove(i)

lista
```

[59]:
```
[5, 6, 7, 8, 9]
```

[60]:
```
lista : list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

for i in lista.copy():
    if i < 5:
        lista.remove(i)

lista
```

[60]:
```
[5, 6, 7, 8, 9]
```

**Copiar las listas en vez de asignar tal cual**

[61]:
```
lista1 : list = [1, 2, 3]
lista2 : list = lista1

lista2.clear()

lista1
```

[61]:
```
[]
```

[62]:
```
lista1 : list = [1, 2, 3]
lista2 : list = lista1.copy()

lista2.clear()

lista1
```

[62]:
```
[1, 2, 3]
```

### 6.3.6 Ejercicios

- Dada una lista, elimina los elementos repetidos
- Dada la lista [1, 2, 'a', [1, 2]], calcula el índice o posición del elemento 'a'
- Dada una lista, ordénala de mayor a menor
- Dada una lista, obtén los elementos en las posiciones impares y multiplicalos por 2 (cuidado con posibles excepciones)
- Investigar Shallow Copy vs Deep Copy en python y probad a copiar una lista de listas o lista de diccionarios con listas como valores

## 6.4 La Clase Tupla

La clase tupla tiene en algunos sentidos un comportamiento similar a la lista:

- Podemos acceder a elementos sueltos
- Podemos hacer slicing y obtener subsecuencias

Pero, la gran diferencia es que la tupla es inmutable, lo que significa que si queremos añadir o quitar elementos, tenemos que crear una nueva tupla. Si queremos modificar elementos, que ya existen en la tupla, no podemos hacerlo a no ser que algunos se sus datos sean mutables.

### 6.4.1 Métodos de las tuplas

count(obj)

[63]:
```
tupla : tuple = (1, 2, 3, 4, 1, 1,  1, 2)
tupla.count(1)
```

[63]: 4

index(obj)

[64]:
```
tupla : tuple = (1, 2, 3, 4, 1, 1,  1, 2)
tupla.index(1)
```

[64]: 0

[65]:
```
tupla : tuple = (1, 2, 3, 4, 1, 1,  1, 2)

try:
    print(tupla.index(-1))
except Exception as exception:
    print('Si busco algo que no está en la lista:', f'"{exception}"')
```

```
Si busco algo que no está en la lista: "tuple.index(x): x not in tuple"
```

### 6.4.2 Modificar elementos mutables pero no inmutables

[66]:
```
tupla = ([], 1, 2)

try:
    tupla[1] = 2
    print(tupla)
except Exception as exception:
    print(exception)

tupla[0].append(1)

tupla # Cambió la tupla
```

```
'tuple' object does not support item assignment
```

```
[66]: ([1], 1, 2)
```

Lo que la tupla indica es que los elementos de la misma no pueden cambiar de id (digamos posición de memoria).

En otras palabras, la tupla ([], 1, 2) tiene 3 elementos. Cada elemento tiene un id. La tupla garantiza que los ids, no cambian. O lo que es lo mismo, el dato no puede ser modificado por otro porque 2 datos distintos tienen ids distintos. Se guardan en diferentes zonas de memoria.

Adémas, el error **'tuple' object does not support item assignment** se debe a que la clase Tupla no tiene el método **setitem** que permite hacer **lista[key] = value**.

## 6.5 La Clase Set

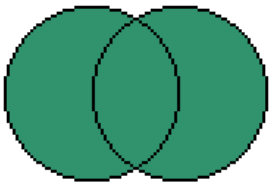
La clase Set es una representación directa del concepto **conjunto** pertenience a las matemáticas.

Un conjunto tiene como caracterítica principal que ningún elemento del conjunto se repite. Además, en python la clase conjunto tiene la propiedad de estar ordenado. Al menos, si los elementos tienen en su código especificada alguna relación de orden.

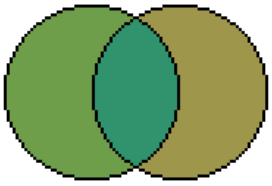
**Nota:** Los conjuntos solo sirven para contener datos, recorrerlos y garantizar que los datos no se repiten.
Los elementos no están indexados por lo que no podemos acceder usando la sintaxis set[key].

### 6.5.1 Representación de las operaciones básicas de conjuntos

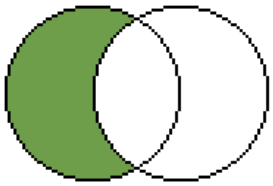
#### 6.5.1.1 Unión $A \cup B$



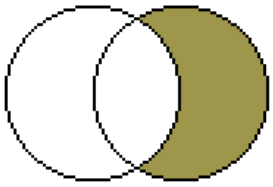
#### 6.5.1.2 Intersección $A \cap B$



#### 6.5.1.3 Diferencia Asimétrica $A \setminus B$

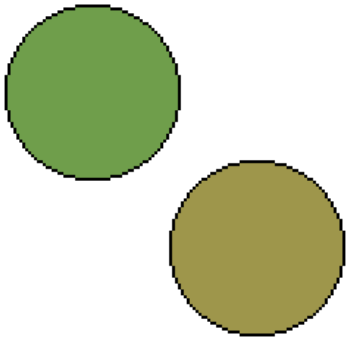


#### 6.5.1.4 Diferencia Asimétrica $B \setminus A$

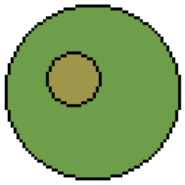


#### 6.5.1.5 Diferencia Simétrica $A \Delta B = B \Delta A$

#### 6.5.1.6 Conjuntos disjuntos $A \cap B = \emptyset$



#### 6.5.1.7 Subconjuntos



### 6.5.2 Métodos de los conjuntos

add(obj)

[67]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.add(1)
conjunto.add(23)

conjunto
```

[67]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 23}
```

clear()

[68]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
conjunto.clear()
conjunto
```

[68]:
```
set()
```

copy()

[69]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
conjunto_1 : set = conjunto
print('ID conjunto_1:', id(conjunto_1), 'ID conjunto:', id(conjunto),␣
 ↪'¿Idénticos?', id(conjunto_1) == id(conjunto))
```

```
ID conjunto_1: 2365958118912 ID conjunto: 2365958118912 ¿Idénticos? True
```

[70]:
```
conjunto_1 : set = conjunto.copy()
print('ID conjunto_1:', id(conjunto_1), 'ID conjunto:', id(conjunto),␣
 ↪'¿Idénticos?', id(conjunto_1) == id(conjunto))
```

```
ID conjunto_1: 2365958118688 ID conjunto: 2365958118912 ¿Idénticos? False
```

difference_update()

[71]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.difference_update({1, 2, 4})

conjunto
```

[71]:
```
{3, 5, 6, 7, 8, 9, 10, 11}
```

difference()

[72]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.difference({1, 2, 4})
```

[72]: {3, 5, 6, 7, 8, 9, 10, 11}

intersection_update()

[73]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.intersection_update({1, 2, 4})

conjunto
```

[73]: {1, 2, 4}

intersection()

[74]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.intersection({1, 2, 4})
```

[74]: {1, 2, 4}

symmetric_difference_update()

[75]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.symmetric_difference_update({1, 2, 4})

conjunto
```

[75]: {3, 5, 6, 7, 8, 9, 10, 11}

symmetric_difference()

[76]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.symmetric_difference({1, 2, 4})
```

[76]: {3, 5, 6, 7, 8, 9, 10, 11}

update()

[77]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.update({1, 2, 4, 34, -90})

conjunto
```

[77]: {-90, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 34}

union()

[78]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.union({1, 2, 4, 34, -90})
```

[78]:
```
{-90, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 34}
```

isdisjoint()

[79]:
```
conjunto_1 : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
conjunto_2 : set = {-9, -10, -11, -12, -13, -14, -15}
conjunto_3 : set = {0, 1, 3}

print(f'{conjunto_1=}, {conjunto_2=}')
print(f'¿isdisjoint? {conjunto_1.isdisjoint(conjunto_2)} || Intersection␣
 ↪{conjunto_1.intersection(conjunto_2)}')
```

```
conjunto_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, conjunto_2={-15, -14, -13, -12,
-11, -10, -9}
¿isdisjoint? True || Intersection set()
```

[80]:
```
print(f'{conjunto_1=}, {conjunto_3=}')
print(f'¿isdisjoint? {conjunto_1.isdisjoint(conjunto_3)} || Intersection␣
 ↪{conjunto_1.intersection(conjunto_3)}')
```

```
conjunto_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, conjunto_3={0, 1, 3}
¿isdisjoint? False || Intersection {1, 3}
```

issubset()

[81]:
```
conjunto_1 : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
conjunto_2 : set = {1, 3, 2, 5}

print(f'{conjunto_1=}, {conjunto_2=}')
print(f'¿conjunto_1 issubset conjunto_2? {conjunto_1.issubset(conjunto_2)}')
```

```
conjunto_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, conjunto_2={1, 2, 3, 5}
¿conjunto_1 issubset conjunto_2? False
```

[82]:
```
print(f'{conjunto_1=}, {conjunto_2=}')
print(f'¿conjunto_2 issubset conjunto_1? {conjunto_2.issubset(conjunto_1)}')
```

```
conjunto_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, conjunto_2={1, 2, 3, 5}
¿conjunto_2 issubset conjunto_1? True
```

issuperset()

[83]:
```
conjunto_1 : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
conjunto_2 : set = {1, 3, 2, 5}

print(f'{conjunto_1=}, {conjunto_2=}')
print(f'¿conjunto_1 issuperset conjunto_2? {conjunto_1.issuperset(conjunto_2)}')
```

```
conjunto_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, conjunto_2={1, 2, 3, 5}
¿conjunto_1 issuperset conjunto_2? True
```

[84]:
```
print(f'{conjunto_1=}, {conjunto_2=}')
print(f'¿conjunto_2 issuperset conjunto_1? {conjunto_2.issuperset(conjunto_1)}')
```

```
conjunto_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, conjunto_2={1, 2, 3, 5}
¿conjunto_2 issuperset conjunto_1? False
```

discard()

[85]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

conjunto.discard(1)
conjunto.discard(0)

conjunto
```

[85]:
```
{2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
```

remove()

[86]:
```
conjunto : set = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

print('PRE remove():', conjunto)
try:
    conjunto.remove(1)
    conjunto.remove(0)
except Exception as exception:
    print('remove(obj) y discard(obj) son lo mismo, pero remove falla si no␣
 ↪existe el elemento a eliminar')
```

```
PRE remove(): {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
remove(obj) y discard(obj) son lo mismo, pero remove falla si no existe el
elemento a eliminar
```

pop()

[87]:
```
conjunto : set = {1, 2}

try:
    conjunto.pop()
    conjunto.pop()
    conjunto.pop()
except Exception as exception:
    print('Si hacemos pop() sobre un conjunto vacio falla')
```

```
Si hacemos pop() sobre un conjunto vacio falla
```

### 6.5.3 Ejemplos de uso

Eliminar elementos repetidos de una lista

[88]:
```
lista : list = [1, 2, 3, 4, 1, 1, 3, 5, 4, 4]

lista = list(set(lista))
lista
```

[88]:
```
[1, 2, 3, 4, 5]
```

### 6.5.4 Errores típicos

No se puede acceder a un elemento según un índice

[89]:
```
conjunto : set = {1, 2, 3, 4, 5, 6}

try:
    conjunto[0]
except Exception as exception:
    print(exception)

print('La clase Set es accesible por índice?', '__getitem__' in dir(set))
print('La clase List es accesible por índice?', '__getitem__' in dir(list))
print('La clase Tuple es accesible por índice?', '__getitem__' in dir(tuple))
print('La clase Str es accesible por índice?', '__getitem__' in dir(str))
print('La clase Dict es accesible por índice?', '__getitem__' in dir(dict))
```

```
'set' object is not subscriptable
La clase Set es accesible por índice? False
La clase List es accesible por índice? True
La clase Tuple es accesible por índice? True
La clase Str es accesible por índice? True
La clase Dict es accesible por índice? True
```

### 6.5.5 Ejercicios

- Dada una lista, obtén los elementos únicos
- Dados dos sets cualquiera de números A y B, obtén aquellos elementos que están en A pero no en B y los elementos que están en B pero no es A

## 6.6 La Clase Dict

La clase Dict podríamos entenderla como una lista cuyos índices no tienen qué ser únicamente números indicando el orden de inserción de una dato en la lista.

En un diccionario identificamos 2 elementos.
**Clave** y **Valor**.
La clave es el índice con el que accedemos a un valor que dicha clave tiene asociado.

Las claves se almacenan en conjuntos, lo que hace que no puedan repetirse las claves.
No pueden haber 2 claves iguales en un diccionario.
Las claves deben ser inmutables: strings, tuplas, enteros, decimales.
Nunca listas. Al menos de forma explícita.

Cada clave puede tener cualquier valor asociado. Es decir, en un mismo diccionario, una clave puede tener asociado un numero y otra clave tener asociada una palabra.

### 6.6.1 Métodos de los conjuntos

clear()

[90]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

diccionario.clear()

diccionario
```

[90]:
```
{}
```

copy()

[91]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

diccionario_1 : dict= diccionario
print('ID diccionario_1:', id(diccionario_1), 'ID diccionario:',␣
 ↪id(diccionario), '¿Idénticos?', id(diccionario_1) == id(diccionario))
```

```
ID diccionario_1: 2365960996672 ID diccionario: 2365960996672 ¿Idénticos? True
```

[92]:
```
diccionario_1 : dict = diccionario.copy()
print('ID diccionario_1:', id(diccionario_1), 'ID diccionario:',␣
 ↪id(diccionario), '¿Idénticos?', id(diccionario_1) == id(diccionario))
```

```
ID diccionario_1: 2365957433472 ID diccionario: 2365960996672 ¿Idénticos? False
```

fromkeys(iterable, value=None)

[93]:
```
{}.fromkeys([1, 2, 3], 'a')
```

[93]:
```
{1: 'a', 2: 'a', 3: 'a'}
```

get(key, default=None)

[94]:
```
{}.get(1, 'a')
```

[94]:
```
'a'
```

[95]:
```
{1 : 'b', 2 : 'aaa'}.get(1, 'a')
```

[95]:
```
'b'
```

keys()

[96]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}
list(diccionario.keys())
```

[96]:
```
['a', 'b', 'c', 'd']
```

values()

[97]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}
list(diccionario.values())
```

[97]:
```
[1, 2, 3, 4]
```

items()

[98]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}
list(diccionario.items())
```

[98]:
```
[('a', 1), ('b', 2), ('c', 3), ('d', 4)]
```

pop()

[99]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

try:
    print(diccionario.pop('a'))
    print(diccionario.pop(11))
except KeyError as exception:
    print('ERROR:', exception)

diccionario
```

```
1
ERROR: 11
```

[99]:
```
{'b': 2, 'c': 3, 'd': 4}
```

popitem()

[100]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

try:
```

```
    print(diccionario.popitem())
    print(diccionario.popitem())
    print(diccionario.popitem())
    print(diccionario.popitem())
    print(diccionario.popitem())
except Exception as exception:
    print('ERROR:', exception)
```

```
('d', 4)
('c', 3)
('b', 2)
('a', 1)
ERROR: 'popitem(): dictionary is empty'
```

update()

[101]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

diccionario.update({'a' : 2, 'b' : (1, 2), 'e' : -1})

diccionario
```

[101]:
```
{'a': 2, 'b': (1, 2), 'c': 3, 'd': 4, 'e': -1}
```

setdefault(key, default=None)

[102]:
```
diccionario : dict = {'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

try:
    print(diccionario.setdefault(1))
    print(diccionario.setdefault('a', -1))
    print(diccionario.setdefault(22, -1222))
except Exception as exception:
    print('ERROR:', exception)
```

```
None
1
-1222
```

### 6.6.2 Ejemplos de Uso

#### 6.6.2.1 Acceder a un elemento según clave

[103]:
```
diccionario : set = {1 : (1, 2), 2 : (), 3 : (), 4 : (2, 3, 4)}
diccionario[1]
```

[103]:
```
(1, 2)
```

Comprobad que existe la clave o usar .get() para que no salte una Excepción

[104]:
```
diccionario : set = {1 : (1, 2), 2 : (), 3 : (), 4 : (2, 3, 4)}
diccionario[-1]
```

```
KeyError: -1
```

#### 6.6.2.2 Borrar de un diccionario claves que tienen tuplas vacías Versión mala

[105]:
```
diccionario : set = {1 : (1, 2), 2 : (), 3 : (), 4 : (2, 3, 4)}

for key, value in diccionario.items():
    if not value:
        diccionario.pop(key)

diccionario
```

```
RuntimeError: dictionary changed size during iteration
```

Versión buena

[106]:
```
diccionario : set = {1 : (1, 2), 2 : (), 3 : (), 4 : (2, 3, 4)}

for key, value in diccionario.copy().items():
    if not value:
        diccionario.pop(key)

diccionario
```

[106]:
```
{1: (1, 2), 4: (2, 3, 4)}
```

### 6.6.3 Ejercicios

- Crea un diccionario de animales a tu gusto con la información que estimes necesaria para definir a un animal
- Añade animales al diccionario anterior
- Eliminar animales del diccionario anterior
- Crea un diccionario nuevo y trata de combinar los dos diccionarios

# 7 Nombres, Espacios de nombres, Módulos, Librerías y Paquetes

- Un **nombre** es el **identificativo** que le damos a un **objeto** (variable, función, ...).
- Un **espacio de nombres** es una **colección de nombres** que referencian a objetos.
  - Ej: operaciones built-in = {print(), len(), ...}
- Un **módulo** es un **archivo .py** que **contiene nombres**.
- Una **librería** es un **conjunto de módulos**.
- Un **paquete** en un **conjunto de librerías**

## 7.1 Alcance de nombres (namespaces)

Es el lugar o espacio dentro de un programa en el cual un nombre (variable, clase, etc.) es válido.

En python hay 3 alcances:

- **Alcance local:** nombres definidos dentro de una función o método
- **Alcance de módulo:** nombres definidos dentro de un archivo
- **Alcance interno:** nombres definidos dentro de Python (están siempre disponibles)

[1]:
```
# variables con alcance de módulo
W = 5
Y = 3

#los parámetros son como variables de la función
#por eso X tiene alcance local
def spam(X):

    #decirle a la función que busque en el nivel de módulo y que no cree su␣
 ↪propia variable W
    global W

    Z = X*2 # crea nueva variable local Z
    W = X+5 # usa la variable W del módulo (global)

    if Z > W:
        # print pertenece al alcance interno de Python
        print( f"2 x {X} es mayor que {X} + 5")
        return Z
    else:
        print( f"2 x {X} es menor o igual que {X} + 5")
        return Y # no exite variable Y local, por lo que usa la variable Y del␣
 ↪módulo

print(f'{W=}, {Y=}')
spam(1)
print(f'{W=}, {Y=}')
spam(20)
print(f'{W=}, {Y=}')
```

```
W=5, Y=3
2 x 1 es menor o igual que 1 + 5
W=6, Y=3
2 x 20 es mayor que 20 + 5
W=25, Y=3
```

### 7.1.1 Ejemplo con bucles

[2]:
```
x = 0

for x in range(10):
    pass

print(x)
```

```
9
```

## 7.2 Uso de módulos

Vamos a introducir algunos de los módulos estándar o que vienen por defecto en Python.

En secciones siguientes veremos algunos módulos típicamente usados para computación con grandes cantidades de datos, ciencia, ...

Aprovechando que en la próxima sección veremos como trabajar con archivos, abrir un archivo, escribir y demás, vamos a introducir brevemente cómo se importa el módulo **os** para poder usarlo en nuestro código.

Para importar un módulo, ya sea estándar o propio (recordemos cualquier archivo .py ya es un módulo) podemos escribir cualquiera de las siguientes opciones.

## 7.3 import module_name | import module_name as alias

Para importar un módulo usamos la palabra **import**.

Cuando importamos un módulo o un nombre contenido en ese módulo, podemos asignar cada elemento importado un alias.
De esta forma evitamos problemas si queremos importar un mismo nombre contenido en varios módulos, podemos evitar conflictos, etc...

[3]:
```
import os
import os as alias
```

## 7.4 from module_name import name | from module_name import name as alias

Para importar nombres de un módulo, usamos la palabra **from**.
Esto importa aquellos nombres que hemos especificado pero no el módulo en sí.

[4]:
```
from os import path
from os import path as ruta
from os import path, __name__ as module_name
from os.path import basename

__name__, module_name
```

[4]:
```
('__main__', 'os')
```

### 7.5 from module_name import *

Esta sentencia import todos los nombres de una módulo.
Pero, no es buena práctica porque un módulo puede ser muy largo (pesado) y es importante saber qué tenemos a nuestra disposición y qué estamos usando.
Si no estamos usando un módulo, es mejor no importarlo.

[5]:
```
# Aplicar dir() nos devuelve todos los nombres que hay en un archivo
dir()
```

[5]:
```
['In',
 'Out',
 'W',
 'Y',
 '_',
 '_4',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '__vsc_ipynb_file__',
 '_dh',
 '_i',
 '_i1',
 '_i2',
 '_i3',
 '_i4',
 '_i5',
 '_ih',
 '_ii',
 '_iii',
 '_oh',
 'alias',
 'basename',
```

```
 'exit',
 'get_ipython',
 'module_name',
 'open',
 'os',
 'path',
 'quit',
 'ruta',
 'spam',
 'x']
```

[6]:
```
from os import *
dir()
```

[6]:
```
['DirEntry',
 'F_OK',
 'In',
 'O_APPEND',
 'O_BINARY',
 'O_CREAT',
 'O_EXCL',
 'O_NOINHERIT',
 'O_RANDOM',
 'O_RDONLY',
 'O_RDWR',
 'O_SEQUENTIAL',
 'O_SHORT_LIVED',
 'O_TEMPORARY',
 'O_TEXT',
 'O_TRUNC',
 'O_WRONLY',
 'Out',
 'P_DETACH',
 'P_NOWAIT',
 'P_NOWAITO',
 'P_OVERLAY',
 'P_WAIT',
 'R_OK',
 'SEEK_CUR',
 'SEEK_END',
 'SEEK_SET',
 'TMP_MAX',
 'W',
 'W_OK',
 'X_OK',
 'Y',
 '_',
```

```
'_4',
'_5',
'__',
'___',
'__builtin__',
'__builtins__',
'__doc__',
'__loader__',
'__name__',
'__package__',
'__spec__',
'__vsc_ipynb_file__',
'_dh',
'_exit',
'_i',
'_i1',
'_i2',
'_i3',
'_i4',
'_i5',
'_i6',
'_ih',
'_ii',
'_iii',
'_oh',
'abort',
'access',
'alias',
'altsep',
'basename',
'chdir',
'chmod',
'close',
'closerange',
'cpu_count',
'curdir',
'defpath',
'device_encoding',
'devnull',
'dup',
'dup2',
'environ',
'error',
'execl',
'execle',
'execlp',
'execlpe',
```

```
'execv',
'execve',
'execvp',
'execvpe',
'exit',
'extsep',
'fdopen',
'fsdecode',
'fsencode',
'fspath',
'fstat',
'fsync',
'ftruncate',
'get_exec_path',
'get_handle_inheritable',
'get_inheritable',
'get_ipython',
'get_terminal_size',
'getcwd',
'getcwdb',
'getenv',
'getlogin',
'getpid',
'getppid',
'isatty',
'kill',
'linesep',
'link',
'listdir',
'lseek',
'lstat',
'makedirs',
'mkdir',
'module_name',
'name',
'open',
'os',
'pardir',
'path',
'pathsep',
'pipe',
'popen',
'putenv',
'quit',
'read',
'readlink',
'remove',
```

```
'removedirs',
'rename',
'renames',
'replace',
'rmdir',
'ruta',
'scandir',
'sep',
'set_handle_inheritable',
'set_inheritable',
'spam',
'spawnl',
'spawnle',
'spawnv',
'spawnve',
'startfile',
'stat',
'stat_result',
'statvfs_result',
'strerror',
'supports_bytes_environ',
'symlink',
'system',
'terminal_size',
'times',
'times_result',
'truncate',
'umask',
'uname_result',
'unlink',
'unsetenv',
'urandom',
'utime',
'waitpid',
'waitstatus_to_exitcode',
'walk',
'write',
'x']
```

# 8 Manejo de archivos externos

Python tiene una función built-in llamada **open** que nos permite abrir archivo como .txt, .csv, .xlsx, ...

Sin embargo, es cierto, que según el tipo de archivo, python tiene módulos específicos para manejar más cómodamente cada archivo.

Ejemplos:

- csv para archivos csv (Comma separated values)
- json para archivos json (JavaScript object notation)

Daremos algunos ejemplos para que todo se entienda mejor.

## 8.1 Acceso a la documentación

[1]:
```
help(open)
```

```
Help on function open in module io:

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None,
closefd=True, opener=None)
    Open file and return a stream.  Raise OSError upon failure.

    file is either a text or byte string giving the name (and the path
    if the file isn't in the current working directory) of the file to
    be opened or an integer file descriptor of the file to be
    wrapped. (If a file descriptor is given, it is closed when the
    returned I/O object is closed, unless closefd is set to False.)

    mode is an optional string that specifies the mode in which the file
    is opened. It defaults to 'r' which means open for reading in text
    mode.  Other common values are 'w' for writing (truncating the file if
    it already exists), 'x' for creating and writing to a new file, and
    'a' for appending (which on some Unix systems, means that all writes
    append to the end of the file regardless of the current seek position).
    In text mode, if encoding is not specified the encoding used is platform
    dependent: locale.getpreferredencoding(False) is called to get the
    current locale encoding. (For reading and writing raw bytes use binary
    mode and leave encoding unspecified.) The available modes are:

    ========= ========================================================
    Character Meaning
    --------- --------------------------------------------------------
    'r'       open for reading (default)
    'w'       open for writing, truncating the file first
    'x'       create a new file and open it for writing
    'a'       open for writing, appending to the end of the file if it exists
    'b'       binary mode
```

```
't'       text mode (default)
'+'       open a disk file for updating (reading and writing)
'U'       universal newline mode (deprecated)
========= ===================================================================

The default mode is 'rt' (open for reading text). For binary random
access, the mode 'w+b' opens and truncates the file to 0 bytes, while
'r+b' opens the file without truncation. The 'x' mode implies 'w' and
raises an `FileExistsError` if the file already exists.

Python distinguishes between files opened in binary and text modes,
even when the underlying operating system doesn't. Files opened in
binary mode (appending 'b' to the mode argument) return contents as
bytes objects without any decoding. In text mode (the default, or when
't' is appended to the mode argument), the contents of the file are
returned as strings, the bytes having been first decoded using a
platform-dependent encoding or using the specified encoding if given.

'U' mode is deprecated and will raise an exception in future versions
of Python.  It has no effect in Python 3.  Use newline to control
universal newlines mode.

buffering is an optional integer used to set the buffering policy.
Pass 0 to switch buffering off (only allowed in binary mode), 1 to select
line buffering (only usable in text mode), and an integer > 1 to indicate
the size of a fixed-size chunk buffer.  When no buffering argument is
given, the default buffering policy works as follows:

* Binary files are buffered in fixed-size chunks; the size of the buffer
  is chosen using a heuristic trying to determine the underlying device's
  "block size" and falling back on `io.DEFAULT_BUFFER_SIZE`.
  On many systems, the buffer will typically be 4096 or 8192 bytes long.

* "Interactive" text files (files for which isatty() returns True)
  use line buffering.  Other text files use the policy described above
  for binary files.

encoding is the name of the encoding used to decode or encode the
file. This should only be used in text mode. The default encoding is
platform dependent, but any encoding supported by Python can be
passed.  See the codecs module for the list of supported encodings.

errors is an optional string that specifies how encoding errors are to
be handled---this argument should not be used in binary mode. Pass
'strict' to raise a ValueError exception if there is an encoding error
(the default of None has the same effect), or pass 'ignore' to ignore
errors. (Note that ignoring encoding errors can lead to data loss.)
See the documentation for codecs.register or run 'help(codecs.Codec)'
```

```
for a list of the permitted encoding error strings.

newline controls how universal newlines works (it only applies to text
mode). It can be None, '', '\n', '\r', and '\r\n'.  It works as
follows:

* On input, if newline is None, universal newlines mode is
  enabled. Lines in the input can end in '\n', '\r', or '\r\n', and
  these are translated into '\n' before being returned to the
  caller. If it is '', universal newline mode is enabled, but line
  endings are returned to the caller untranslated. If it has any of
  the other legal values, input lines are only terminated by the given
  string, and the line ending is returned to the caller untranslated.

* On output, if newline is None, any '\n' characters written are
  translated to the system default line separator, os.linesep. If
  newline is '' or '\n', no translation takes place. If newline is any
  of the other legal values, any '\n' characters written are translated
  to the given string.

If closefd is False, the underlying file descriptor will be kept open
when the file is closed. This does not work when a file name is given
and must be True in that case.

A custom opener can be used by passing a callable as *opener*. The
underlying file descriptor for the file object is then obtained by
calling *opener* with (*file*, *flags*). *opener* must return an open
file descriptor (passing os.open as *opener* results in functionality
similar to passing None).

open() returns a file object whose type depends on the mode, and
through which the standard file operations such as reading and writing
are performed. When open() is used to open a file in a text mode ('w',
'r', 'wt', 'rt', etc.), it returns a TextIOWrapper. When used to open
a file in a binary mode, the returned class varies: in read binary
mode, it returns a BufferedReader; in write binary and append binary
modes, it returns a BufferedWriter, and in read/write mode, it returns
a BufferedRandom.

It is also possible to use a string or bytearray as a file for both
reading and writing. For strings StringIO can be used like a file
opened in a text mode, and for bytes a BytesIO can be used like a file
opened in a binary mode.
```

## 8.2 Lectura/Escritura clásica

### 8.2.1 Lectura

[2]:
```
filename = r'C:\Users\sergi\Documents\repos\python_course\data\Dummy.txt' # Es␣
 ↪buena idea usar r delante de los paths explícitos cuando estamos en Windows.

txt_file = open(filename, mode = 'r')
content = txt_file.read()

txt_file = open(filename, mode = 'r')
content_first_line = txt_file.readline()

txt_file = open(filename, mode = 'r')
content_first_second_lines = txt_file.readlines(19) # Leemos hasta dónde se␣
 ↪supera el nº de caracteres | hint=-1 lee todo

txt_file.close() # Cuando no se usa se cierra explícitamente

print(f'{content=} \n{content_first_line=} \n{content_first_second_lines=}')
```

```
content='AAAAAAAAAAAAAAAAAA\nBBBBBBBBBBBBBBBBBB\nCCCCCCCCCCCCCCCCCCC\nDDDDDDDDDDD
DDDDDDD\n'
content_first_line='AAAAAAAAAAAAAAAAAA\n'
content_first_second_lines=['AAAAAAAAAAAAAAAAAA\n', 'BBBBBBBBBBBBBBBBBB\n']
```

### 8.2.2 Lectura y Escritura

[3]:
```
filename = r'C:\Users\sergi\Documents\repos\python_course\data\Dummy.txt' # Es␣
 ↪buena idea usar r delante de los paths explícitos cuando estamos en Windows.

txt_file = open(filename, mode = 'r+') # Leer y escribir
txt_file.write(txt_file.read() + '\n')
txt_file.write('E' * 18 + '\n')
txt_file.close()
```

### 8.2.3 Append (Si ya existe, escribe al final)

[4]:
```
filename = r'C:\Users\sergi\Documents\repos\python_course\data\Dummy.txt' # Es␣
 ↪buena idea usar r delante de los paths explícitos cuando estamos en Windows.

txt_file = open(filename, mode = 'a')
txt_file.write('F' * 18 + '\n')
txt_file.close()
```

## 8.3 Estructura with open

[5]:
```
filename = r'C:\Users\sergi\Documents\repos\python_course\data\Dummy.txt' # Es␣
↪buena idea usar r delante de los paths explícitos cuando estamos en Windows.

with open(filename, 'r') as txt_file:
    content = txt_file.read()

print(content)
```

```
AAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDD
AAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDD

EEEEEEEEEEEEEEEEEE
FFFFFFFFFFFFFFFFFF
```

## 8.4 Manejo con módulos específicos

### 8.4.1 CSV

[8]:
```
import csv

csv_path = r'C:\Users\sergi\Documents\repos\python_course\data\data.csv'

## READ
with open(csv_path) as csv_file:
    reader = csv.DictReader(csv_file, delimiter = ',')
    for row in reader:
        print(row)

## WRITE
with open(csv_path, 'a') as csv_file:
    spamwriter = csv.DictWriter(csv_file, delimiter = ',', fieldnames = list(row.
↪keys()))
    spamwriter.writerow(row)
```

```
{'UUID': '65.tif', 'LONG': '36.47207866666667', 'LAT': '-6.249120222222222'}
{'UUID': '65.tif', 'LONG': '36.47207866666667', 'LAT': '-6.249120222222222'}
{'UUID': '65.tif', 'LONG': '36.47207866666667', 'LAT': '-6.249120222222222'}
{'UUID': '65.tif', 'LONG': '36.47207866666667', 'LAT': '-6.249120222222222'}
{'UUID': '65.tif', 'LONG': '36.47207866666667', 'LAT': '-6.249120222222222'}
{'UUID': '65.tif', 'LONG': '36.47207866666667', 'LAT': '-6.249120222222222'}
```

### 8.4.2 JSON

[7]:
```
import json

json_path = r'C:\Users\sergi\Documents\repos\python_course\data\config.json'

## READ
with open(json_path) as json_file:
    data = json.load(json_file)

print(data)

## WRITE
with open(json_path, 'w') as json_file:
    data['WRITE'] = None
    data = json.dump(data, json_file)
```

```
{'ok': True, 'fail': False}
```

### 8.4.3 YAML

[8]:
```
import yaml

yaml_path = r'C:\Users\sergi\Documents\repos\python_course\data\config.yml'

## READ
with open(yaml_path, "r") as yml_file:
    try:
        data = yaml.safe_load(yml_file)
        print(data)
    except yaml.YAMLError as exc:
        print(exc)

## WRITE
with open(yaml_path, "a") as yml_file:
    try:
        data['WRITE'] = None
        yaml.dump(data, yml_file)
    except yaml.YAMLError as exc:
        print(exc)
```

```
{'n_processed': 1, 'logs': {0: 'OK', 1: 'OK', 2: 'OK', 3: 'KO', 4: 'OK', 5:
'OK', 6: 'KO', 7: 'OK', 8: 'KO', 9: 'OK'}}
```

### 8.4.4 TIF

[9]:
```
import rasterio
from pprint import pprint

tif_path = r'C:\Users\sergi\Documents\repos\python_course\data\rgb.tif'

## READ
with rasterio.open(tif_path, 'r') as src:
    data = src.read()
    profile = src.profile
    pprint(f'{profile=}')
    print(f'{data.shape=}')

## WRITE
with rasterio.open(tif_path, 'w', **profile) as dst:
    dst.write(data)
```

```
("profile={'driver': 'GTiff', 'dtype': 'uint8', 'nodata': None, 'width': 433, "
 "'height': 578, 'count': 4, 'crs': None, 'transform': Affine(1.0, 0.0, 0.0,\n"
 "       0.0, 1.0, 0.0), 'blockysize': 7, 'tiled': False, 'compress': 'lzw', "
 "'interleave': 'pixel'}")
data.shape=(4, 578, 433)
```

### 8.4.5 Excel

[10]:
```
import openpyxl

workbook = openpyxl.load_workbook(filename = '../data/Financial Sample.xlsx')

worksheet = workbook['Sheet1']

column_headers = []
for column in worksheet.iter_cols(min_row = 1, max_row = 1, values_only = True):
    column_headers.extend(column)

print('Header')
for index, header in enumerate(column_headers, start = 1):
    print(f"{header}", end = ' | ')
else:
    print(end = '\n')

print('\nContent')
for row in worksheet.iter_rows(min_row = 2, max_row = 4, values_only = True):
    for index, value in enumerate(row, start = 1):
        print(f"{value}", end = ' | ')
    else:
        print(end = '\n')
```

```
workbook.close()
```

```
Header
Segment | Country | Product | Discount Band | Units Sold | Manufacturing Price |
Sale Price | Gross Sales | Discounts |  Sales | COGS | Profit | Date | Month
Number | Month Name | Year |

Content
Government | Canada | Carretera | None | 1618.5 | 3 | 20 | 32370 | 0 | 32370 |
16185 | 16185 | 2014-01-01 00:00:00 | 1 | January | 2014 |
Government | Germany | Carretera | None | 1321 | 3 | 20 | 26420 | 0 | 26420 |
13210 | 13210 | 2014-01-01 00:00:00 | 1 | January | 2014 |
Midmarket | France | Carretera | None | 2178 | 3 | 15 | 32670 | 0 | 32670 |
21780 | 10890 | 2014-06-01 00:00:00 | 6 | June | 2014 |
```

# 9 Librerías comúnmente utilizadas en Ciencia

Al igual que otros lenguajes como R que está especializado en estadística y contiene muchos paquetes, python también una lista muy amplia de librerías y paquetes para procesamiento numérico, estadístico, ...

Algunos ejemplos como se muestra en esta web https://devopedia.org/python-for-scientific-computing son:

Los más conocidos son:

- **Numpy** para procesamiento numérico
- **Pandas** para análisis de datos
- **Matplotlib** para graficación
- **Seaborn** para graficación más avanzada
- **Scikit-learn** para Machine Learning
- ...

## 9.1 Numpy

En python, al ser todo objetos, tenemos el gran inconveniente de que los datos pesan muchos.
Para representar un número entero necesitamos muchos bytes dado que tenemos que almacenar el número en sí junto con los métodos de la clase entero.
Y así con el resto de datos.

Numpy (Numerical Python), es un módulo optimizado de tal forma que podemos realizar operaciones matriciales con más rapidez.

El principal tipo de dato que ofrece Numpy es ndarray (array o lista n-dimensional).

### 9.1.1 Definición de ndarray

El objeto **ndarray** es lo mismo que las listas de python más optimizadas en espacio y tiempo.

Entre sus características destacamos:

- Todos los elementos son del **mismo tipo**, solo trabajamos con listas de enteros, de strings, etc.
- Tenemos un atributo **shape** que es una tupla indicando cuantos elementos tenemos en cada dimensión del ndarray
- El tipo de dato viene determinado por un atributo **dtype** que nos indica con qué dato trabajamos. Ej: int8 (entero de 8 bits), etc.
- Podemos hacer **slicing** y **acceso indexado**
- Podemos modificar en tiempo de ejecución los elementos de la lista
- El **tamaño es fijo**, no podemos añadir o quitar elementos, para ello debemos definir un nuevo ndarray
- Podemos realizar operaciones estadísticas como mean, median, ... y más

### 9.1.2 Creación básica de una ndarray

Crear un ndarray de una dimensión

[1]:
```
import numpy as np

array1d = np.array([1, 2, 3, 4, 5])
array1d.shape
```

[1]:
```
(5,)
```

Crear un ndarray de dos dimensiones

[2]:
```
import numpy as np

array2d = np.array( [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] )
array2d.shape
```

[2]: (3, 3)

Crear un ndarray de tres dimensiones

[3]:
```
import numpy as np

array3d = np.array( [ [ [1, 2] ], [ [3, 4] ], [ [6, 4] ] ] )
array3d.shape
```

[3]: (3, 1, 2)

Crear un ndarray de ceros de tamaño 3x3

[4]:
```
import numpy as np

zeros = np.zeros((3, 3))
zeros.shape
```

[4]: (3, 3)

### 9.1.3 Operaciones sobre ndarrays

#### 9.1.3.1 Indexado

Array 1D

[5]:
```
import numpy as np

array1d = np.array( [1, 2, 3, 4, 5, 6, 7, 8, 9], dtype = np.int8)

print(f'{array1d[0]=}')

print(f'{array1d[2]=}')

print(f'{array1d[1]=}')

print(f'{array1d[-1]=}')

print(f'{array1d[[1, 3]]=}') # Acceder con varios índices
```

```
array1d[0]=1
array1d[2]=3
array1d[1]=2
array1d[-1]=9
array1d[[1, 3]]=array([2, 4], dtype=int8)
```

Array 2D

[6]:
```
import numpy as np
```

```
array2d = np.array( [[1, 2], [3, 4]], dtype = np.int8)

print(f'{array2d[0]=}') # Fila 0

print(f'{array2d[1]=}') # Fila 1

print(f'{array2d[0][0]=}') # Fila 0 Columna 0

print(f'{array2d[:, 0]=}') # Columna 0 de todas las filas
```

```
array2d[0]=array([1, 2], dtype=int8)
array2d[1]=array([3, 4], dtype=int8)
array2d[0][0]=1
array2d[:, 0]=array([1, 3], dtype=int8)
```

#### 9.1.3.2 Slicing

Array 1D

[7]:
```
import numpy as np

array1d = np.array( [1, 2, 3, 4, 5, 6, 7, 8, 9], dtype = np.int8)

print(f'{array1d[0:]=}')
print(f'{array1d[1:]=}')
print(f'{array1d[1:2]=}')
print(f'{array1d[0:2]=}')
```

```
array1d[0:]=array([1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)
array1d[1:]=array([2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)
array1d[1:2]=array([2], dtype=int8)
array1d[0:2]=array([1, 2], dtype=int8)
```

Array 2D

[8]:
```
import numpy as np

array2d = np.array( [ [1, 2, 3, 4, 5, 6, 7, 8],
                      [9, 10, 11, 12, 13, 14, 15, 16],
                      [1, 2, 3, 4, 5, 6, 7, 8],
                      [19, 110, 111, 112, 113, 114, 115, 116] ], dtype = np.int8)

print(f'{array2d[1::2, 0:4]=}') # 4 primeras columnas de las filas impares
```

```
array2d[1::2, 0:4]=array([[  9,  10,  11,  12],
       [ 19, 110, 111, 112]], dtype=int8)
```

#### 9.1.3.3 Modificar elementos

[9]:
```
import numpy as np

array1d = np.array( [1, 2, 3, 4, 5, 6, 7, 8, 9], dtype = np.int8)

array1d[0] = 9

array1d
```

[9]:
```
array([9, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)
```

#### 9.1.3.4 Operaciones aritméticas

[10]:
```
import numpy as np

array1d = np.array( [1, 2, 3], dtype = np.int8)
array2d = np.array( [ [0, 0, 0],
                      [1, 1, 1]], dtype = np.int8)

print(array2d + 1)
```

```
[[1 1 1]
 [2 2 2]]
```

[11]:
```
print(array2d * 1)
```

```
[[0 0 0]
 [1 1 1]]
```

[12]:
```
print(array2d * array1d)
```

```
[[0 0 0]
 [1 2 3]]
```

### 9.1.4 Álgebra lineal

Producto de dos matrices

[13]:
```
import numpy as np

A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
np.dot(A, B)
```

[13]:
```
array([[19, 22],
       [43, 50]])
```

Cálculo de la inversa de una matriz

[14]:
```
import numpy as np
```

```
A = np.array([[1, 2], [3, 4]])
A_inv = np.linalg.inv(A)

print(A)
print(A_inv)
print(A.dot(A_inv))
```

```
[[1 2]
 [3 4]]
[[-2.   1. ]
 [ 1.5 -0.5]]
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

Resolver un sistema de ecuaciones lineales 2x + 3y = 5 x - y = 1

[15]:
```
coefficients = np.array([[2, 3], [1, -1]])
constants = np.array([5, 1])
np.linalg.solve(coefficients, constants)
```

[15]:
```
array([1.6, 0.6])
```

### 9.1.5 Diferencia usar Numpy y no usar Numpy

[16]:
```
import time
import numpy as np

# Tamaño de la matriz
n = 200

# Crear matrices aleatorias sin NumPy
start_time = time.time()

matrix_a = [[i + j for j in range(n)] for i in range(n)]
matrix_b = [[i - j for j in range(n)] for i in range(n)]

result = [[0] * n for _ in range(n)]
for i in range(n):
    for j in range(n):
        for k in range(n):
            result[i][j] += matrix_a[i][k] * matrix_b[k][j]

end_time = time.time()
execution_time = end_time - start_time
print("Tiempo de ejecución en segundos sin NumPy:", execution_time)

# Crear matrices aleatorias con NumPy
start_time = time.time()
```

```
array_a = np.random.rand(n, n)
array_b = np.random.rand(n, n)

result_array = np.dot(array_a, array_b)

end_time = time.time()
execution_time = end_time - start_time
print("Tiempo de ejecución en segundos con NumPy:", execution_time)
```

```
Tiempo de ejecución en segundos sin NumPy: 3.33677339553833
Tiempo de ejecución en segundos con NumPy: 0.0037114620208740234
```

### 9.1.6 Estadísticas

[17]:
```
import numpy

array = np.random.rand(4, 5)

print('Media de todos los valores: ', array.mean())
```

```
Media de todos los valores:  0.6026659401474472
```

[18]:
```
print('Media de todos las filas: ', array.mean(axis = 1))
```

```
Media de todos las filas:  [0.62138473 0.73264987 0.53778201 0.51884715]
```

[19]:
```
print('Media de todas las columnas: ', array.mean(axis = 0))
```

```
Media de todas las columnas:  [0.72060379 0.58694439 0.5962877  0.56353339
0.54596043]
```

[20]:
```
print('Mediana de todos los valores: ', np.median(array))
```

```
Mediana de todos los valores:  0.6267550427898875
```

[21]:
```
print('Maximo de todos los valores: ', np.max(array))
```

```
Maximo de todos los valores:  0.9729526340616049
```

[22]:
```
print('Mínimo de todos los valores: ', np.min(array))
```

```
Mínimo de todos los valores:  0.14986822185497506
```

[23]:
```
print('Desviación estandar de todos los valores: ', np.std(array))
```

```
Desviación estandar de todos los valores:  0.22150210369951986
```

### 9.1.7 Ejercicios

- Crea una matriz de 1s con dimensiones (20, 30, 10)
- Mete algunos NaNs en la matriz anterior usando índices y slicing

- Calcula la media, median, min, max. Pista: buscar nanmean, ...
- Crea varias matrices 2x2 y súmalas entre sí, multiplícalas por un número, ...

## 9.2 Pandas

Pandas en una librería de Python generalmente utilizada para análisis numérico, tratamiento de datos, etc...

En Numpy vimos que el elemento estrella sobre el que está montaba toda la funcionalidad es ndarray. En pandas, de igual forma, tenemos los elementos **DataFrame** y **Series**.

---

- Series:
  - Una serie en una lista al estilo ndarray cuyos elementos son del mismo tipo, enteros, decimales, fechas, ...
  - Las series permiten organizar datos en forma de lista. Estas están indexadas o por números, o por otros elementos como pueden ser texto o fechas.

---

- DataFrame:
  - Un DataFrame, en ensencia, es una concatenación de Series donde cada serie está indexada por una columna (normalmente).
  - Las series del DataFrame deben de tener el mismo número de elementos. Y los elementos de una misma fila tiene el mismo índice para esa fila.

---

En otras palabras, un DataFrame es una tabla donde tenemos columnas y filas. Las filas tienen un índice que nos permite acceder a todos los elementos de dicha fila. Esos índices pueden ser números, letras o fechas. Las columnas son Series indexadas por columnas que normalmente suelen ser texto.

---

### 9.2.1 Creación DataFrame a partir de un diccionario

[24]:
```
import pandas as pd
import numpy as np

data = {
    'Fecha': pd.date_range('2023-01-01', periods = 5),
    'Nombre': ['Juan', 'María', 'Pedro', 'Ana', 'Luisa'],
    'Edad': [25, 30, np.nan, 35, 40],
    'Puntuación': [8.2, 7.5, 6.9, np.nan, 9.0]
}

df = pd.DataFrame(data)
df
```

[24]:

| | Fecha | Nombre | Edad | Puntuación |
|---|---|---|---|---|
| 0 | 2023-01-01 | Juan | 25.0 | 8.2 |
| 1 | 2023-01-02 | María | 30.0 | 7.5 |
| 2 | 2023-01-03 | Pedro | NaN | 6.9 |
| 3 | 2023-01-04 | Ana | 35.0 | NaN |

```
4 2023-01-05  Luisa  40.0         9.0
```

### 9.2.2 Crear DataFrame desde un CSV, ...

[25]:
```
import pandas as pd

df = pd.read_csv(filepath_or_buffer = r'C:
↪\Users\sergi\Documents\repos\python_course\data\pandas_data.csv', sep = ',',␣
↪index_col = False)
df
```

[25]:
```
        Fecha Nombre  Edad  Puntuación
0  2023-01-01   Juan  25.0         8.2
1  2023-01-02  María  30.0         7.5
2  2023-01-03  Pedro   NaN         6.9
3  2023-01-04    Ana  35.0         NaN
4  2023-01-05  Luisa  40.0         9.0
```

### 9.2.3 Análisis de datos

Resumen inicial

Vamos a usar un fichero .csv que contiene información sobre pasajeros del titanic. Indicando si murieron, si eran tercera clase, ...

[26]:
```
import pandas as pd

df = pd.read_csv(r'C:
↪\Users\sergi\Documents\repos\python_course\data\pandas\Titanic.csv')

df.head() # Mostramos las primeras filas
```

[26]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
```

```
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Indices y columnas

[27]:
```
print(f'Indices: {df.index=}')
print(f'Columnas: {list(df.columns)=}')
```

```
Indices: df.index=RangeIndex(start=0, stop=891, step=1)
Columnas: list(df.columns)=['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex',
'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
```

Información general

[28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Estadísticos básicos

[29]:
```
df.describe()
```

[29]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
```

```
max     891.000000    1.000000    3.000000   80.000000    8.000000
```

```
            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

Valores perdidos

[30]:
```
for series in df:
    no_nan_values = df[series].count()
    total_values = len(df[series])
    print(f'Column {series}: Non NAN values {no_nan_values}/{total_values}, {␣
 ↪round((no_nan_values/total_values) * 100, 4)}% of data')
```

```
Column PassengerId: Non NAN values 891/891, 100.0% of data
Column Survived: Non NAN values 891/891, 100.0% of data
Column Pclass: Non NAN values 891/891, 100.0% of data
Column Name: Non NAN values 891/891, 100.0% of data
Column Sex: Non NAN values 891/891, 100.0% of data
Column Age: Non NAN values 714/891, 80.1347% of data
Column SibSp: Non NAN values 891/891, 100.0% of data
Column Parch: Non NAN values 891/891, 100.0% of data
Column Ticket: Non NAN values 891/891, 100.0% of data
Column Fare: Non NAN values 891/891, 100.0% of data
Column Cabin: Non NAN values 204/891, 22.8956% of data
Column Embarked: Non NAN values 889/891, 99.7755% of data
```

Eliminar una o varias columnas

[31]:
```
df.drop(columns = 'PassengerId')
```

[31]:
```
     Survived  Pclass                                               Name  \
0           0       3                            Braund, Mr. Owen Harris
1           1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2           1       3                             Heikkinen, Miss. Laina
3           1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4           0       3                           Allen, Mr. William Henry
..        ...     ...                                                ...
886         0       2                              Montvila, Rev. Juozas
887         1       1                       Graham, Miss. Margaret Edith
888         0       3           Johnston, Miss. Catherine Helen "Carrie"
889         1       1                              Behr, Mr. Karl Howell
890         0       3                                Dooley, Mr. Patrick
```

```
        Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0      male  22.0      1      0         A/5 21171   7.2500   NaN        S
1    female  38.0      1      0          PC 17599  71.2833   C85        C
2    female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3    female  35.0      1      0            113803  53.1000  C123        S
4      male  35.0      0      0            373450   8.0500   NaN        S
..      ...   ...    ...    ...               ...      ...   ...      ...
886    male  27.0      0      0            211536  13.0000   NaN        S
887  female  19.0      0      0            112053  30.0000   B42        S
888  female   NaN      1      2        W./C. 6607  23.4500   NaN        S
889    male  26.0      0      0            111369  30.0000  C148        C
890    male  32.0      0      0            370376   7.7500   NaN        Q

[891 rows x 11 columns]
```

Eliminar columas con NaNs

[32]:
```
df.dropna(axis = 1)
```

[32]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex  SibSp  Parch  \
0                             Braund, Mr. Owen Harris    male      1      0
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female      1      0
2                                Heikkinen, Miss. Laina  female      0      0
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female      1      0
4                               Allen, Mr. William Henry    male      0      0
..                                                 ...     ...    ...    ...
886                               Montvila, Rev. Juozas    male      0      0
887                        Graham, Miss. Margaret Edith  female      0      0
888            Johnston, Miss. Catherine Helen "Carrie"  female      1      2
889                               Behr, Mr. Karl Howell    male      0      0
890                                 Dooley, Mr. Patrick    male      0      0

               Ticket     Fare
0           A/5 21171   7.2500
```

```
1            PC 17599  71.2833
2    STON/O2. 3101282   7.9250
3              113803  53.1000
4              373450   8.0500
..                ...      ...
886            211536  13.0000
887            112053  30.0000
888        W./C. 6607  23.4500
889            111369  30.0000
890            370376   7.7500

[891 rows x 9 columns]
```

Eliminar filas con NaNs

[33]:
```
df.dropna(axis = 0)
```

[33]:
```
     PassengerId  Survived  Pclass  \
1              2         1       1
3              4         1       1
6              7         0       1
10            11         1       3
11            12         1       1
..           ...       ...     ...
871          872         1       1
872          873         0       1
879          880         1       1
887          888         1       1
889          890         1       1

                                                  Name     Sex   Age  SibSp  \
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                              McCarthy, Mr. Timothy J    male  54.0      0
10                    Sandstrom, Miss. Marguerite Rut  female   4.0      1
11                           Bonnell, Miss. Elizabeth  female  58.0      0
..                                                ...     ...   ...    ...
871   Beckwith, Mrs. Richard Leonard (Sallie Monypeny)  female  47.0      1
872                           Carlsson, Mr. Frans Olof    male  33.0      0
879      Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)  female  56.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
889                             Behr, Mr. Karl Howell    male  26.0      0

     Parch    Ticket     Fare        Cabin Embarked
1        0  PC 17599  71.2833          C85        C
3        0    113803  53.1000         C123        S
6        0     17463  51.8625          E46        S
10       1   PP 9549  16.7000           G6        S
```

```
11       0    113783  26.5500         C103        S
..     ...       ...      ...          ...      ...
871      1     11751  52.5542          D35        S
872      0       695   5.0000  B51 B53 B55        S
879      1     11767  83.1583          C50        C
887      0    112053  30.0000          B42        S
889      0    111369  30.0000         C148        C

[183 rows x 12 columns]
```

Gestión valores duplicados

En este caso nos quedamos igual, así que no hay duplicados por filas

[34]:
```
df.drop_duplicates()
```

[34]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex   Age  SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                              Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0
..                                                ...     ...   ...    ...
886                            Montvila, Rev. Juozas    male  27.0      0
887                     Graham, Miss. Margaret Edith  female  19.0      0
888         Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                            Behr, Mr. Karl Howell    male  26.0      0
890                              Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
```

```
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

Rellenar datos NaN con un valor específico

[35]:
```
# df['Age'].fillna(value = -11)
# df.fillna(value = -11)
df.Age.fillna(value = -11).describe()
```

[35]:
```
count    891.000000
mean      21.614108
std       20.809470
min      -11.000000
25%        6.000000
50%       24.000000
75%       35.000000
max       80.000000
Name: Age, dtype: float64
```

Rellenar datos NaN con un valor estadístico

[36]:
```
df.Age.fillna(value = df.Age.mean())
```

[36]:
```
0      22.000000
1      38.000000
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: Age, Length: 891, dtype: float64
```

Estudio de correlación

[37]:
```
df.drop(columns = 'PassengerId').corr(method = 'pearson', numeric_only=True)
```

[37]:
```
           Survived    Pclass       Age     SibSp     Parch      Fare
Survived   1.000000 -0.338481 -0.077221 -0.035322  0.081629  0.257307
Pclass    -0.338481  1.000000 -0.369226  0.083081  0.018443 -0.549500
```

```
Age      -0.077221 -0.369226  1.000000 -0.308247 -0.189119  0.096067
SibSp    -0.035322  0.083081 -0.308247  1.000000  0.414838  0.159651
Parch     0.081629  0.018443 -0.189119  0.414838  1.000000  0.216225
Fare      0.257307 -0.549500  0.096067  0.159651  0.216225  1.000000
```

[38]:
```
df.drop(columns = 'PassengerId').corr(method = 'kendall', numeric_only=True)
```

[38]:
```
           Survived    Pclass       Age     SibSp     Parch      Fare
Survived  1.000000 -0.323533 -0.043385  0.085915  0.133933  0.266229
Pclass   -0.323533  1.000000 -0.286081 -0.039552 -0.021019 -0.573531
Age      -0.043385 -0.286081  1.000000 -0.142746 -0.200112  0.093249
SibSp     0.085915 -0.039552 -0.142746  1.000000  0.425241  0.358262
Parch     0.133933 -0.021019 -0.200112  0.425241  1.000000  0.330360
Fare      0.266229 -0.573531  0.093249  0.358262  0.330360  1.000000
```

[39]:
```
df.drop(columns = 'PassengerId').corr(method = 'spearman', numeric_only=True)
```

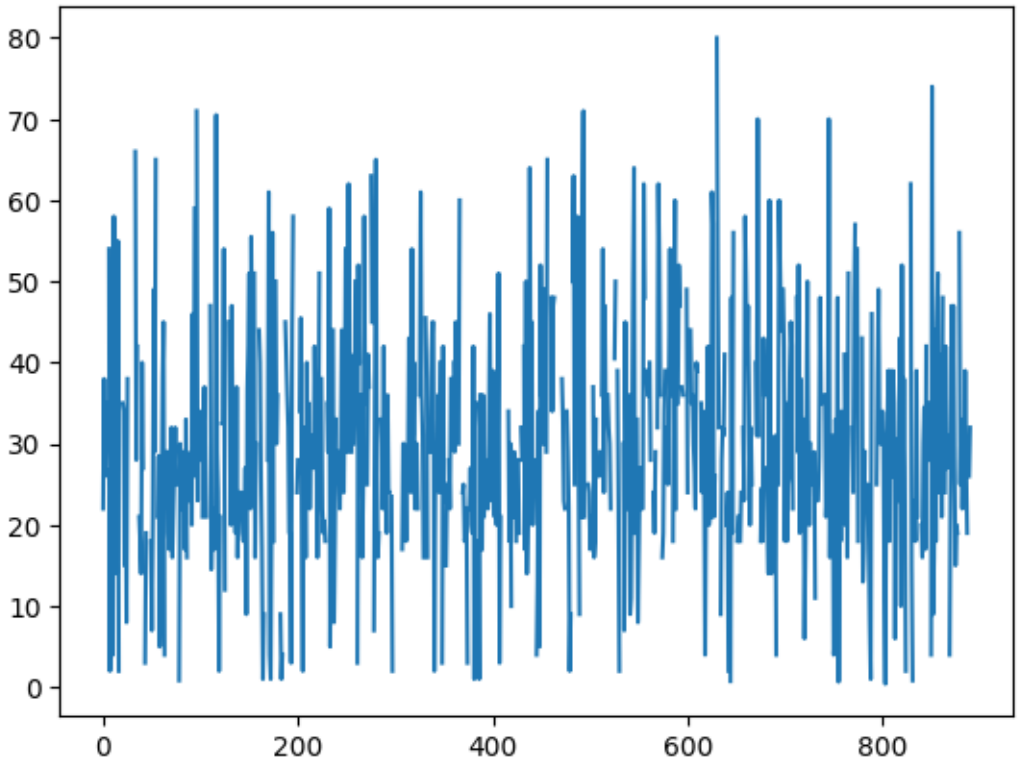
[39]:
```
           Survived    Pclass       Age     SibSp     Parch      Fare
Survived  1.000000 -0.339668 -0.052565  0.088879  0.138266  0.323736
Pclass   -0.339668  1.000000 -0.361666 -0.043019 -0.022801 -0.688032
Age      -0.052565 -0.361666  1.000000 -0.182061 -0.254212  0.135051
SibSp     0.088879 -0.043019 -0.182061  1.000000  0.450014  0.447113
Parch     0.138266 -0.022801 -0.254212  0.450014  1.000000  0.410074
Fare      0.323736 -0.688032  0.135051  0.447113  0.410074  1.000000
```

### 9.2.4 Gráficas básicas

Gráfica de líneas sobre Edad

[40]:
```
df.Age.plot()
```

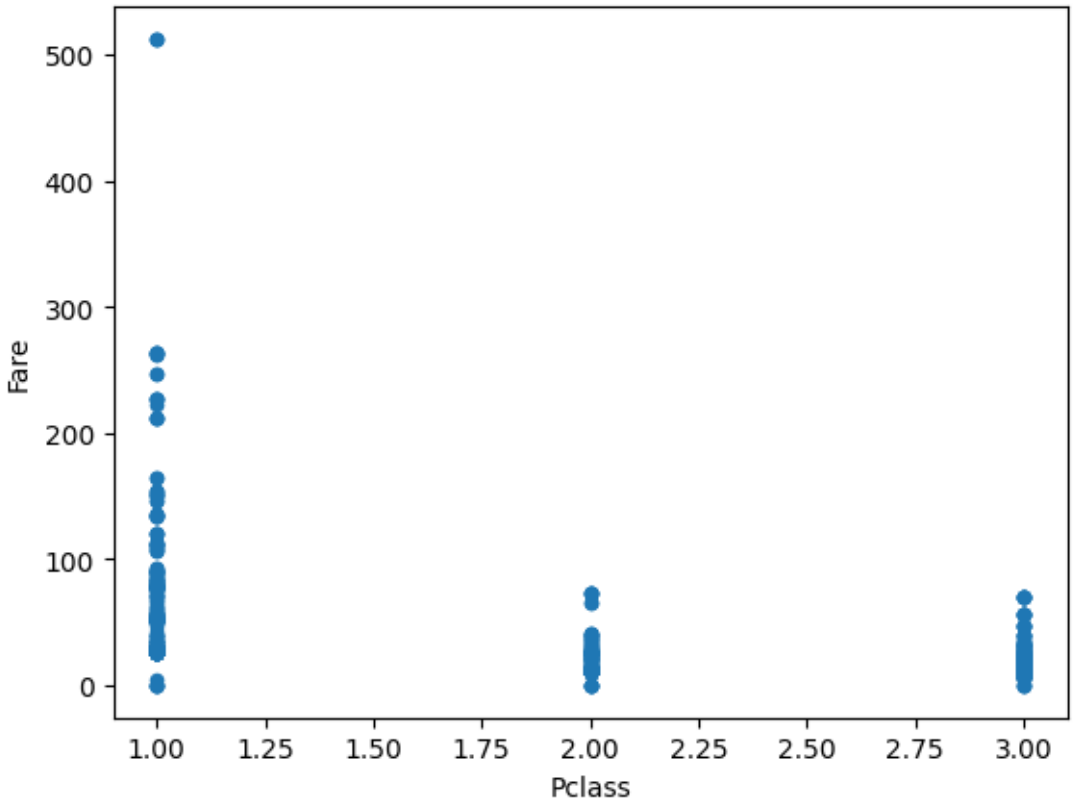
[40]:
```
<Axes: >
```




Distribución Clase y precio Ticket

[41]:
```
df.plot(x = 'Pclass', y = 'Fare', kind='scatter')
```

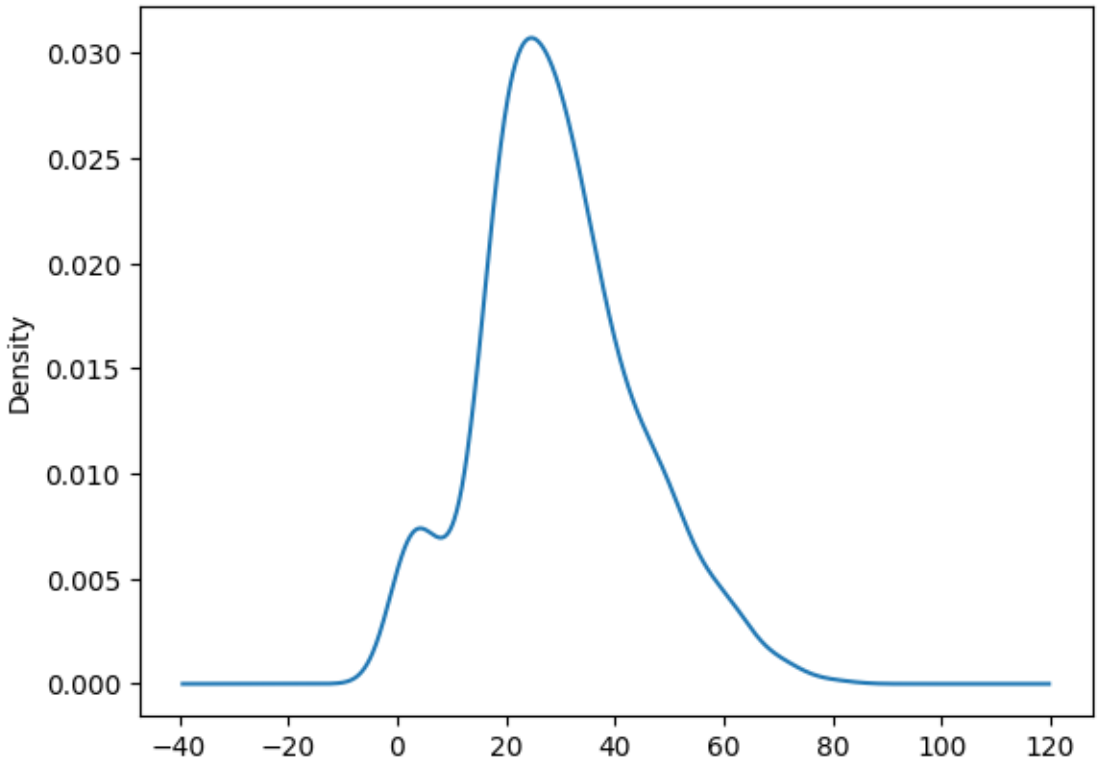
[41]:
```
<Axes: xlabel='Pclass', ylabel='Fare'>
```




Gráficas de distribución

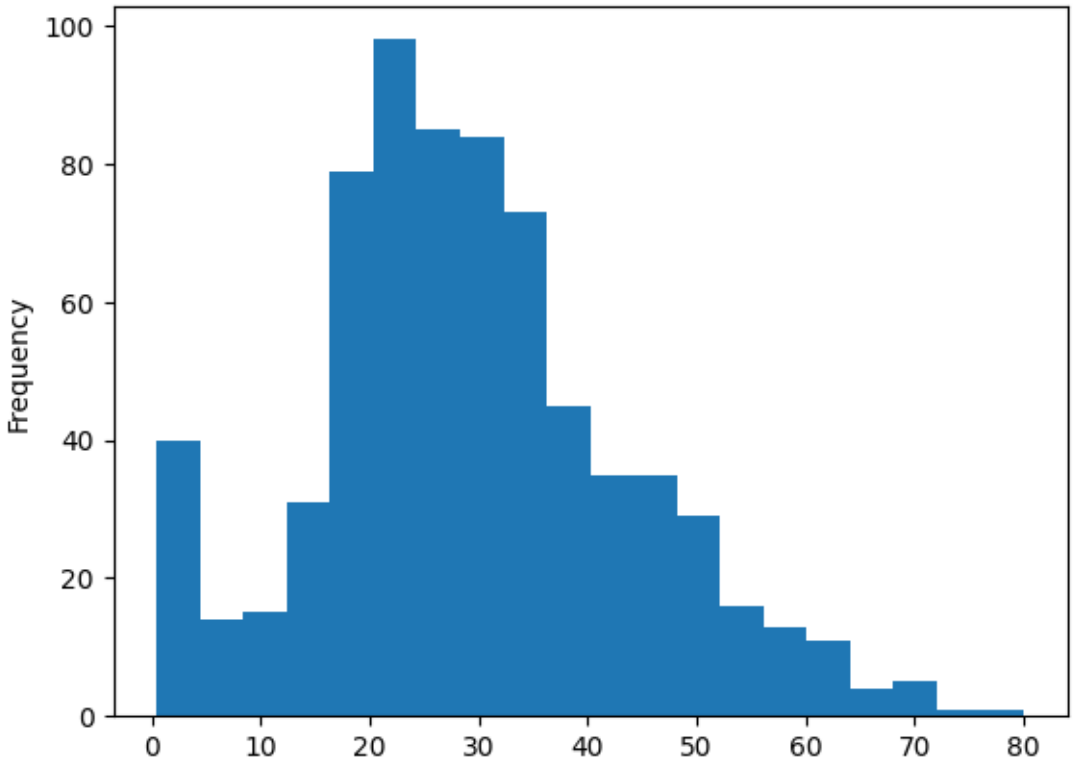
[42]:
```
df.Age.plot(kind = 'kde')
```

[42]:
```
<Axes: ylabel='Density'>
```



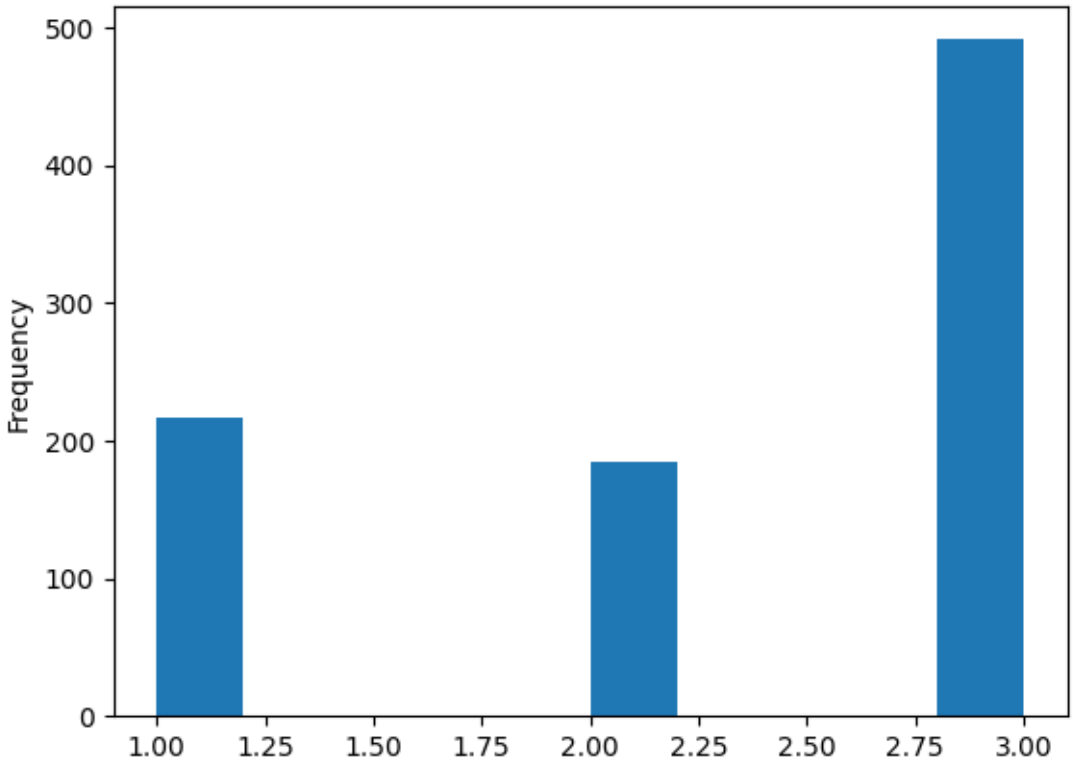

[43]:
```
df.Age.plot(kind = 'hist', bins = 20)
```

[43]: <Axes: ylabel='Frequency'>



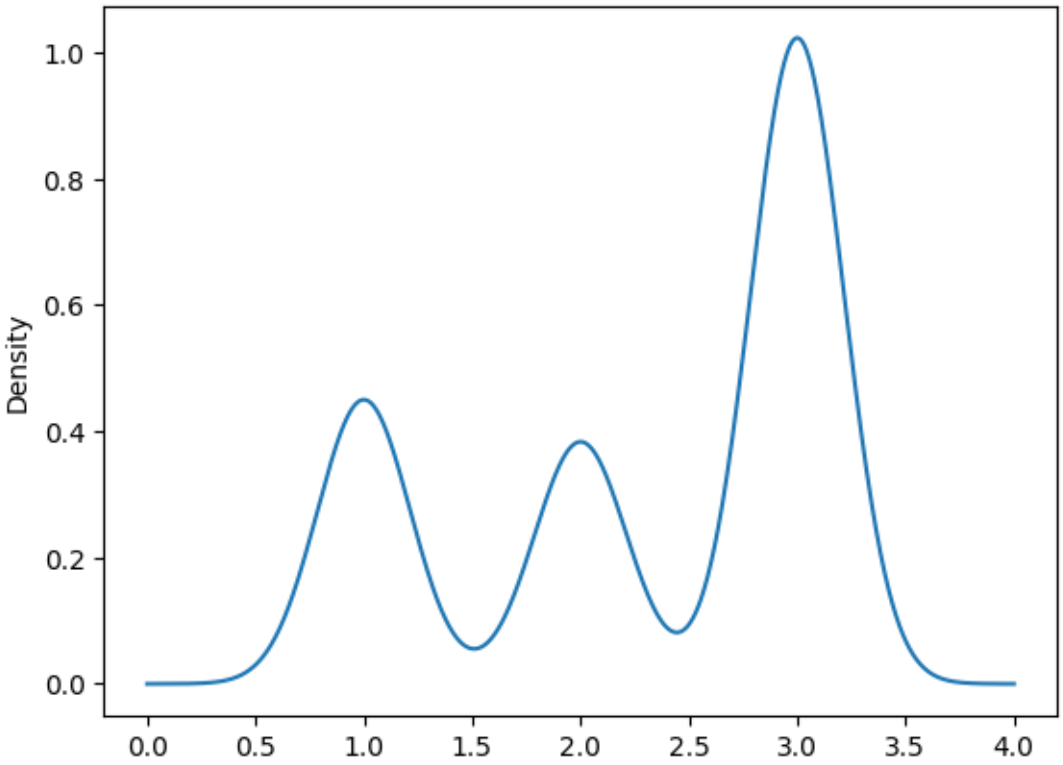

[44]:
```
df.Pclass.plot(kind='hist')
```

[44]:
```
<Axes: ylabel='Frequency'>
```




[45]:
```
df.Pclass.plot(kind='kde')
```

[45]: <Axes: ylabel='Density'>

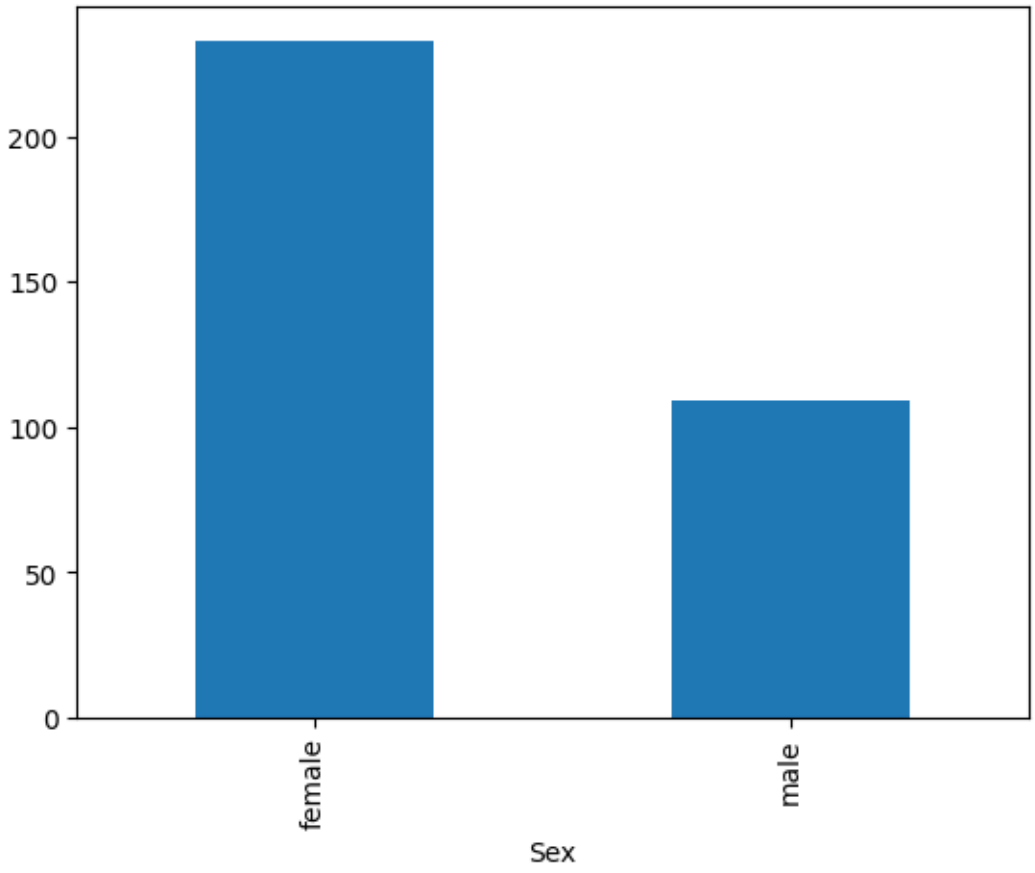



Gráficos de barras

Números supervivientes según sexo

[46]:
```
df.groupby('Sex').Survived.sum().plot(kind='bar')
```

[46]:
```
<Axes: xlabel='Sex'>
```




Extraemos los valores únicos para el **Sexo de los pasajeros** y **Clases**

[47]:
```
print(f"Géneros: {list(df['Sex'].unique())}")
print(f"Clases: {list(df['Pclass'].unique())}")
```

```
Géneros: ['male', 'female']
Clases: [3, 1, 2]
```

### 9.2.5 Ejercicios

Usa el archivo births.csv de la carpeta data/pandas

- Crea un DataFrame a partir del archivo especificado
- Haz un estudio de valores únicos, no nulos, duplicados, ...
- En caso de faltar datos (NaNs) haz lo que creas conveniente. Eliminar filas, columnas, rellenar, ...
- Crea una nueva columna llamada **date** que una los valores de las columnas **year**, **month** y **day** con el formato **year-month-day**
- Crea un nuevo DataFrame a partir del nuevo (incluyendo la nueva columna), borra la columna gender y en la crea una nueva columna con el número de nacimientos en cada fecha sin discriminar por género.

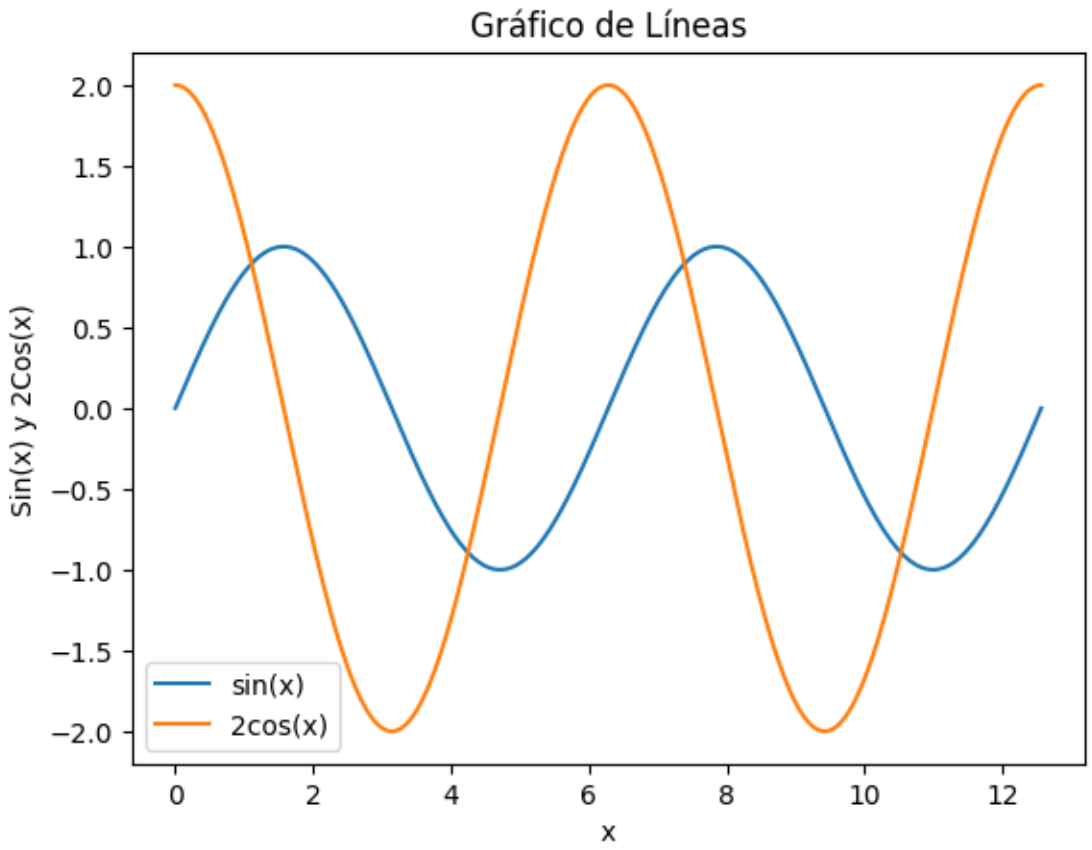
- Crea una gráfica de líneas usando el DataFrame original para ver la evolución de los nacimientos desde el año 1969 hasta el año 2008. Crea una gráfica para cada género.
- Haz lo mismo pero sin discriminar por género con el DataFrame nuevo

## 9.3 Matplotlib

Matplotlib es una librería de Python montada bajo Numpy con la finalidad de generar gráficas. Está basada en Matlab y al igual que ese lenguaje, se suele usar mucho en Ciencia y en ingeniería.

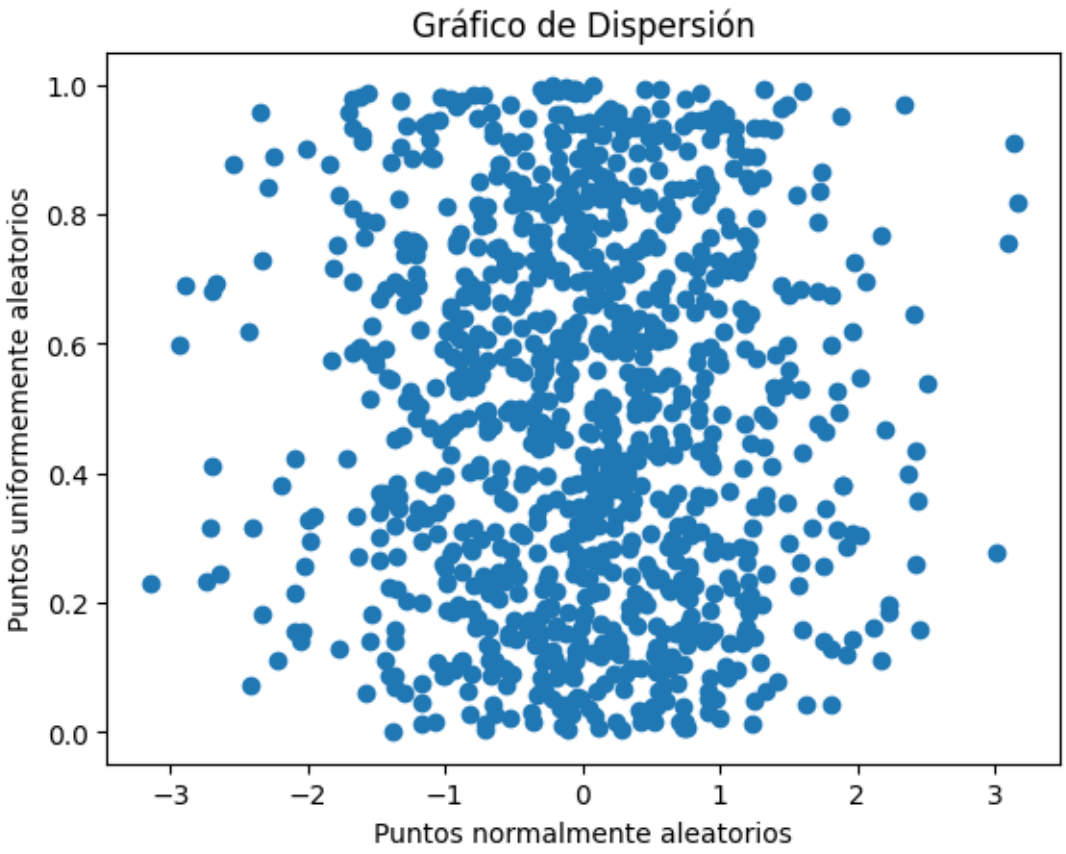
### 9.3.1 Gráfica de lineas

[48]:
```
import matplotlib.pyplot as plt
import numpy as np

# Datos de ejemplo
x = np.linspace(0, 4 * np.pi, 1000)
y_sin = np.sin(x)
y_2cos = np.cos(x) * 2

# Crear el gráfico de líneas
plt.plot(x, y_sin, label = 'sin(x)')
plt.plot(x, y_2cos, label = '2cos(x)')

# Personalizar el gráfico
plt.xlabel('x')
plt.ylabel('Sin(x) y 2Cos(x)')
plt.title('Gráfico de Líneas')
plt.legend()

# Mostrar el gráfico
plt.show()
```




### 9.3.2 Gráfica de puntos

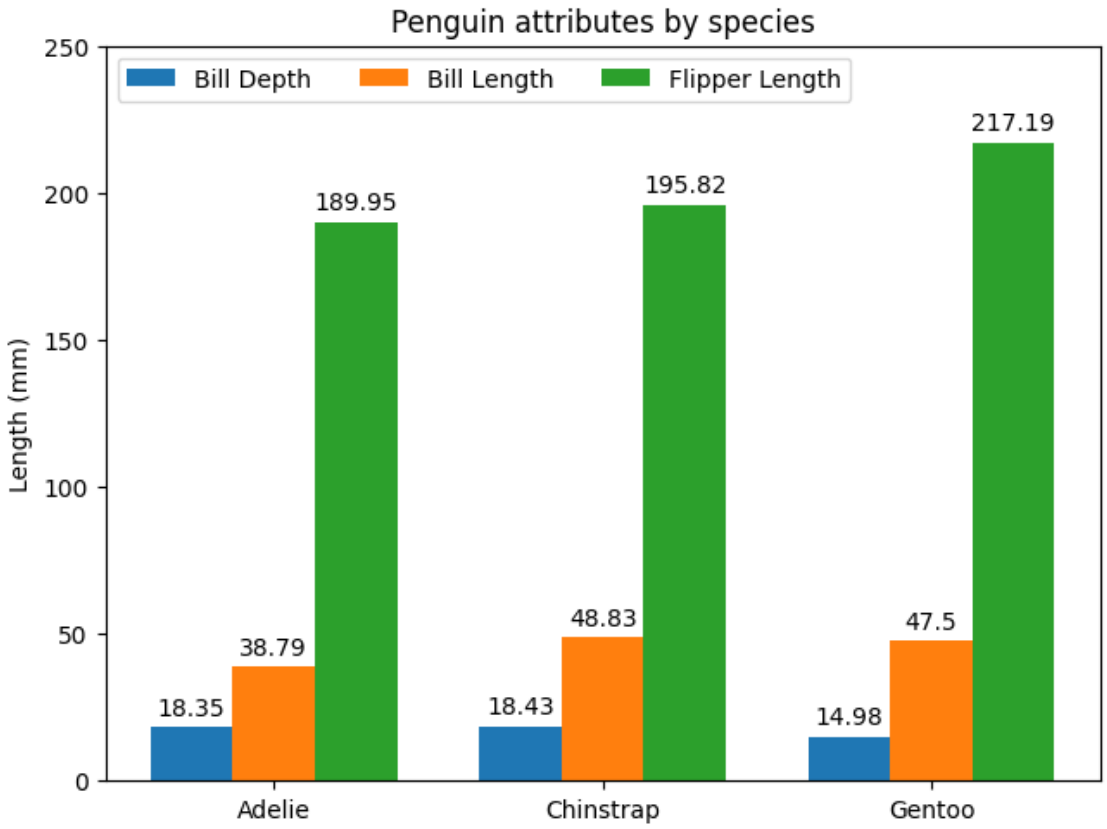
[49]:
```
import matplotlib.pyplot as plt
import numpy as np

# Datos de ejemplo
x = np.random.randn(1000)
y = np.random.rand(1000)

# Crear el gráfico de dispersión
plt.scatter(x, y)

# Personalizar el gráfico
plt.xlabel('Puntos normalmente aleatorios')
plt.ylabel('Puntos uniformemente aleatorios')
plt.title('Gráfico de Dispersión')

# Mostrar el gráfico
plt.show()
```




### 9.3.3 Gráfica de barras

[50]:
```
# data from https://allisonhorst.github.io/palmerpenguins/

import matplotlib.pyplot as plt
import numpy as np

species = ("Adelie", "Chinstrap", "Gentoo")
penguin_means = {
    'Bill Depth': (18.35, 18.43, 14.98),
    'Bill Length': (38.79, 48.83, 47.50),
    'Flipper Length': (189.95, 195.82, 217.19),
}

x = np.arange(len(species))  # the label locations
width = 0.25  # the width of the bars

fig, ax = plt.subplots(layout = 'constrained')

for idx, (attribute, measurement) in enumerate(penguin_means.items()):
```

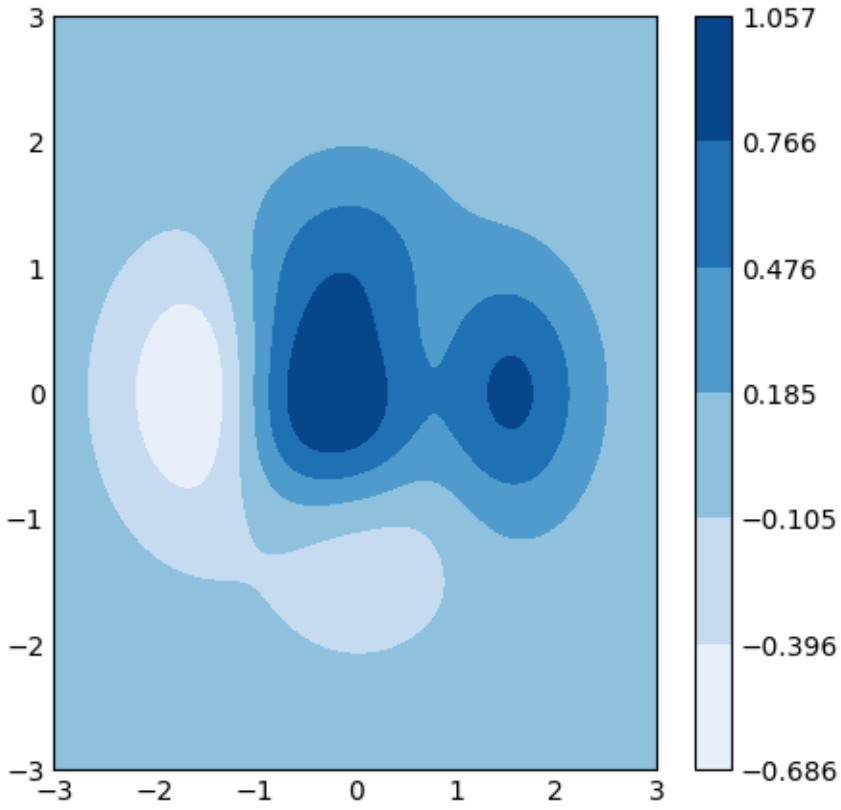
```
    offset = width * idx
    rects = ax.bar(x + offset, measurement, width, label = attribute)
    ax.bar_label(rects, padding = 3)

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Length (mm)')
ax.set_title('Penguin attributes by species')
ax.set_xticks(x + width, species)
ax.legend(loc = 'upper left', ncols = 3)
ax.set_ylim(0, 250)

plt.show()
```




### 9.3.4 Contornos

[51]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('_mpl-gallery-nogrid')
```

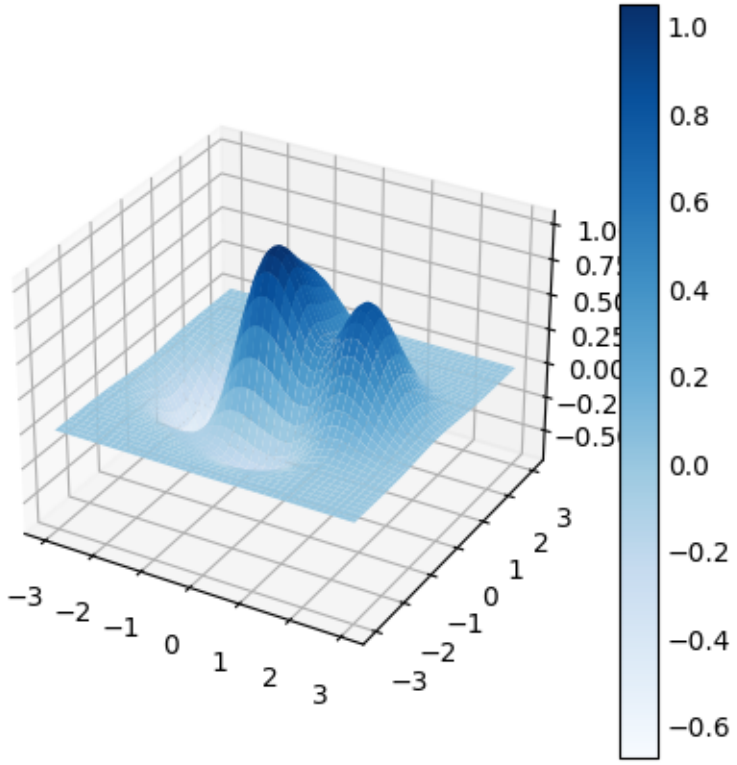
```
# make data
X, Y = np.meshgrid(np.linspace(-3, 3, 256), np.linspace(-3, 3, 256))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)
levels = np.linspace(Z.min(), Z.max(), 7)

# plot
fig, ax = plt.subplots()
fig.set_figwidth(4)
fig.set_figheight(4)
contour = ax.contourf(X, Y, Z, levels = levels)

plt.colorbar(contour)
plt.show()
```




### 9.3.5 Gráficos 3D

[52]:
```
import matplotlib.pyplot as plt
import numpy as np

from matplotlib import cm
```

```
plt.style.use('_mpl-gallery-nogrid')

# make data
X, Y = np.meshgrid(np.linspace(-3, 3, 256), np.linspace(-3, 3, 256))
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)

# plot
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
fig.set_figwidth(4)
fig.set_figheight(4)
surface = ax.plot_surface(X, Y, Z, cmap = cm.Blues)

plt.colorbar(surface)

plt.show()
```

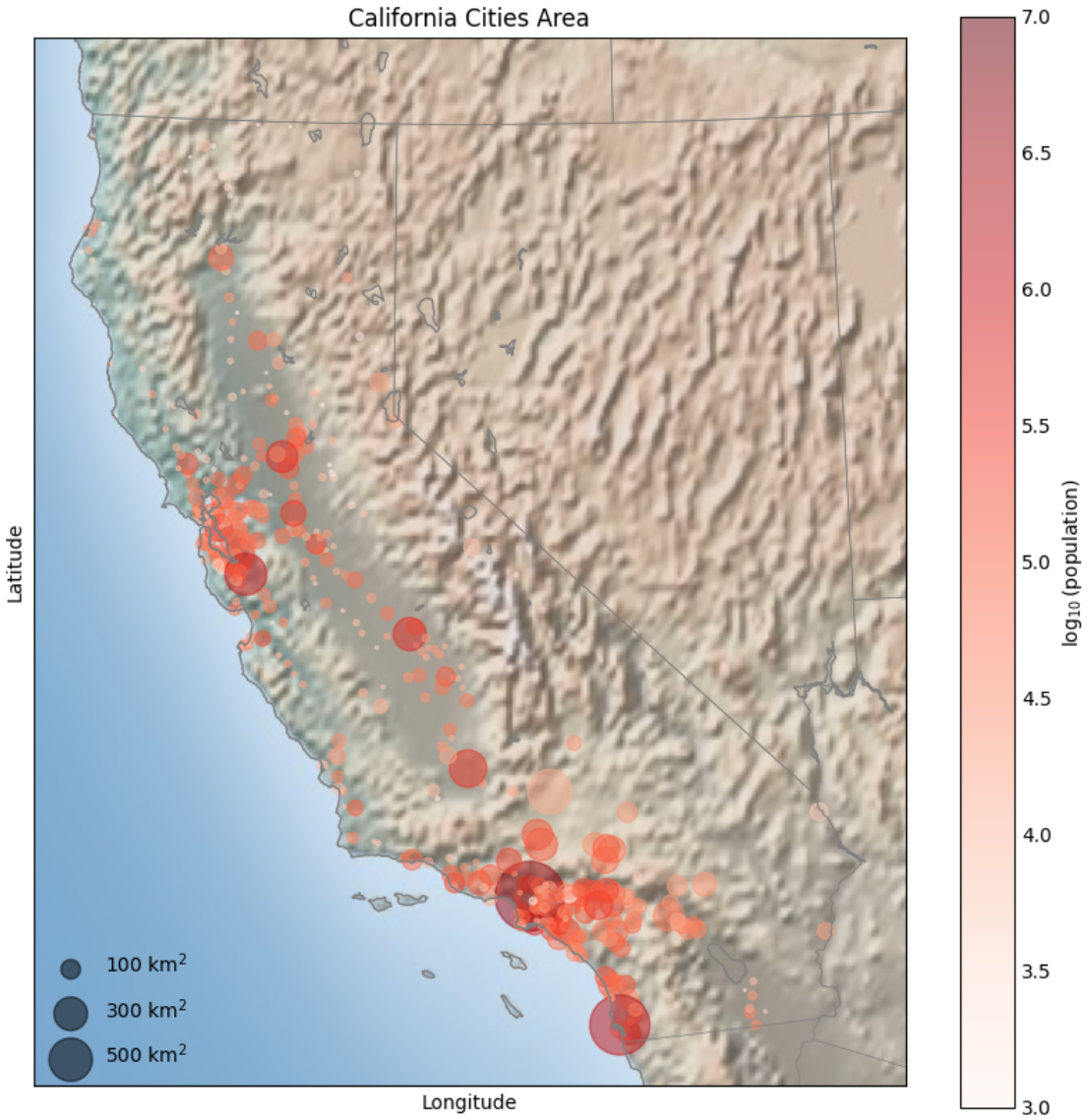



### 9.3.6 Gráficos con mapas mundiales

[53]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
from mpl_toolkits.basemap import Basemap

cities = pd.read_csv(r'C:
 ↪\Users\sergi\Documents\repos\python_course\data\california_cities.csv')

# Extract the data we're interested in
lat = cities['latd'].values
lon = cities['longd'].values
population = cities['population_total'].values
area = cities['area_total_km2'].values

# 1. Draw the map background
fig = plt.figure(figsize = (8, 8))
m = Basemap(projection = 'lcc', resolution='h',
            lat_0 = 37.5, lon_0 = -119,
            width = 1E6, height = 1.2E6)
m.shadedrelief() # Añade el fondo con relieve terrestre
m.drawcoastlines(color = 'gray') # Añade las líneas de costa
m.drawcountries(color = 'gray') # Añade las fronteras de los paises
m.drawstates(color = 'gray') # Añade las fronteras de los estados de EEUU

# 2. scatter city data, with color reflecting population
# and size reflecting area
m.scatter(lon, lat, latlon=True,
          c = np.log10(population), s = area,
          cmap = 'Reds', alpha = 0.5)

# 3. create colorbar and legend
plt.colorbar(label = r'$\log_{10}({\rm population})$') # Añade la colorbar
plt.clim(3, 7) # Limita la colorbar a mínimo 3 y máximo 7

# make legend with dummy points
for a in [100, 300, 500]: # Añade las legendas
    plt.scatter([], [], c = 'k', alpha = 0.5, s = a, label = str(a) + ' km$^2$')
plt.legend(scatterpoints = 1, frameon = False, labelspacing = 1, loc = 'lower␣
 ↪left')
plt.title('California Cities Area')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

[53]: `Text(0, 0.5, 'Latitude')`




### 9.3.7 Ejercicios

Mira brevemente los siguientes enlaces:

- Tipos de gráficas: https://matplotlib.org/stable/plot_types/index.html
- Ejemplos: https://matplotlib.org/stable/gallery/index.html#

Ejercicios:

- Haz una gráfica de una serie temporal
- Haz una gráfica del histograma de una variable continua y otra discreta
- Elige algún ejemplo que te interese y trata de replicarlo con datos tuyos

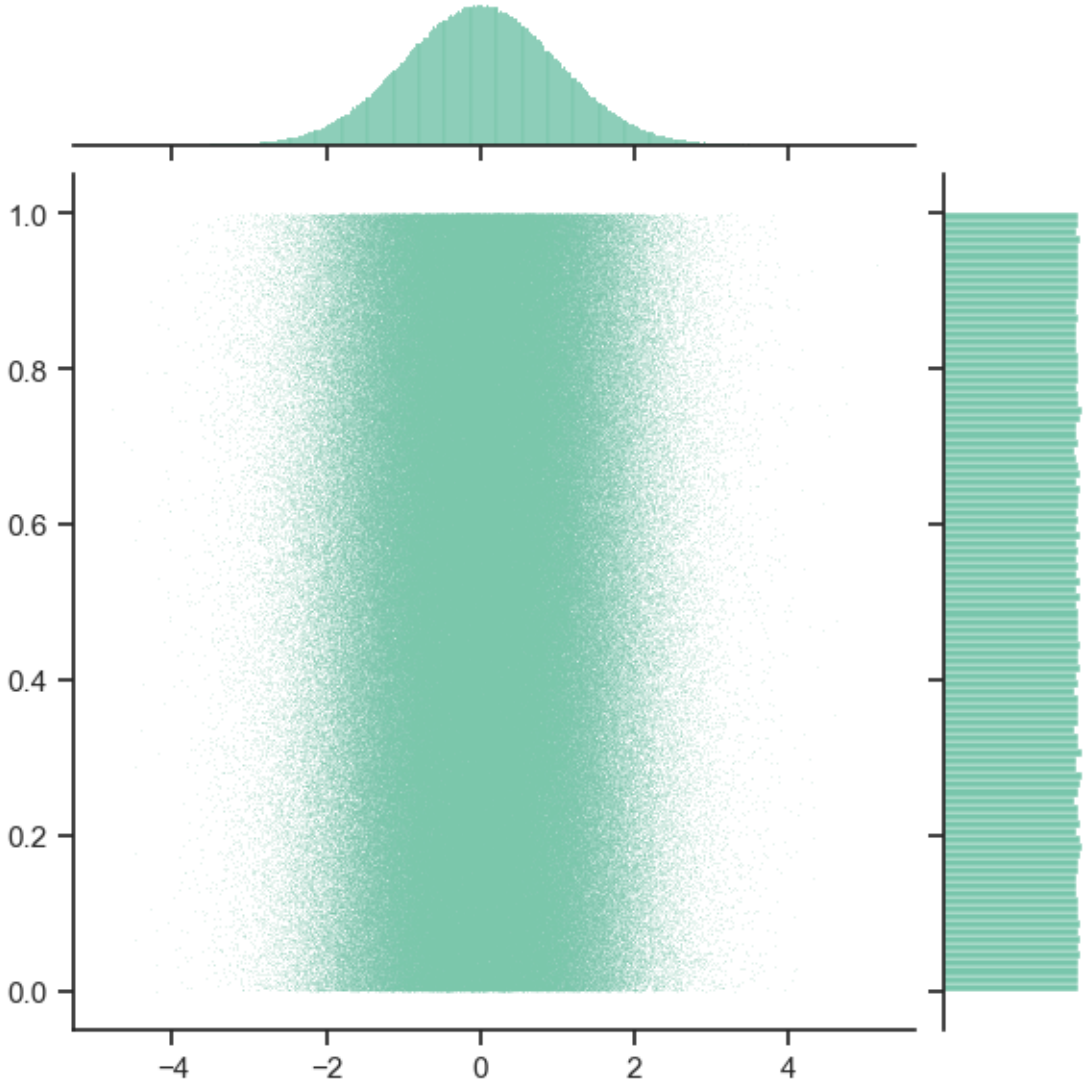
## 9.4 Seaborn

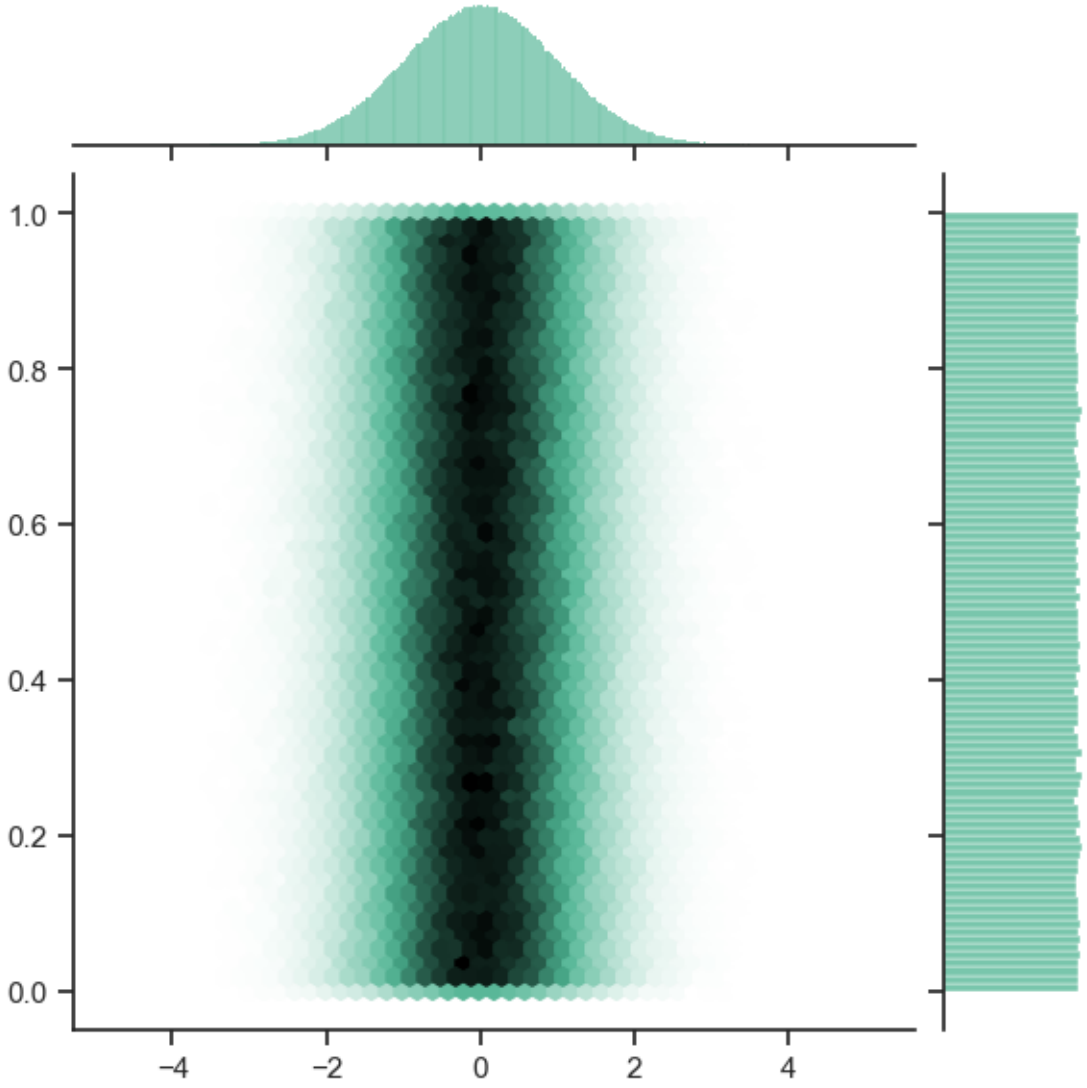
SeaBorn es una librería montada sobre Matplotlib para realizar gráficos más descriptivos para el análisis estadístico

### 9.4.1 Gráfico de distribución hexagonal para concentraciones

[54]:
```
import numpy as np
import seaborn as sns

sns.set_theme(style = 'ticks')

x = np.random.randn(1000000)
y = np.random.rand(1000000)

sns.jointplot(x = x, y = y, kind = 'scatter', color = '#4CB391', s = 0.1)
sns.jointplot(x = x, y = y, kind = 'hex', color = '#4CB391')
```

[54]:
```
<seaborn.axisgrid.JointGrid at 0x266c89576a0>
```


1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4

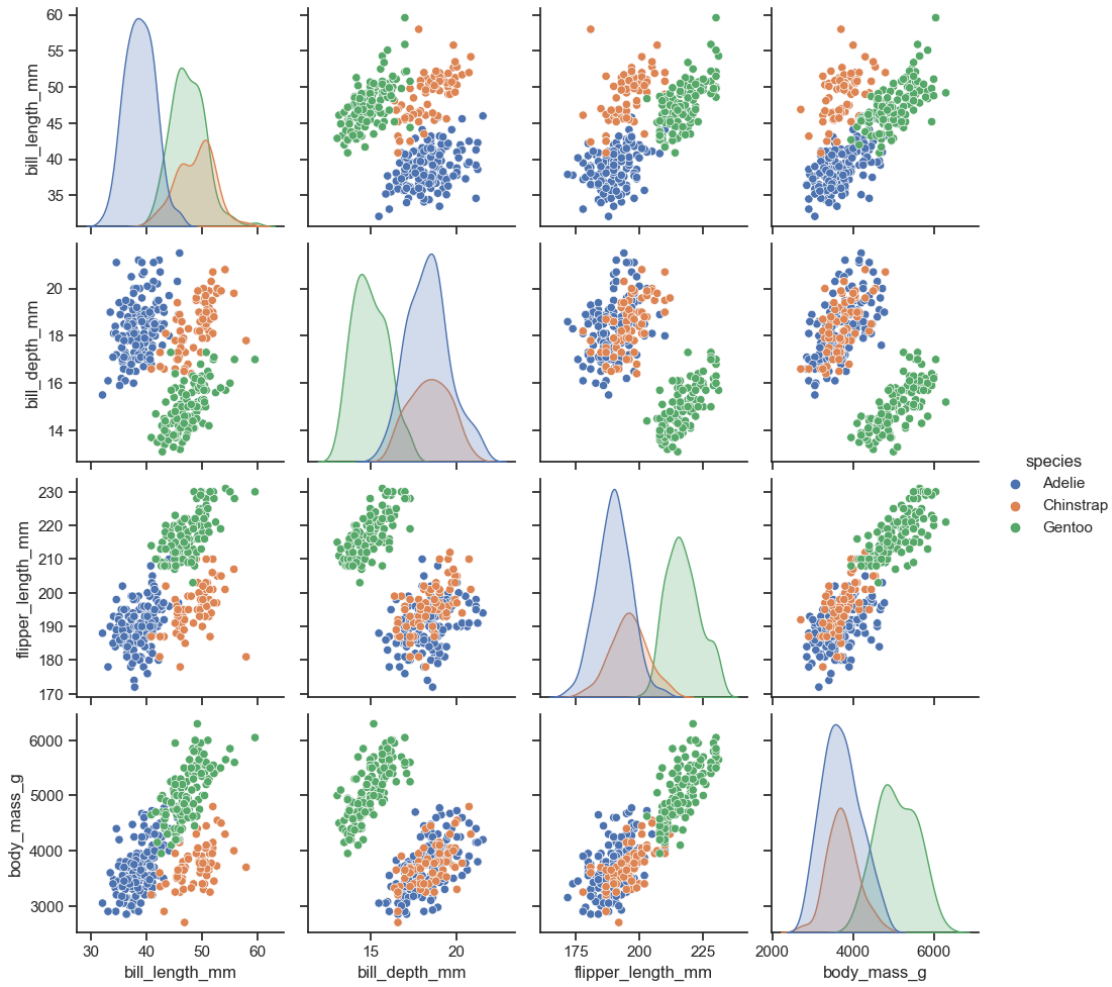



[1]:
```
import seaborn as sns

df = sns.load_dataset('titanic')
df = df.drop(columns = ['adult_male', 'alone'])
corr = df.corr(numeric_only = True)

cmap = sns.diverging_palette(230, 20, as_cmap = True)
sns.heatmap(corr, cmap = cmap, linewidths = .5, annot = True)
```

[1]: <Axes: >

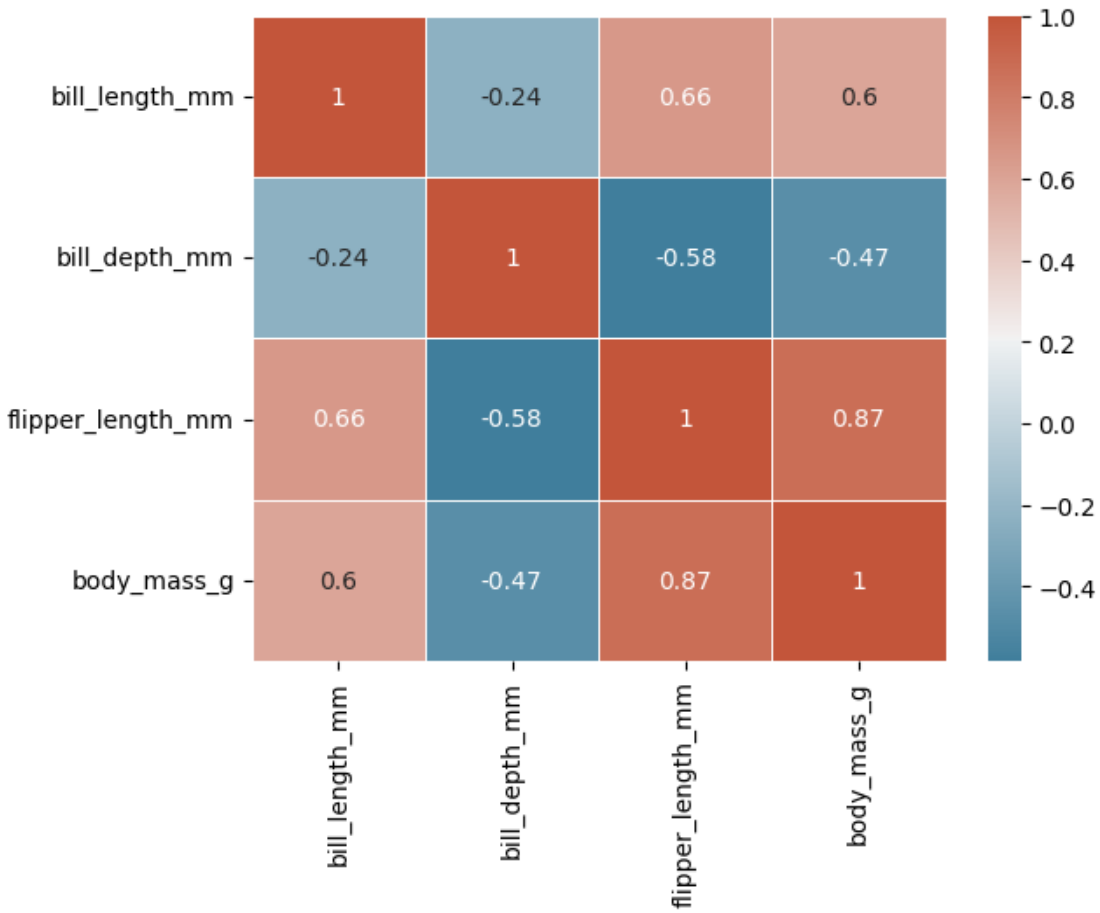

### 9.4.2 Gráficos de densidad y distribución

[7]:
```
import seaborn as sns

sns.set_theme(style = 'ticks')

plot = sns.pairplot(sns.load_dataset('penguins'), hue = 'species')
```




[2]:
```
import seaborn as sns

corr = sns.load_dataset('penguins').corr(numeric_only = True)

cmap = sns.diverging_palette(230, 20, as_cmap = True)
sns.heatmap(corr, cmap = cmap, linewidths = .5, annot = True)
```

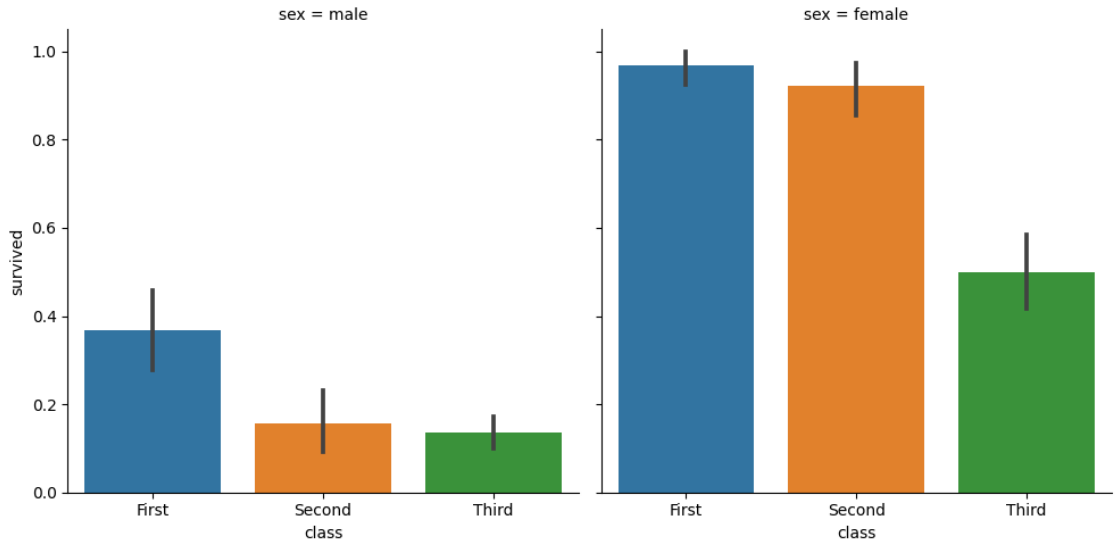
[2]: <Axes: >




### 9.4.3 Gráficos de barras

[6]:
```
sns.catplot(data = df, x = 'class', y = 'survived', col = 'sex', kind = 'bar')
```

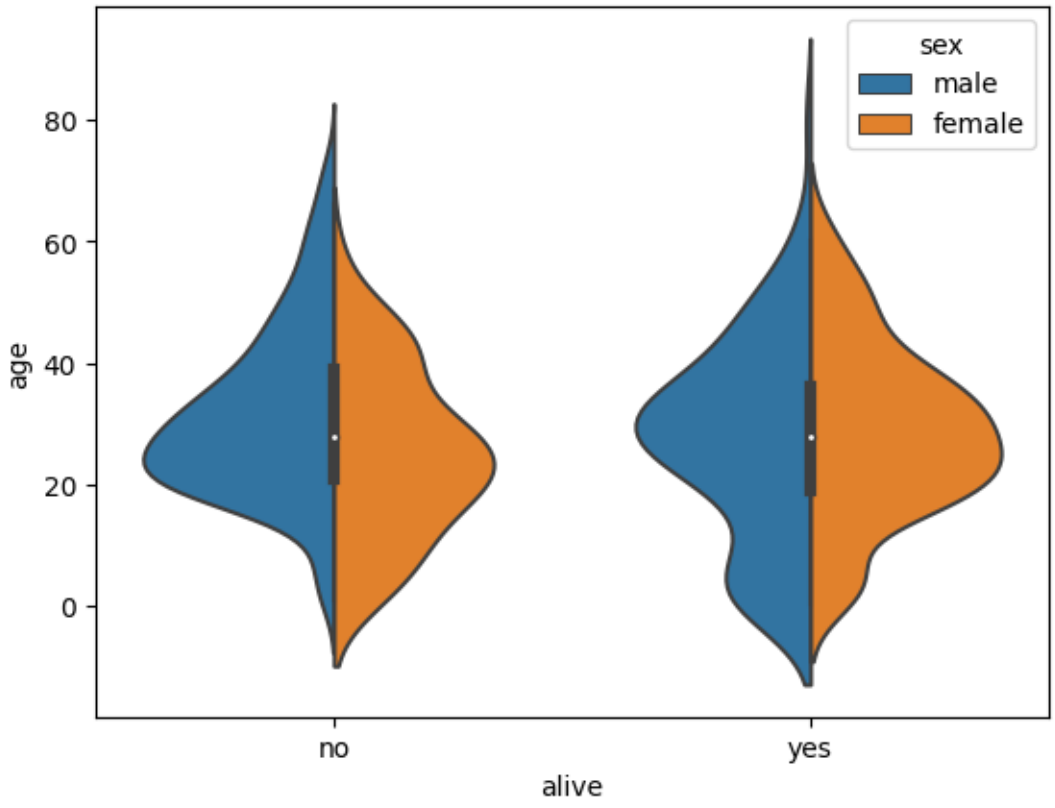
[6]:
```
<seaborn.axisgrid.FacetGrid at 0x15b822eecb0>
```




### 9.4.4 Gráficos de Violin

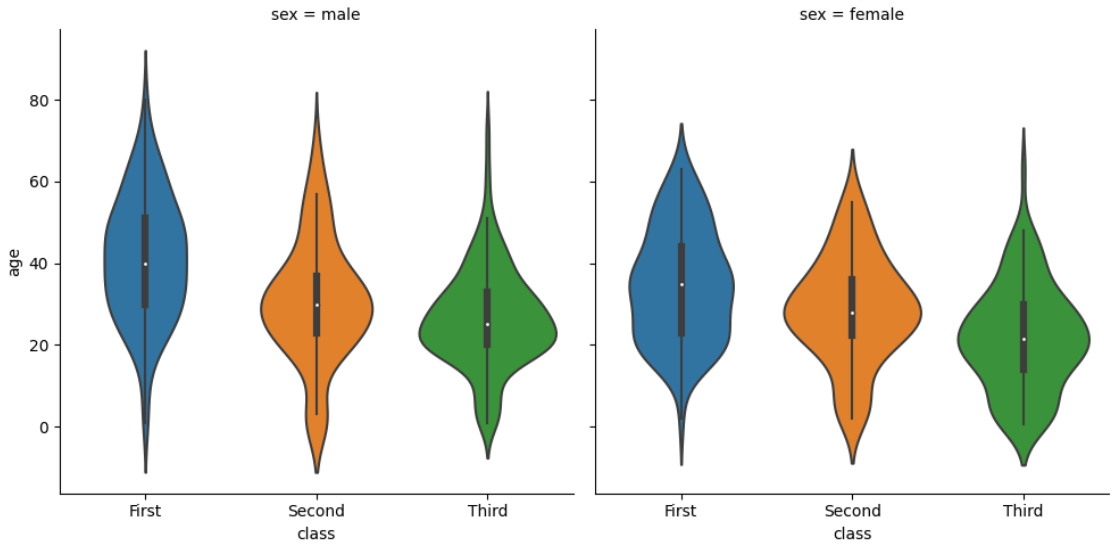
[3]:
```
sns.violinplot(data = df, x = 'alive', y = 'age', split = True, hue = 'sex')
```

[3]:
```
<Axes: xlabel='alive', ylabel='age'>
```



[4]:
```
sns.catplot(data = df, x = 'class', y = 'age', col = 'sex', kind = 'violin')
```

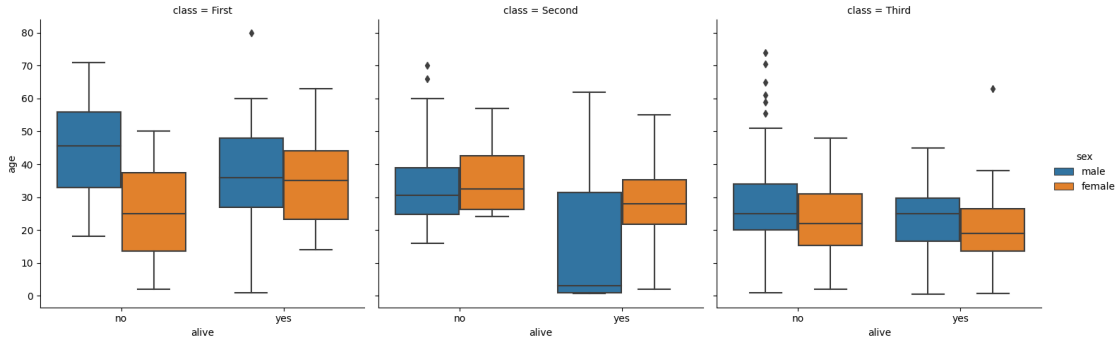
[4]: <seaborn.axisgrid.FacetGrid at 0x15b7feddf90>




### 9.4.5 Gráficos de caja

[5]:
```
sns.catplot(data = df, x = 'alive', y = 'age', col = 'class', hue = 'sex', kind␣
↪= 'box')
```

[5]: <seaborn.axisgrid.FacetGrid at 0x15b8223e9e0>



### 9.4.6 Ejercicios

- Elige un dataset propio o de Internet y prueba a realizar gráficas con las funciones **catplot**, **violinplot**, **heatmap**, **pairplot**, **jointplot**, ...

Mira la documentación para ver más gráficas interesantes

# 10 Entornos virtuales

Un entorno virtual es una computadora simulada en donde podemos instalar librerías sin afectar a nuestra computadora u otros entornos virtuales.

En Python existen al menos 2 formas de crear y gestionar un entorno virtual:

- **conda**
- **venv**

**conda** es un gestor de paquetes y de entornos de software libre usable en Windows, Linux y Mac.

**venv** es un librería nativa de python que permite crear entornos de forma rápida y fácil.

## 10.1 Creación entorno

### 10.1.1 Desde cero

```
[ ]: %zpython -m venv <env-name> --prompt="<alias>''
     %conda create --name <env-name>
```

### 10.1.2 Desde archivo

```
[ ]: %python -m venv <env-name> --prompt="<alias>"
     %python -m pip install -r requirements.txt

     %conda env create -f environment.yml
```

## 10.2 Activar entorno

```
[ ]: %<env-name>\Scripts\Activate
     %conda activate snowflakes
```

## 10.3 Instalar paquete

```
[ ]: %pip install <name>
     %conda install <name>
```

## 10.4 Desintalar paquete

```
[ ]: %pip uninstall <name>
     %conda remove <name>
```

## 10.5 Desactivar entorno

```
[ ]: %deactivate
     %conda deactivate
```

### 10.6 Borrar entorno

[ ]:
```
%rm -r <evn-name>
%conda remove -n <env-name> --all
```

### 10.7 Guardar configuración

[ ]:
```
%python -m pip freeze > requirements.txt
%conda env export > environment.yml
```

# 11 Bibliografía

## 11.1 Canales de YouTube

- Indently: https://www.youtube.com/@Indently
- Píldoras informáticas: https://www.youtube.com/watch?v=G2FCfQj-9ig&list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS

## 11.2 Documentación de Python

- https://docs.python.org/3/
- Funciones Built-in: https://docs.python.org/es/3.11/library/functions.html

## 11.3 Librerias usadas

- openpyxl: https://openpyxl.readthedocs.io/en/stable/index.html
- numpy: https://numpy.org/doc/stable/
- pandas: https://pandas.pydata.org/docs/
- matplotlib: https://matplotlib.org/
- seaborn: https://seaborn.pydata.org/index.html
- csv: https://docs.python.org/3/library/csv.html
- json: https://docs.python.org/es/3/library/json.html
- yaml: https://pyyaml.org/wiki/PyYAMLDocumentation
- rasterio: https://rasterio.readthedocs.io/en/stable/
- pprint: https://docs.python.org/3/library/pprint.html

## 11.4 Contenido Extra

- El libro de Python: https://ellibrodepython.com/
- Otras librerías para Ciencia: https://devopedia.org/python-for-scientific-computing

## 11.5 Juegos para aprender

- NotNiNi: https://app.notnini.com/
- CheckiO: https://checkio.org/

## 11.6 Referencias:

- ¿Qué es un código?: https://piperlab.es/glosario-de-big-data/codigo/
- Sintáxis de una variable: https://realpython.com/python-variables/
- Nombres: http://www.alan-g.me.uk/tutor/spanish/tutname.htm\T1\textgreater{}