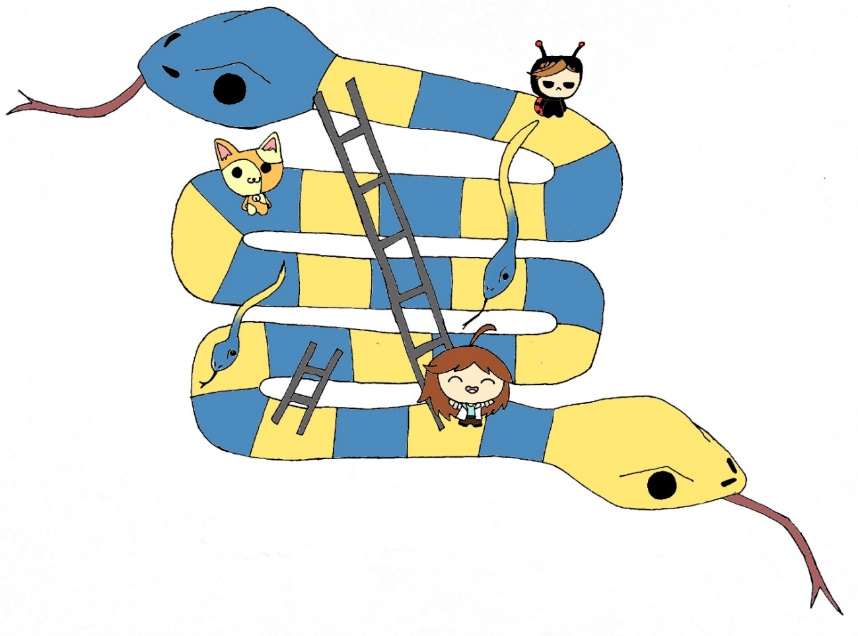
Programación para niñ@s



Aprende a programar con Python

Pello Xabier Altadill

June Altadill

<http://pello.io/mocos>

Prólogo

Este libro es una guía para aprender a programar. Para ello utiliza las herramientas más simples posibles, un lenguaje fácil y util como Python y la posibilidad de poder empezar a escribir programas desde el primer capítulo. Se empieza desde lo más fácil y se van introduciendo nuevas técnicas poco a poco.

Pero debes tener algo muy claro. Si te limitas a leer, puede que te quedes con alguna idea pero **no aprenderás** a programar de verdad. Para convertirte en un *pro* tienes que tener el ordenador delante y practicar lo que te explicamos aquí. Tienes que intentar hacer los ejercicios propuestos, escribirlos, probarlos y mejorarlos a tu gusto. Porque **a programar se aprende programando**.

Ponemos a tu disposición este libro, los ejercicios resueltos e incluso una página web donde lo único que tienes que hacer es entrar y ya podrás jugar con las prácticas que te proponemos:

<http://pello.io>/mocos

Índice de contenidos

[1 Programación para niñ@s 8](#_Toc73790886)

[2 El entorno 11](#_Toc73790887)

[3 Hola mundo 14](#_Toc73790888)

[3.2 Comentarios 16](#_Toc73790889)

[4 Variables 18](#_Toc73790890)

[4.1 Mostrar variables por pantalla 20](#_Toc73790891)

[5 Tipos de datos 24](#_Toc73790892)

[5.1 Números 25](#_Toc73790893)

[5.2 Texto 26](#_Toc73790894)

[5.3 Booleanos 28](#_Toc73790895)

[5.4 Listas 29](#_Toc73790896)

[5.5 None 30](#_Toc73790897)

[6 Leyendo datos 33](#_Toc73790898)

[6.2 Cuidado con los datos 34](#_Toc73790899)

[6.3 Otras conversiones 37](#_Toc73790900)

[7 Operadores 39](#_Toc73790901)

[7.1 Operadores aritméticos 39](#_Toc73790902)

[7.2 Operadores de comparación 44](#_Toc73790903)

[7.3 Operadores booleanos 46](#_Toc73790904)

[7.4 Ejercicios propuestos 53](#_Toc73790905)

[8 Condiciones 58](#_Toc73790906)

[8.1 if 59](#_Toc73790907)

[8.2 if else 61](#_Toc73790908)

[8.3 if elif else 63](#_Toc73790909)

[8.4 Ejercicios propuestos 67](#_Toc73790910)

[9 Bucles 75](#_Toc73790911)

[9.1 Bucle while 76](#_Toc73790912)

[9.2 Bucle For 79](#_Toc73790913)

[9.3 ¿Cuándo usar while o for? 88](#_Toc73790914)

[9.4 Ejercicios propuestos 90](#_Toc73790915)

[10 Estructuras de datos 99](#_Toc73790916)

[10.1 Listas 99](#_Toc73790917)

[10.2 Extraer partes de la lista 101](#_Toc73790918)

[10.3 Añadir y eliminar elementos 102](#_Toc73790919)

[10.4 Diccionarios 105](#_Toc73790920)

[10.5 Estructuras de datos combinadas 110](#_Toc73790921)

[11 Textos 114](#_Toc73790922)

[11.1 Los textos son listas 114](#_Toc73790923)

[11.2 Mayúsculas/Minúsculas 116](#_Toc73790924)

[11.3 split: de texto a lista 116](#_Toc73790925)

[11.4 Búsquedas 117](#_Toc73790926)

[11.5 Eliminar sobrantes 118](#_Toc73790927)

[11.6 Ejercicios propuestos 120](#_Toc73790928)

[12 Funciones 133](#_Toc73790929)

[12.2 Parámetros 135](#_Toc73790930)

[12.3 Retorno 139](#_Toc73790931)

[12.4 Llamadas anidadas 142](#_Toc73790932)

[12.5 ¿Por qué usar funciones? 145](#_Toc73790933)

[12.6 Cómo hacer buenas funciones 149](#_Toc73790934)

[12.7 Ejercicios propuestos 150](#_Toc73790935)

[13 Clases 163](#_Toc73790936)

[13.1 Cómo crear clases 164](#_Toc73790937)

[13.2 Clase vs instancia 165](#_Toc73790938)

[13.3 Función constructora 167](#_Toc73790939)

[13.4 Herencia 169](#_Toc73790940)

[13.5 Encapsulación 172](#_Toc73790941)

[13.6 Clases que contiene otras clases 177](#_Toc73790942)

[13.7 Métodos estáticos 180](#_Toc73790943)

[13.8 Ejercicios propuestos 183](#_Toc73790944)

[14 Excepciones 203](#_Toc73790945)

[14.1 Excepciones en Python 204](#_Toc73790946)

[15 Manejo de ficheros 207](#_Toc73790947)

[15.1 Lectura de ficheros 208](#_Toc73790948)

[15.2 Ficheros JSON 209](#_Toc73790949)

[15.3 Escritura de ficheros 211](#_Toc73790950)

[15.4 Escribir en un fichero json 213](#_Toc73790951)

[16 Librerías 214](#_Toc73790952)

[16.2 Ejercicios propuestos 221](#_Toc73790953)

[17 Sobre Python 235](#_Toc73790954)

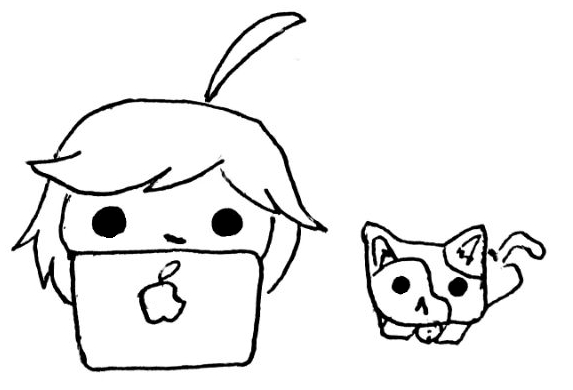
[17.1 Instalando Python localmente 236](#_Toc73790955)

[17.2 Editores de código 236](#_Toc73790956)

[17.3 Test unitarios 237](#_Toc73790957)

[17.4 Iniciando un proyecto Python localmente 240](#_Toc73790958)

# Programación para niñ@s



Los ordenadores, son rápidos y tienen una memoria enorme: son máquinas capaces de ejecutar millones de instrucciones por segundo y de gestionar cantidades inimaginables de datos. Pero eso no significa que sean listos. De hecho no saben hacer nada por sí solos. Para que un ordenador, o una *tablet*, un móvil, hagan algo concreto, deben ejecutar un programa. Y un programa es un conjunto de órdenes escritas por los programadores.

Lo interesante de la programación es que podemos tomar el control de la máquina para que ésta haga lo que nosotros queramos: desde una instrucción simple, hasta programás muy complejos como navegadores o juegos. La programación engancha porque puedes crear lo que quieras, sin más límite que tu imaginación y tus habilidades.

Pero ¿en qué lenguaje nos podemos comunicar con los ordenadores? Existen infinidad de ellos. Internamente, el ordenador utiliza el lenguaje binario, es decir, una secuencia de unos y ceros que es capaz de interpretar para hacer algo. Sin embargo, para facilitar la tarea de programación, hay lenguajes más simples, que se parecen mucho al lenguaje de las personas.

Python es uno de esos lenguajes, y tiene muchas virtudes: es sencillo, muy fácil de aprender, permite hacer cualquier cosa, dispone de infinidad de librerías y encima se utiliza de manera profesional. En este libro aprenderás a programar utilizando Python. De forma progresiva, irás conociendo nuevas herramientas que te ofrece el lenguaje para aprender a crear programas más complejos.

¿Qué necesitas para empezar ya mismo? Un navegador. Desde el primer capítulo nos dedicaremos a programar. La mejor forma de aprender a programar es... ¡programando! No hay que tener ningún miedo, y además, es lo más divertido. Así que basta de cháchara.¡Pasa al siguiente capítulo para escribir tu primer programa!

Nota:

Quizá hayas oído que los ordenadores pueden tener inteligencia artificial o puede que te hayas enfrentado a juegos en los que los enemigos parecen ser muy listos. En realidad, lo que hace el ordenador es ejecutar programas que imitan comportamientos inteligentes.

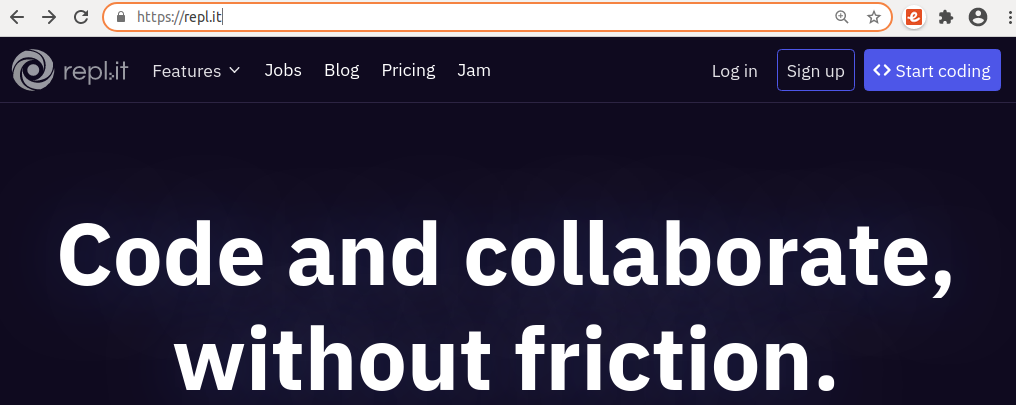
Nota:

Si quieres instalarte un entorno de programación convencional, ve al capítulo 17.

# El entorno

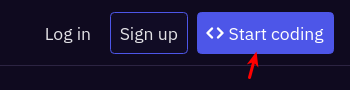
En cualquier ordenador con conexión a Internet, accede a:

https://repl.it/



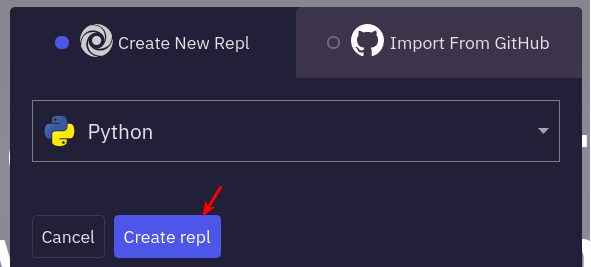
Site de replt.it

Desde ahí puedes pulsar el botón Start coding:



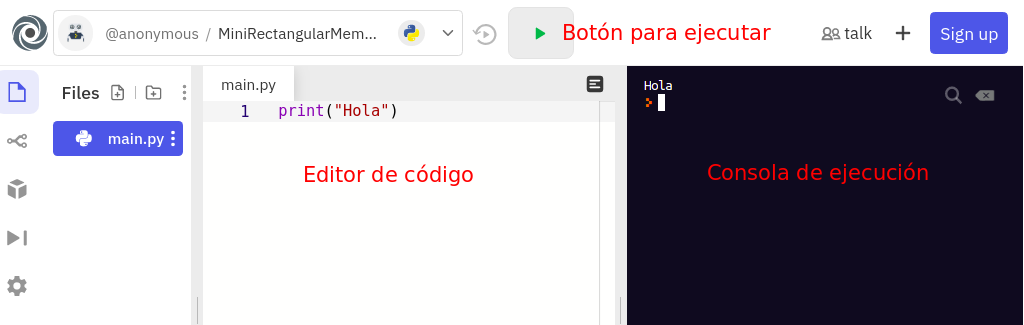
Start Coding en replt.it

Y seleccionar el lenguaje **Python**:



Selección de Python en replt.it

Una vez hecho, se cargará el entorno de programación:



Entorno de desarrollo de replt.it

* A la izquierda, tienes el editor donde puedes escribir el programa
* A la derecha, la consola, donde verás el resultado cuando ejecutes el programa
* En la parte superior, el botón **Run** con el que podrás ejecutar el programa tantas veces como quieras.

Te recomendamos que te des de alta en el sitio repl.it. De esa manera podrás tener guardados y localizados todos los programas que vayas haciendo.

Otras alternativas online:

* https://paiza.io/es
* https://www.programiz.com/python-programming/online-compiler/

# Hola mundo



El primer programa que suelen escribir los programadores es uno que simplemente saca un mensaje por pantalla. Y ese mensaje suele ser un saludo al mundo: "¡Hola mundo!" Se hace así:

print("¡Hola mundo!")

Si pruebas esto deberías ver por la pantalla algo así:

¡Hola mundo!

print es una función del lenguaje Python que nos permite mostrar mensajes por pantalla y la utilizaremos a menudo para mostrar mensajes, resultados, etc.



¡OJO!

En los programas muy simples como los de este capítulo, intenta no poner espacios antes de las instrucciones del programa, o Python dará un error:

print("¡Hola mundo!")

Resultaría en error:

print("¡Hola mundo!")  
 ^  
IndentationError: unexpected indent

En Python solo se añaden espacios o tabulaciones para indicar que el código está dentro de otros bloques, como irás viendo poco a poco. De momento, para este capítulo, empieza tu código desde el principio de la línea.



### ¡Te toca! Ejercicio 0.0

Escribe un programa que muestre por pantalla tu nombre.

print("Hola soy Ada")

Resultado:

Hola soy Ada

¡OJO!

En el lenguaje Python es muy importante que no utilices tabulaciones o espacios al principio de la línea (salvo que utilices bloques que veremos en los siguientes capítulos). Esto, daría un error:

print("¡Hola mundo!")

## Comentarios

En un programa, se pueden poner comentarios. Se trata de texto que no se ejecuta y que el ordenador ignora completamente. ¿para qué se utiliza? Generalmente los comentarios se utilizan para explicar determinadas partes del programa.

# Este programa dice Hola  
print("Hola")

Python ignora el comentario y mostrará Hola por pantalla. También se pueden hacer comentarios de varias líneas:

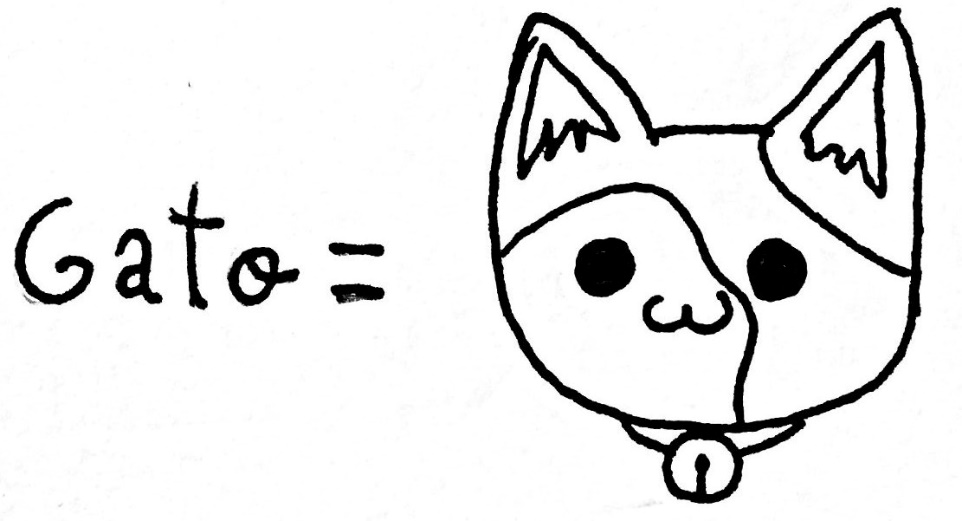
"""  
Este es un programa Python  
creado por Ada  
y revisado por Neko  
"""

A veces, los comentarios se utilizan de manera temporal para "desactivar" una parte del código que no queremos que se ejecute.

Nota:

En general tienes que evitar los comentarios. Un buen programador tiene que intentar escribir programas tan fáciles de entender que no necesiten comentario alguno.

# Variables



Las variables sirven para guardar datos. Los programas de ordenador se dedican, básicamente, a manejar datos para solucionar un problema y ofrecer un resultado. En todo ese proceso es necesario guardar datos, y para eso se utilizan las variables. Las variables son como contenedores de datos. En cierto modo es como los vasos, platos, fuentes que se utilizan para cocinar: contienen algo, se trabajo con ello, se mezcla, se procesa y se consigue un resultado: con un poco de suerte algo rico.

Para definir una variable en python basta con indicar su nombre y darla algún valor. Por ejemplo:

nombre = "Ada"

Acabamos de crear una variable que contiene el valor "Ada". "Ada" es un dato, y es de tipo texto. Ahora podemos mostrar el valor de esa variable por la pantalla:

print(nombre)

Que por pantalla sería:

Ada

En cualquier momento, podríamos cambiar el valor de esa variable:

nombre = "Ada"  
print(nombre)  
nombre = "Neko"  
print(nombre)

Y por pantalla veríamos:

Ada  
Neko



**Nota:**

También se puede mostrar el contenido de una variable como parte del mensaje. Existen varias opciones que veremos a continuación.

## Mostrar variables por pantalla

### Separar por comas

Basta con intercalar las variables y el texto con comas:

nombre = "Bug"  
edad = 10  
print("Hola, me llamo", nombre)  
print("Soy", nombre, "tengo", edad, "años")

Que por pantalla sería:

Hola, me llamo Bug  
Soy Bug tengo 10 años

### ¡Ahora tú! Ejercicio 0.1

Crea dos variables nombre y edad y muestra su valor por pantalla.

nombre = "Ada"  
edad = 14  
print("Tu nombre es", nombre, ", tienes", edad, "años")  
  
# Alternativa:  
# print("Tu nombre es %s, tienes %d años" % (nombre, edad))

Resultado:

Tu nombre es Ada y tienes 14 años.

### String de formato

Otra forma de mostrar varias variables es con un mensaje que precedido de la letra f y las variables entre llaves:

nombre = "Bug"  
edad = 10  
print(f"Hola, me llamo {nombre}")  
print(f"Soy {nombre} tengo {edad} años")

Que por pantalla sería:

Hola, me llamo Bug  
Soy Bug tengo 10 años



¡OJO!

Ésta opción solo está disponible desde Python 3.6

### Sustitución de porcentaje

Otra alternativa para mostrar variables por pantalla: se crea un mensaje donde los elementos %s son sustituidos por variables.

print("Hola, me llamo %s" % nombre)

Que por pantalla sería:

Hola, me llamo Ada

Puedes hacer lo mismo con varias variables

nombre = "Neko"  
edad = 5  
  
print("Hola, me llamo %s y tengo %d años" % (nombre, edad))

Que por pantalla sería:

Hola, me llamo Neko y tengo 5 años

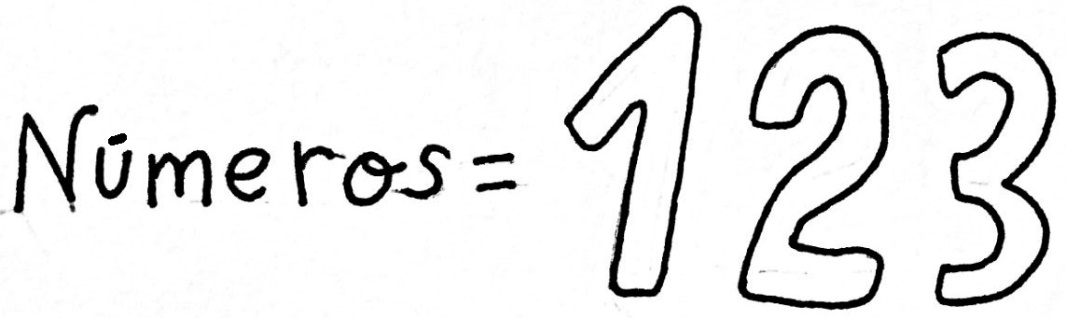
# Tipos de datos

¡Datos!

Es la materia prima con la que trabajan los programas. Son el elemento que nuestros programas transforman. Un programa recibe datos, los transforma y los devuelve como un resultado. Los datos pueden ser de distinto tipo, según lo que nuestro programa tenga que hacer. Pueden ser números, puedes ser palabras o textos, pueden incluso ser nulos o vacíos. Para guardar los datos, generalmente usamos variables. Si comparamos la programación con la cocina, entonces el azúcar, la harina y los huevos serían datos, los recipientes serían variables: la tarta sería otro dato, el resultado y la receta sería el programa. ¿Cómo sabe Python que tipo de dato maneja? No hace falta indicárselo como en otros lenguajes. Por eso es un lenguaje más simple y flexible. Aunque tampoco podremos hacer lo que nos de la gana con los datos.

A continuación, vamos a ver los tipos básicos de datos.

## Números



Se trata de todos los tipos de números:

* **Enteros**: 1, 2, 3, 4,...

contador = 10  
edad = 12

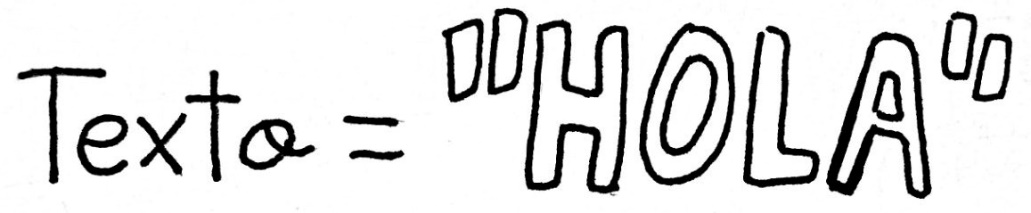
* **Con decimales**: Los números con decimales se expresan utilizando un . para separar la parte entera de la decimal, como 4.5 o 3.1415. Es probable que en clase de mates utilices una coma para separar las decimales, pero en programación se usa el formato inglés y debemos usar un ..

peso = 34.67  
precio = 242.9943

* **Negativos**: Los números menores que 0 se expresan con un guión por delante: -4, -5, -3.1415,...

nota = -5  
temperaturaEnMarte = -50.676

## Texto



El texto, también llamados cadenas o *strings* son cualquier conjunto de palabras entre comillas dobles o simples.

nombre = "Ada"  
frase = ""  
palabras = 'Voy a trabajar'  
novia = 'Solo quiero que seamos "amigos"'

En el caso del texto, podemos meter una serie de caracteres especiales que nos permiten efectos interesantes. Esos caracteres se escriben con una contra-barra o *backslash* por delante: \

* **Salto de linea** Esto añade un salto de línea al texto si este se muestra por pantalla:

frase = "Hola,\n qué tal"

se mostrará así:

Hola,  
qué tal

También se puede definir un texto de varias líneas así:

cancion = """Era un domingo  
a la tarde  
fui a los coches   
de choque"""

* **Tabulaciones** Esto añade una tabulación (varios espacios) al texto si este se muestra por pantalla:

frase = "Nombre\tApellido\tEdad"

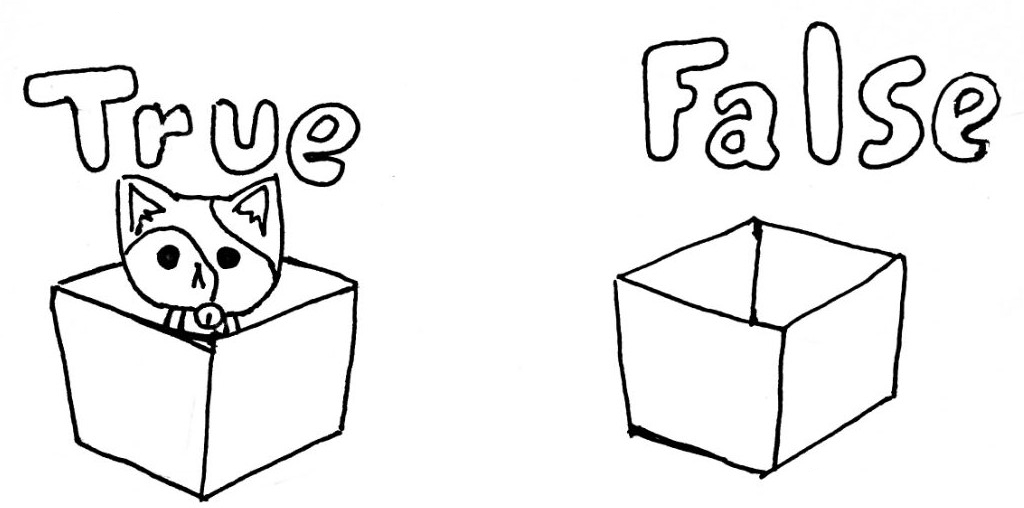
se mostrará así:

Nombre Apellido Edad

Otros caracteres especiales:

* \\ Para mostrar la contra-barra en un texto.
* \" Para mostrar una comilla doble en un texto.
* \' Para mostrar una comilla simple en un texto.
* \a Para hacer sonar un pitido.

## Booleanos



El tipo booleano solo puede tener dos valores posibles: True o False, es decir verdadero o false. Se trata de un tipo de dato fundamental en programación ya que se utiliza para tomar decisiones.

terminado = False  
esMayor = True  
pythonMola = True

## Listas

Las listas son conjuntos de datos, que se definen de la siguiente manera:

amigas = ["Ada", "Miranda", "Ruby"]

Pueden ser de cualquier tipo, pero lo normal es que todos los elementos de una lista sean del mismo tipo:

enemigas = [] # lista vacía  
edades = [12, 16, 30, 0, 22, 1, 1, 12]  
verdades = [True, False, False, True]

Para poder referirnos a un valor concreto de la lista, tenemos que indicar la posición del elemento de la lista que nos interesa, empezando desde 0:

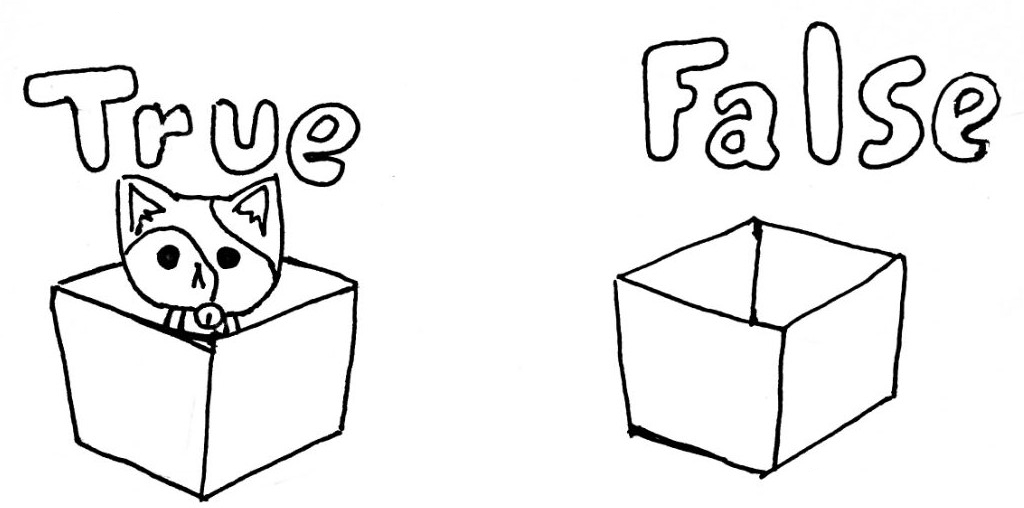
nombres = ["Ada", "Neko", "Bug"]  
print(nombres[0]) # "Ada"

En el caso de la lista nombres, las posiciones posibles serán 0, 1 y 2. ¡Pero ojo! si pasas una posición demasiado grande, el programa terminará con error:

nombres = ["Ada", "Neko", "Bug"]  
print(nombres[4]) # ¡ERROR!

Volveremos sobre las listas y otras estructuras más adelante.

## None



Aunque suene un poco raro, en los programas a veces hay que tratar con algo que representa el vacío, la nada. Existe una palabra que nos permite representar la nada en Python, y esa es: None

valorInicial = None  
dato = None

En realidad no se suele utilizar para crear variables. None representa un valor en situaciones especiales. Por ejemplo, si se trata de sacar información de un sitio en el que no hay nada como un fichero, o un dato que el usuario no nos da.



### Ejercicio 0.2

Escribe un programa que defina una variable de cada tipo visto aquí, y los muestre por la consola.

nombre = "Ada"  
edad = 42  
peso = 101.54  
vivo = True  
riquezas = None  
amigas = ["Ada", "Ruby", "Miranda"]  
  
print(nombre)  
print(edad)  
print(peso)  
print(vivo)  
print(riquezas)  
print(amigas)

Resultado:

Ada  
42  
101.54  
true  
null  
["Ada", "Ruby", "Miranda"]

# Leyendo datos

Para que un programa pueda hacer algo, muchas veces necesita que el usuario introduzca un dato. Por ejemplo, si queremos que un programa nos diga cuántas letras tiene nuestro nombre, o cuánto falta para nuestro cumpleaños, lo primero que tendrá que hacer el programa es pedir un dato.

Los programas básicos como los que estamos viendo de momento, utilizan la consola para ejecutarse. Son esas pantallas negras donde se ponen órdenes escritas ;)

Para poder pedirle al usuario un dato y guardarlo en una variable, se utiliza la función input:

nombre = input("Dime tu nombre: ")  
print("Hola ", nombre)

En la pantalla verías lo siguiente:

Dime tu nombre:

La función input hace que el mensaje Dime tu nombre: aparezca en pantalla. También hace que el programa se pare a la espera de que el usuario escriba algo. Si el usuario escribe Rosa en la consola, se vería:

Dime tu nombre: Rosa  
Hola Rosa

### Ejercicio 0.3

Escribe un programa que solicite un nombre al usuario y lo guarde en una variable. A continuación debe mostrar un saludo por consola.

nombre = input("Introduce tu nombre: ")  
print("Hola, qué tal estás ", nombre)  
  
# Alternativa:  
# print("Hola, qué tal estás %s" % nombre)

Resultado:

Introduce tu nombre: Juan  
Hola, qué tal estás Juan

## Cuidado con los datos

Cada vez que uses la función input para que el usuario escriba algo, sea lo que sea **se guardará como texto**. Aunque se escriba un número:

valor = input("Dame un número: ")  
doble = valor + valor  
print(doble)

Si el usuario introduce un número como 4 este sería el resultado:

Dame un número: 4  
44

En lugar de sumar 4 + 4 y mostrar 8, lo que ha hecho es unir 4 y 4, porque en realidad, cuando se ha leído a través de input ese 4 es texto: "4"

Para evitarlo, tenemos que usar otra función para convertir ese dato en número entero: int()

valor = input("Dame un número: ")  
doble = int(valor) + int(valor)  
print(doble)

O incluso lo podemos convertir antes:

valor = input("Dame un número: ")  
valor = int(valor)  
doble = valor + valor  
print(doble)

O incluso aplicarlo a input:

valor = int(input("Dame un número: "))  
doble = valor + valor  
print(doble)

Ahora sí, la suma será numérica:

Dame un número: 4  
8



### Ejercicio 0.4

Escribe un programa que solicite un número al usuario y le sume 10. A continuación debe mostrar el resultado por la consola. Recuerda que conviene hacer un int del valor introducido para convertirlo a número entero.

valor = input("Introduce un número: ")  
resultado = int(valor) + 10  
  
print("La suma es:", resultado)

Resultado:

Introduce un número: 32  
La suma es: 42

## Otras conversiones

Aunque en el lenguaje no tengamos que declarar los tipos de variables, los tipos (texto, número, etc...) sí se tienen en cuenta a la hora de ejecutarse el programa.

Por ejemplo si tenemos una variable que contiene un número, y la queremos concatenar en un texto, obtendríamos un error:

valor = 66  
texto = "Mi edad es " + valor

El error sería:

TypeError: cannot concatenate 'str' and 'int' objects

Para evitarlo, tenemos que forzar el tipo a texto con str():

valor = 66  
texto = "Mi edad es " + str(valor)

Por tanto, hay ocasiones en los que tendremos que convertir un valor en determinado tipo. Las funciones para convertir son las siguientes:

* str(): convierte un valor a texto. str(5) devolvería "5".
* int(): convierte un valor a número entero. int("5") devolvería 5.
* float(): convierte un valor a número con decimales. float("5.5") devolvería 5.5
* bool(): convierte un valor a un booleano. bool("False") devolvería False.

¡OJO!

Si tratas de convertir un valor que no es compatible, el programa fallará y terminará de golpe.

# Operadores

Los programas necesitan hacer cálculos con números, procesar datos, tomar decisiones según los valores. Para ello necesitamos los operadores.

## Operadores aritméticos

Son todos aquellos que te permitirán hacer sumas, restas y todas los cálculos básicos con valores y con aquello que se guarde en variables, como por ejemplo, la suma:

chicles = 4  
chicles = chicles + 2

El programa calcula que de tener 4 chicles, has pasado a tener 6. Las operaciones básicas en programación son:

* suma: +
* resta: -
* multiplicación: \*
* división: /

Por ejemplo para calcular el total de segundos que tiene un día:

minutos = 60  
segundos = 60  
horas = 24  
  
totalSegundos = segundos \* minutos \* horas

Puedes hacer operaciones tan complejas como hagan falta. Para que estas sean más fáciles de leer se pueden utilizar paréntesis como se hace en mates:

ada = 14  
bug = 10  
neko = 2  
media = (ada + bug + neko) / 3

### Ejercicio 0.5

Escribe un programa que solicite un número al usuario y le reste 5. A continuación debe mostrar el resultado por la consola.

valor = input("Introduce un número: ")  
resultado = int(valor) - 5  
  
print("La resta es:", resultado)

Resultado:

Introduce un número: 30  
La resta es: 25

### Módulo y exponencial

Hay una operación muy importante en programación, que quizá no sea tan frecuente en tu clase de mates: se trata del **módulo**. Es una división que devuelve el residuo en lugar del resultado de la división:

valor = 8  
resultado = valor % 3

El valor de resultado será: 2.

El exponencial es el resultado de multiplicar un número por sí mismo varias veces. En Python se puede hacer esta operación con el operador \*\*:

valor = 2  
resultado = valor \*\* 3 # equivale a: 2 \* 2 \* 2

El resultado sería 8.

### Cambio de signo

Como bien sabes, hay números menores que cero llamados negativos, que se representan con un - por delante:

-5, -248, -1.87, ...

Si queremos cambiar el signo de un número podemos poner un - por delante:

temperatura = -11  
cuenta = 200  
  
temperatura = -temperatura # 11  
cuenta = -cuenta # -200

### Operadores abreviados

En muchas ocasiones, tendrás que operar sobre una variable y guardar el resultado en la propia variable:

contador = 0  
contador = contador + 2

En ese tipo de situaciones, puedes usar un operador **abreviado**, el cual hace la operación y asigna al mismo tiempo. Esto sería equivalente al anterior código:

contador = 0  
contador += 2

Lo mismo se puede hacer con todos los operadores:

|  |  |  |
| --- | --- | --- |
| Operación | Es lo mismo que | Abreviada |
| a = a + 1 |  | a += 1 |
| a = a - 1 |  | a -= 1 |
| a = a \* 1 |  | a \*= 1 |
| a = a / 1 |  | a /= 1 |
| a = a % 1 |  | a %= 1 |



### Ejercicio 0.6

Escribe un programa que solicite un número al usuario y lo incremente. A continuación debe mostrar el resultado por la consola. Luego debe decrementar el valor y mostrar el resultado por consola.

¡Utiliza operadores abreviados!

valor = input("Introduce un número: ")  
valor += 1  
  
print("El incremento es", valor)  
  
valor -= 1  
  
print("El decremento es", valor)

Resultado:

Introduce un número: 6  
El incremento es 7  
El decremento es 6

## Operadores de comparación

Se trata de operadores que nos permiten comparar un valor con otro. Generalmente se usa con números y el resultado de estas operaciones es Trueo False.

Por ejemplo, para comprobar si un valor es igual a otro usamos el operador: ==

valor = 5  
resultado = valor == 5

El resultado sería True.

Los operadores de comparación serían:

* Igual: ==
* Distinto: !=
* Mayor que: >
* Menor que: <
* Mayor o igual: >=
* Menor o igual: <=

También puede utilizarse este operador con texto, tanto para comprobar la igualdad:

nombre = "Ada"  
resultado = nombre == "Bug"

El resultado sería False.

También nos permite comparar si un texto es mayor o menor en orden alfabético:

nombre = "Ada"  
resultado = "Ada" < "Bug"

El resultado sería True.



### Ejercicio 0.7

Escribe un programa que solicite dos números al usuario. Después debe comparar su desigualdad y debe mostrar el resultado por la consola.

valor1 = input("Introduce un número: ")  
valor2 = input("Introduce otro número: ")  
  
resultado = valor1 != valor2  
  
print("¿Son distintos?", resultado)

Resultado:

Introduce un número: 42  
Introduce otro número: 42  
¿Son distintos? False

## Operadores booleanos

Los operadores booleanos nos permiten hacer operaciones con valores booleanos Trueo False.

### and

Este operador solo devuelve True si los dos operandos también son True:

valor = 5  
resultado = (valor == 5) and True;

El resultado sería True. Para resumir todas las posibles opciones, esta sería lo que se denomina tabla de la verdad del operador and.

|  |  |  |  |
| --- | --- | --- | --- |
| a |  | b | resultado |
| False | and | False | False |
| False | and | True | False |
| True | and | False | False |
| True | and | True | True |



### Ejercicio 0.8

Escribe un programa que solicite un número al usuario. Después debe comparar si el primero es mayor que 0 y además es par.

valor = input("Introduce un número: ")  
valor = int(valor)  
resultado = (valor >= 0) and (valor % 2 == 0)  
  
print("¿Es par y positivo?", resultado)

Resultado:

Introduce un número: 14  
¿Es par y positivo? True

### or

Este operador devuelve True si **cualquiera** de los dos operandos también son True:

valor = 5  
resultado = (valor == 5) or True;

El resultado sería True. Para resumir todas las posibles opciones, esta sería la tabla de la verdad del operador or:

|  |  |  |  |
| --- | --- | --- | --- |
| a |  | b | resultado |
| False | or | False | False |
| False | or | True | True |
| True | or | False | True |
| True | or | True | True |



### Ejercicio 0.9

Escribe un programa que solicite dos números al usuario y compruebe si alguno de los dos es positivo. A continuación debe mostrar el resultado por la consola.

valor1 = input("Introduce un número: ")  
valor2 = input("Introduce otro número: ")  
  
resultado = (int(valor1) >= 0) or (int(valor2) >= 0)  
  
print("¿Es alguno de los dos positivo?", resultado)

Resultado:

Introduce un número: -4  
Introduce otro número: 6  
¿Es alguno de los dos positivo? True

### not

Este operador devuelve el valor contrario al operando. Si se aplica a True devuelve False y si se aplica a False devuelve True:

valor = True  
resultado = not valor;

El resultado sería False. Para resumir todas las posibles opciones, esta sería la tabla de la verdad del operador not.

|  |  |  |
| --- | --- | --- |
|  | a | resultado |
| not | False | False |
| not | True | False |



### Ejercicio 0.10

Escribe un programa que solicite un número al usuario y compruebe que no es ni positivo ni par.

valor = input("Introduce un número: ")  
valor = int(valor)  
  
positivoYPar = (valor >= 0) and (valor % 2 == 0)  
resultado = not positivoYPar  
print("¿No es par y positivo?", resultado)

Resultado:

Introduce un número: -4  
¿No es par y positivo? True

### Combinando operadores

Podemos combinar los operadores tanto como necesitemos:

jubilado = 65  
if edad > 17 and edad < (jubilado + 1):  
 print("Ya puedes trabajar")

Generalmente, los operadores condicionales se utilizan dentro de las condiciones de los bloques condicionales, bucles, etc.

## Ejercicios propuestos

### Ejercicio 0.0

Escribe un programa que solicite un número al usuario y le multiplique 7. A continuación debe mostrar el resultado por la consola.

valor = input("Introduce un número: ")  
resultado = int(valor) \* 7  
  
print("La multiplicación es:", resultado)

Resultado:

Introduce un número: 3  
La multiplicación es: 21

### Ejercicio 0.1

Escribe un programa que solicite un número al usuario y lo divida por 2. A continuación debe mostrar el resultado por la consola.

valor = input("Introduce un número: ")  
resultado = int(valor) / 2  
  
print("La división es:", resultado)

Resultado:

Introduce un número: 60  
La división es: 30

### Ejercicio 0.2

Escribe un programa que solicite un número al usuario y haga módulo 3. A continuación debe mostrar el resultado por la consola.

valor = input("Introduce un número: ")  
resultado = int(valor) % 3  
  
print("El módulo es:", resultado)

Resultado:

Introduce un número: 7  
El módulo es: 1

### Ejercicio 0.3

Escribe un programa que solicite un número al usuario y le aplique un exponencial de 2. A continuación debe mostrar el resultado por la consola.

valor = input("Introduce un número: ")  
resultado = int(valor) \*\* 2  
  
print("El exponencial es:", resultado)

Resultado:

Introduce un número: 4  
El exponencial es: 16

### Ejercicio 0.4

Escribe un programa que solicite un número al usuario y le reste 5. A continuación debe cambiarle el signo y mostrar el resultado por la consola.

valor = input("Introduce un número: ")  
resta = int(valor) - 5  
resultado = -resta  
  
print("La resta final es:", resultado)

Resultado:

Introduce un número: 4  
La resta final es: 1

### Ejercicio 0.5

Escribe un programa que solicite un número al usuario. Después debe comprobar si la operación % 2 es igual a 0 y mostrar el resultado. Si se divide un número por 2 y la resta es 0, significa que ese número es par.

valor = input("Introduce un número: ")  
modulo = int(valor) % 2  
  
resultado = modulo == 0  
  
print("¿Valor es par?", resultado)

Resultado

Introduce un número: 8  
¿Valor es par? True

### Ejercicio 0.6

Escribe un programa que solicite un número al usuario. Después debes comprobar si ese número es mayor o igual que 0, es decir, positivo.

valor = input("Introduce un número: ")  
  
resultado = int(valor) >= 0  
  
print("¿Es positivo?", resultado)

Resultado:

Introduce un número: 6  
¿Es positivo? True

### Ejercicio 0.7

Escribe un programa que solicite un número al usuario. Después debe comparar si el primero es menor que 0 y mostrar el resultado por la consola. Estaríamos detectando si el número es negativo.

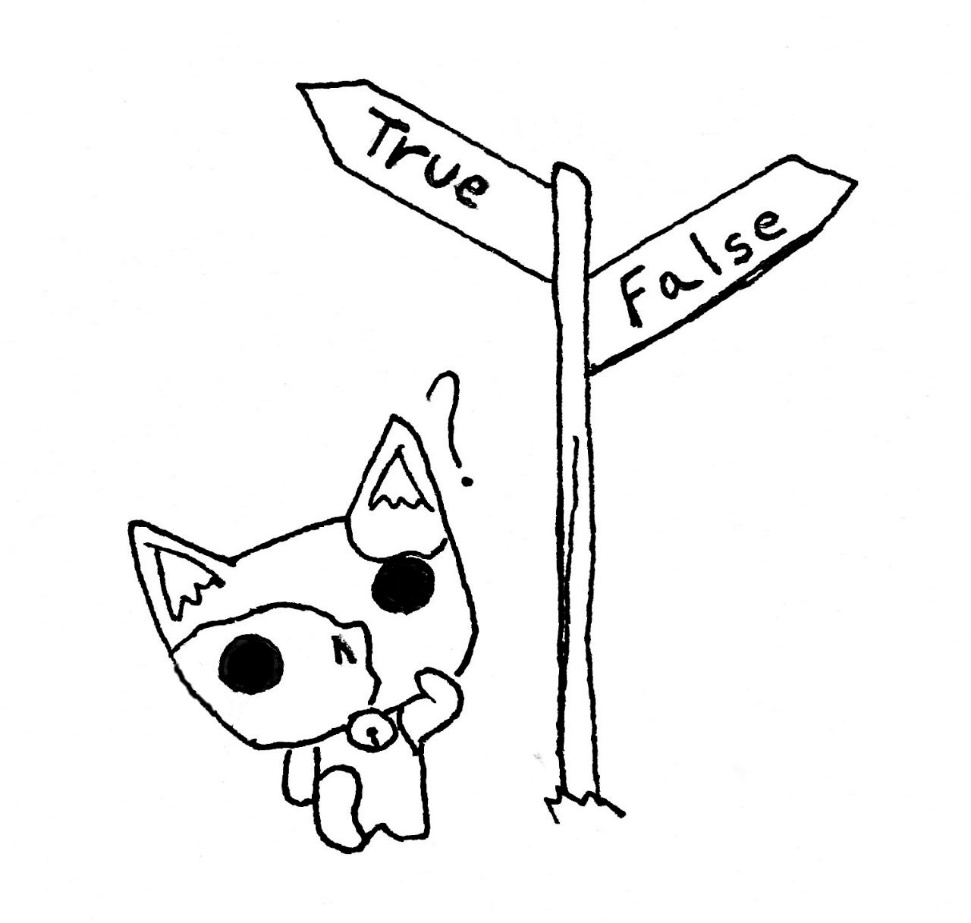
valor = input("Introduce un número: ")  
  
resultado = int(valor) < 0  
  
print("¿Es negativo?", resultado)

Resultado:

Introduce un número: -3

¿Es negativo? True

# Condiciones



En algún momento, los programas necesitan hacer una cosa u otra dependiendo de una condición. Por ejemplo, si un usuario introduce un dato incorrecto, el programa se acaba. Si un dato tiene determinado valor, se procesa de una forma y si no, de otra. ¿Cómo se consigue ese comportamiento? Mediante condiciones.

Las condiciones son estructuras de programación que nos permiten que un código se ejecute solo cuando se cumplan unas condiciones.

## if

La estructura más simple para hacer una condición es el if, el cual tiene este aspecto:

if \*condición\*:  
 \*instrucciones\*  
 \*instrucciones\*  
 \*...\*

Como puedes observar, if comienza con una condición. La condición puede ser cualquier expresión que devuelva un booleano, es decir, será True o False, verdadero o falso. Si es True, las instrucciones dentro del if se ejecutarán, y si no se saltarán. Por ejemplo:

valor = -2  
if valor < 0:  
 print("El valor es menor que 0")  
print("Fin del programa")

Resultaría en:

El valor es menor que 0  
Fin del programa

En cambio:

valor = 5  
if valor < 0:  
 print("El valor es menor que 0")  
print("Fin del programa")

Resultaría en:

Fin del programa

Nota:

También debes observar algo muy importante: las instrucciones dentro del if van detrás de unos espacios o una tabulación. Esa es una peculiaridad del lenguaje de programación Python: en cualquier bloque como una condición, un bucle, una función, su contenido debe ir tabulado. Esa es una forma que facilita la lectura y permite reconocer fácilmente la estructura de un programa para otros programadores. Incluso para ti mismo si es tu propio programa.



### Ejercicio 1.0

Escribe un programa que solicite un número al usuario y compruebe si es negativo. So es negativo, debe mostrar un mensaje por consola.

valor = input("Escribe un número: ")  
  
if int(valor) < 0:  
 print("Es negativo")

Resultado:

Escribe un número: -5  
Es negativo

## if else

Con el if podemos crear un bloque que solo se ejecute si se cumple una condición. Pero ¿Qué pasa si queremos que el programa haga una cosa *u otra* según una condición? Para poder meter "la otra" opción, utilizamos una estructura if-else:

if \*condición\*:  
 \*instrucciones\*  
else:  
 \*instrucciones\*

Por ejemplo:

nombre = input("Dime tu nombre: ")  
if nombre != "":  
 print("Hola ", nombre)  
else:  
 print("¡No has metido nada!")

Podría verse algo así, según lo que meta el usuario:

Dime tu nombre: Ada  
Hola Ada

Pero si el usuario simplemente pulsa enter sin escribir nada:

Dime tu nombre:  
¡No has metido nada!



### Ejercicio 1.1

Escribe un programa que solicite un texto al usuario. Si el texto es "saluda" debe mostrar un saludo, en caso contrario debe mostrar un mensaje que diga "no entiendo".

texto = input("Introduce un texto: ")  
  
if texto == "saludo":  
 print("Hola!")  
else:  
 print("No entiendo.")

Resultado:

Introduce un texto: no sé  
No entiendo.

## if elif else

Existe otra variante cuando necesitamos comprobar varias condiciones. Para eso existe la estructura if-elif-else:

if \*condición1\*:  
 \*instrucciones\*  
elif: \*condición2\*:  
 \*instrucciones\*  
elif \*condición3\*  
 \*instrucciones\*  
else:  
 \*instrucciones\*

Supongamos que queremos un programa que sea capaz de saludar en distintos idiomas. Podríamos crear un programa como el siguiente:

idioma = input("¿Qué idioma hablas?")  
  
if idioma == "Español":  
 print("Hola"):  
elif idioma == "Inglés":  
 print("Hello"):  
elif idioma == "Francés":  
 print("Salut")  
else:  
 print("No conozco ese idioma")

Podemos tener tantos elif como hagan falta.



### Ejercicio 1.2

Escribe un programa que solicite un texto al usuario. Si el texto es "mañana", debe mostrar el mensaje "Buenos días", si el texto es "tarde" debe mostrar el mensaje "Buenas tardes", y si no debe mostrar el mensaje "Buenas noches"

texto = input("Introduce un texto: ")  
  
if texto == "mañana":  
 print("Buenos días.")  
elif texto == "tarde":  
 print("Buenas tardes.")  
elif texto == "noche":  
 print("Buenas noches.")  
else:  
 print("No entiendo.")

Resultado:

Introduce un texto: tarde  
Buenas tardes.



### Ejercicio 1.3

Crea un programa que solicite al usuario dos valores enteros, los compare y muestre por pantalla si uno es mayor que el otro o si son iguales.

numero1 = input("Introduce un número: ")  
numero2 = input("Introduce otro número: ")  
  
if numero1 > numero2:  
 print(numero1, " es mayor que ", numero2)  
elif numero1 < numero2:  
 print(numero1, " es menor que ", numero2)  
else:  
 print(numero1, " es igual que ", numero2)

Resultado:

Introduce un número: 5  
Introduce otro número: 10  
5 es menor que 10

## Ejercicios propuestos

### Ejercicio 1.0

Crea un programa que solicite al usuario dos valores enteros y muestre por pantalla si el primero es múltiplo del segundo. Para saber si un número es múltiplo del otro, debes hacer la operación módulo (%) entre ellos: si es 0, será múltiplo.

numero1 = input("Introduce un número: ")  
numero2 = input("Introduce otro número: ")  
  
resto = int(numero1) % int(numero2)  
  
if resto == 0:  
 print(numero1, " es múltiplo de ", numero2)  
else:  
 print(numero1, " NO es múltiplo de ", numero2)

Resultado:

Introduce un número: 40  
Introduce otro número: 4  
40 es múltiplo de 4

### Ejercicio 1.1

Crea un programa que solicite al usuario un número entero y haga lo siguiente: primero debe mostrar por pantalla si el número es negativo o positivo. Luego, si el número es positivo lo debe convertir a negativo y si es negativo lo debe convertir a positivo.

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero >= 0:  
 print(numero, " es positivo")  
else:  
 print(numero, " es negativo")  
  
numero = -numero  
  
print("Conversión: ", numero)

Resultado:

Introduce un número: -6  
-6 es negativo  
Conversión: 6

### Ejercicio 1.3

Escribe un programa que solicite un número un mes del año y muestre el número de días que tiene. En caso de introducir un mes desconocido, mostrar un mensaje de Mes desconocido.

mes = input("Introduce un mes del año: ")  
  
if mes == "Enero":  
 print("31 días")  
elif mes == "Febrero":  
 print("28/29 días")  
elif mes == "Marzo":  
 print("31 días")  
elif mes == "Abril":  
 print("30 días")  
elif mes == "Mayo":  
 print("31 días")  
elif mes == "Junio":  
 print("30 días")  
elif mes == "Julio":  
 print("31 días")  
elif mes == "Agosto":  
 print("31 días")  
elif mes == "Septiembre":  
 print("30 días")  
elif mes == "Octubre":  
 print("31 días")  
elif mes == "Noviembre":  
 print("30 días")  
elif mes == "Diciembre":  
 print("31 días")  
else:  
 print("Mes desconocido")

Resultado:

Introduce un mes del año: Junio  
30 días

### Ejercicio 1.3

Crea un programa que solicite al usuario un número entero y muestre por pantalla si ese número es par y positivo. En caso contrario debe indicar si es negativo, impar o ambos.

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero >= 0 and numero % 2 == 0:  
 print(numero, " es par y positivo")  
elif numero < 0 and numero % 2 != 0:  
 print(numero, " es impar y negativo")  
elif numero < 0:  
 print(numero, " es negativo")  
else:  
 print(numero, " es impar")

Resultado:

Introduce un número: -9  
-9 es impar y negativo

### Ejercicio 1.4

Crea un programa que solicite al usuario su peso en kilos y su altura en centímetros y calcule el IMC (peso / altura2); debe mostrar el resultado y luego mostrar un mensaje:

* Si el IMC es menor que 16 se muestra el mensaje: "Necesitas comer".
* Si el IMC está entre (>=)16 y 25(<) se muestra el mensaje: "Estás bien".
* Si el IMC está entre 25 y 30(<) se muestra el mensaje: "Tienes sobrepeso".
* Si el IMC es superior a 30 se muestra el mensaje: "Tienes un problema de obesidad".

peso = input("Introduce tu peso: ")  
altura = input("Introduce tu altura: ")  
peso = int(peso)  
altura = int(altura)  
  
imc = peso / (altura \* altura)  
  
imc = (imc \* 10000)  
print("Tu imc: ", imc)  
  
if imc < 16 :  
 print("Necesitas comer más")  
elif imc >= 16 and imc < 25:  
 print("Estás bien")  
elif imc >= 25 and imc < 30:  
 print("Tienes sobrepeso")  
else:  
 print("Tienes un problema de obesidad")

Resultado:

Introduce tu peso: 70  
Introduce tu altura: 172  
Tu imc: 23.66143861546782  
Estás bien

### Ejercicio 1.5

Crea un programa que solicite al usuario un dorsal de jugador y haga lo siguiente: comprobar que ese número está entre 0 y 99. Si no lo está, entonces el programa debe terminar con un mensaje de error. Si el número está entre 0 y 99 el programa debe mostrar un texto con la posición que corresponde a cada dorsal:

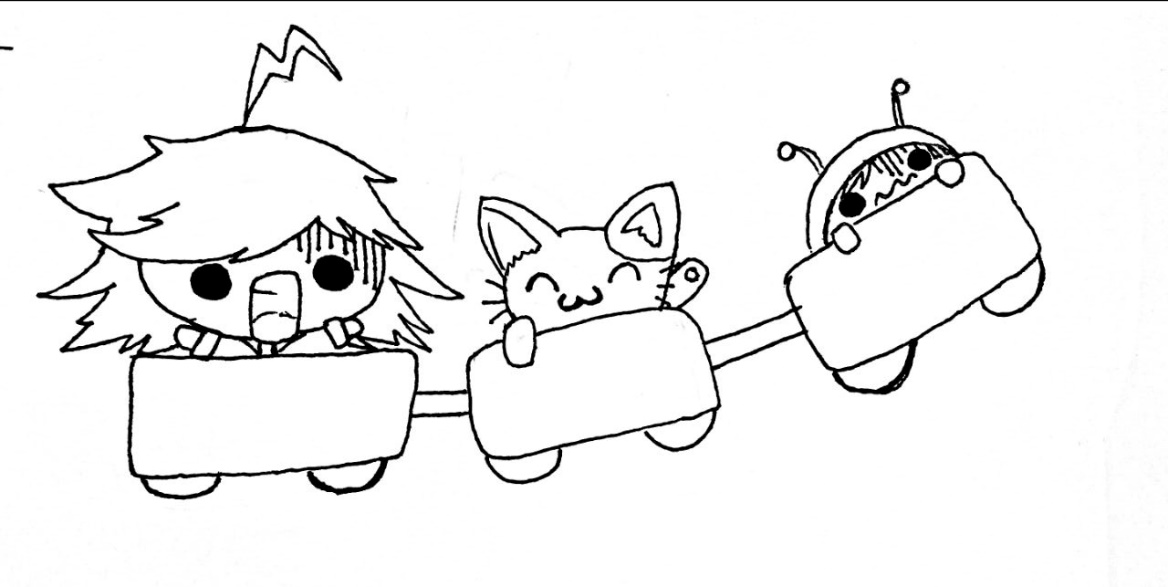
* Si el usuario ha tecleado 1 el texto será "Portero"
* Si el usuario ha tecleado algo entre 1 y 5 el texto será "Defensa"
* Si el usuario ha tecleado algo entre 6 y 8, u 11 el texto será "Centrocampista"
* Si el usuario ha tecleado 9 el texto será "Delantero".
* Para cualquier otra opción el texto será "Cualquiera".

dorsal = input("Introduce dorsal: ")  
dorsal = int(dorsal)  
  
if dorsal >= 0 and dorsal <= 99:  
 if dorsal == 1:  
 print("Portero")  
 elif dorsal >= 1 and dorsal <= 5:  
 print("Defensa")  
 elif dorsal >= 6 and dorsal <= 8 or dorsal == 11:  
 print("Centrocampista")  
 elif dorsal == 9:  
 print("Delantero")  
 else:  
 print("Cualquiera")  
else:  
 print("Error, el dorsal no está entre 0 y 99")

Resultado:

Introduce dorsal: 11  
Centrocampista

# Bucles



Como decíamos al principio, los ordenadores son muy muy tontos. Solo hacen lo que se les diga. Pero por contra, tienen enormes capacidades y una paciencia infinita. No les importará lo más mínimo hacer lo que sea tantas veces como sea necesario.

Una de las tareas más típicas para un ordenador es repetir una instrucción. Esto es algo que se puede conseguir mediante estructuras de bucle. Un bucle es una acción que se repite. Generalmente un bucle tiene una condición para ejecutarse: si esas condiciones se cumplen, entonces se ejecutarán las órdenes que contenga ese bucle. Puedes ver un bucle como una montaña rusa en la que das varias vueltas. A continuación veremos distintos tipos de bucle.

## Bucle while

Un bucle while se ejecuta mientras una condición se cumpla. Su estructura es muy simple:

while \*condición\*:  
 \*instrucciones\*

Por ejemplo, vamos a ejecutar un bucle mientras el valor de una variable sea mayor que 0.

contador = 4  
while contador > 0:  
 print("Estoy dentro del bucle")  
 contador = contador - 1

El resulto en pantalla sería:

Estoy dentro del bucle  
Estoy dentro del bucle  
Estoy dentro del bucle  
Estoy dentro del bucle



Nota:

¿Te has fijado en que dentro del bucle estamos restando un valor a contador? Si no tenemos cuidado y nos olvidamos de hacer eso, el valor de contador nunca cambiaría y crearíamos un bucle infinito. ¡El programa nunca terminaría y se quedaría atascado para siempre!

### Ejercicio 2.0

Crea un programa que defina una variable contador iniciada a 0. Luego escribe un bucle while que mientras contador sea menor que 5 muestre el mensaje Estoy dentro del bucle y que incremente contador en 1.

contador = 0  
while contador < 5:  
 print("Estoy dentro del bucle")  
 contador = contador + 1

Resultado:

Estoy dentro del bucle  
Estoy dentro del bucle  
Estoy dentro del bucle  
Estoy dentro del bucle  
Estoy dentro del bucle

Veamos otro ejemplo. El siguiente programa solicita un dato al usuario en un bucle. El programa no saldrá del bucle mientras el usuario no meta un dato distinto de vacío "":

nombre = ""  
while nombre == "":  
 nombre = input("¿Cómo te llamas?")  
  
print("Hola", nombre)



### Ejercicio 2.1

Escribe un programa que solicite dos números al usuario. El primero debe ser menor que el segundo. El bucle debe mostrar los números que hay en el intervalo entre esos dos números.

min = input("Introduce un mínimo: ")  
min = int(min)  
  
max = input("Introduce un máximo: ")  
max = int(max)  
  
while min < max:  
 print(min)  
 min = min + 1

Resultado:

Introduce un mínimo: 3  
Introduce un máximo: 8  
3  
4  
5  
6  
7

## Bucle For

El bucle for lo utilizamos para repetir una acción un número concreto de veces. Más que una condición, utiliza una especie de contador:

for \*variable\* in \*rango\*:  
 \*instrucciones\*

Por ejemplo, el siguiente bucle mostrará el mensaje hola 4 veces:

for i in range(4):  
 print("Hola")

El resultado sería:

Hola  
Hola  
Hola  
Hola

Algo muy interesante en los bucles for es que la variable i, la cual podría tener el nombre que queramos, tendrá **el valor correspondiente a cada vuelta** del bucle. Para verlo mejor, basta con cambiar un poco el programa anterior:

for i in range(4):  
 print("Hola", i)

Y comprobar el resultado:

Hola 0  
Hola 1  
Hola 2  
Hola 3

Eso nos puede ser útil en muchos programas.



### Ejercicio 2.2

Escribe un programa que solicite un número al usuario y muestre su tabla de multiplicar del 0 al 10.

valor = input("Introduce un número: ")  
valor = int(valor)  
  
for i in range(11):  
 print(valor,"x",i,"=",(valor\*i))  
 # Alternativas:  
 # print("%d x %d = %d" % (valor, i, valor \* i))

Resultado:

Introduce un número: 3  
0 x 3 = 0  
1 x 3 = 3  
2 x 3 = 6  
...



### Ejercicio 2.3

Escribe un programa que pida un número al usuario. Si es igual o menor a 0 debe indicar que meta algo mayor, y si no, debe mostrar el mensaje "Python mola!" por pantalla tantas veces como indique el número:

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero <= 0:  
 print("Debes introducir un número mayor que 0")  
else:  
 for i in range(0, numero):  
 print("Python mola!")

Resultado:

Introduce un número: 3  
Python mola!  
Python mola!  
Python mola!

### Cambiar el rango

Por defecto, range(4) está devolviendo una lista de 0 a 3, es decir: 0, 1, 2, 3. Son un total de cuatro elementos y por tanto el bucle dará cuatro vueltas.

Obviamente, se puede crear cualquier tipo de rango. Si no se indica nada, el rango comienza en 0. Pero se puede indicar un rango entre dos números:

range(0, 4) # 0, 1, 2, 3   
range(2, 6) # 2, 3, 4, 5

Por ejemplo:

for i in range(5, 9):  
 print("Hola", i)

El resultado sería:

Hola 5  
Hola 6  
Hola 7  
Hola 8

También se puede indicar un tercer parámetro para indicar cómo se salta de un valor a otro. Por ejemplo de 2 en 2:

range(1, 11, 2) # 1, 3, 5, 7, 9

En el siguiente ejemplo, el bucle se haría con números pares.

for i in range(0, 6, 2):  
 print("Hola", i)

El resultado sería:

Hola 0  
Hola 2  
Hola 4

### Hacia atrás

También se podría recorrer el bucle hacia atrás, utilizando un salto negativo:

print("Iniciando cuenta atrás: ")  
for i in range(5, 0, -1):  
 print(i)

El resultado sería:

5  
4   
3  
2  
1



### Ejercicio 2.4

Escribe un programa que pida un número al usuario. Luego debe mostrar todos los números pares desde 0 hasta el número que haya metido el usuario. Utiliza un bucle for y salta de dos en dos.

valor = input("Introduce un número: ")  
valor = int(valor)  
  
for i in range(0, valor, 2):  
 print(i)

Resultado:

Introduce un número: 5  
0  
2  
4

### Bucles sobre listas

Los bucles for son especialmente útiles cuando queremos recorrer todos los elementos de una lista, para mostrarlos, procesarlos o lo que sea. La forma es muy sencilla:

objetos = ["estrella", "seta", "flor"]  
for objeto in objetos:  
 print(objetos)

En cada vuelta del bucle la variable objeto tomara un valor de la lista objetos, así que el resultado sería el siguiente:

estrella  
seta  
flor

### Saliendo del bucle

En ocasiones puede que nos interese salir de un bucle y no seguir procesando nada más. Supongamos que tenemos un programa para buscar un nombre dentro de una lista:

nombres = ["Mia", "Jon", "Arya", "Ane", "Bug", "Ada", "Lisp"]  
for nombre in nombres:  
 if nombre == "Ane":  
 print("Ane está en la lista")

El problema de ese programa es que aunque encuentre a Ane, el bucle seguirá hasta el final de la lista. Si esa lista es muy grande, ¡nuestro programa será ineficiente! Como decíamos al principio, los ordenadores son muy tontos. Si no les decimos que paren, seguirán adelante.

Por suerte, en los bucles podemos usar la instrucción break, la cual conseguirá que el bucle termine de golpe:

nombres = ["Mia", "Jon", "Arya", "Ane", "Bug", "Ada", "Lisp"]  
for nombre in nombres:  
 if nombre == "Ane":  
 print("Ane está en la lista")  
 break

### Ejercicio 2.5

Escribe un programa que defina una lista con 3 números. Luego crea un bucle for que recorra la lista y repita la palabra Python tantas veces como indique el número.

repetir = [3, 6, 2]  
  
for veces in repetir:  
 for i in range(veces):  
 print("Python")

Resultado:

Python  
Python  
Python  
Python  
...

## ¿Cuándo usar while o for?

Aunque con los dos podrías hacer lo mismo, realmente cada uno tiene un uso más lógico.

El bucle for se utiliza claramente cuando se quiere ejecutar algo un número concreto de veces, ni más ni menos. O bien, como veremos más adelante, cuando se quieren recorrer estructuras de datos como listas de principio a fin.

El bucle while se puede utilizar cuando las condiciones no son muy concretas. Por ejemplo, si queremos que el usuario introduzca un dato, lo podemos hacer en un bucle. El bucle no terminará hasta que el usuario no introduzca un dato bueno (esa sería la condición).

## Ejercicios propuestos

### Ejercicio 2.0

Escribe un programa con un bucle while que solicite un nombre al usuario por ejemplo "Ada" y muestre un saludo a ese nombre "Hola Ada". Si se introduce el texto "salir" el bucle debe terminar.

nombre = ""  
  
while nombre != "salir":  
 nombre = input("Introduce un nombre: ")  
 print("Hola", nombre)  
  
print("Final.")

Resultado:

Introduce un nombre: Ada  
Hola Ada  
Introduce un nombre: Neko  
Hola Neko  
Introduce un nombre: salir  
Final.

### Ejercicio 2.1

Escribe un programa que contenga un bucle while que solicite al usuario un número y que no termine mientras el número sea diferente de 0. Una vez introducido el número debe mostrarse un saludo tantas veces como indique el número. Si el número es menor que 0 debe terminarse el bucle con un break;

valor = ""  
  
while valor != 0:  
 valor = input("Introduce un numero: ")  
 valor = int(valor)  
 if valor < 0:  
 break  
  
 for i in range(valor):  
 print("Hola")

Resultado:

Introduce un numero: 3  
Hola  
Hola  
Hola  
Introduce un numero: -1

### Ejercicio 2.2

Crea un programa que solicite al usuario un valor entero, comprueba si es mayor que 0 y además par y si es así muestre por pantalla una línea con el carácter \* (asterisco) tantas veces como el valor del número. Usa print("\*"). Por ejemplo si introduce 8 mostrará:

\*\*\*\*\*\*\*\*

Si el valor introducido no cumple los requisitos debes mostrar un mensaje de advertencia al usuario y terminar el programa.

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero <= 0 or numero % 2 != 0:  
 print("Debes introducir un número par mayor que 0")  
else:  
 estrellas = ""  
 while numero > 0:  
 estrellas = estrellas + "\*"  
 numero = numero - 1  
  
 print(estrellas)

Resultado:

Introduce un número: 6  
\*\*\*\*\*\*

### Ejercicio 2.3

Crea un proyecto parecido al anterior, que solo debe admitir pares positivos, pero la línea que debes mostrar debe tener este aspecto:

2:\*-\*

6: \*-\*-\*-\*

Y siempre debe terminar en \*

Por ejemplo, si introducen el 4: \*-\*-\*

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero <= 0 or numero % 2 != 0:  
 print("Debes introducir un número par mayor que 0")  
else:  
 secuencia = ""  
 numero = numero / 2  
 while numero > 0:  
 secuencia = secuencia + "\*-"  
 numero = numero - 1  
  
 secuencia = secuencia + "\*"  
  
 print(secuencia)

Resultado:

Introduce un número: 8  
\*-\*-\*-\*-\*

### Ejercicio 2.4

Crea un programa que solicite al usuario un número entero y usando ese valor debe "dibujar" en la consola un cuadrado formado por \*. Por ejemplo si introduce 4 se mostrará:

\*\*\*\*  
\*\*\*\*  
\*\*\*\*  
\*\*\*\*

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero <= 0:  
 print("Debes introducir un número mayor que 0")  
else:  
 estrellas = "\n"  
 for i in range(numero):  
 for j in range(numero):  
 estrellas = estrellas + "\*"  
   
 estrellas = estrellas + "\n"  
  
 print(estrellas)

Resultado:

Introduce un número: 2  
\*\*  
\*\*

### Ejercicio 2.5

Crea un programa que solicite al usuario un número entero y calcule su factorial. Por ejemplo el factorial de 5 sería 5 x 4 x 3 x 2 x 1 = 120

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero <= 0:  
 print("Debes introducir un número mayor que 0")  
else:  
 factorial = numero  
 while numero > 1:  
 numero = numero - 1  
 factorial = factorial \* numero  
  
 print("Resultado: ", factorial)

Resultado:

Introduce un número: 4  
Resultado: 24

### Ejercicio 2.6

Crea un programa que solicite al usuario un número entero y comprueba si ese número es primo o no, es decir si solamente es divisible por sí mismo o por 1.

numero = input("Introduce un número: ")  
numero = int(numero)  
  
if numero <= 0:  
 print("Debes introducir un número mayor que 0")  
else:  
 divisible = False  
 original = numero  
 numero = numero - 1  
  
 while numero > 1 and not divisible:  
 if original % numero == 0:  
 divisible = True  
  
 numero = numero - 1  
  
  
 if not divisible:  
 print(original, " es primo.")  
 else:  
 print(original, " NO es primo.")

Resultado:

Introduce un número: 5  
5 es primo.

### Ejercicio 2.7

Crea un programa que muestre todas las tablas de multiplicar desde el número 0 al 10.

for i in range(11):  
 for j in range(11):  
 print(i,"x",j,"=",i\*j)  
  
# Lo mismo, de otro modo  
for i in range(11):  
 for j in range(11):  
 print(" %d x %d = %d" % (i, j, i\*j))

# Estructuras de datos

|  |  |  |
| --- | --- | --- |
|  |  |  |

Hasta ahora hemos estado jugando con datos simples, variables que contienen un número, un texto, etc. Pero existen otros tipos que nos permiten crear datos más complejos. No es que sean difíciles, simplemente pueden contener algo más que un simple número.

Los programas de ordenador pueden hacer cosas muy complicadas, pero en esencia, todo lo que hacen es procesar datos. Y muchas veces, esos datos viene en secuencias largas. A continuación vamos a ver algunos de esos tipos de datos.

## Listas

Las listas son un conjunto de datos indexados numéricamente. Esa es la definición muy formal, pero su propio nombre ya te dice lo que son: ¡una lista! En el capítulo sobre tipos de datos ya presentamos las listas y vimos cómo se crean:

idiomas = ["Inglés", "Español", "Francés"]

Recuerda que puedes acceder a los elementos a través de un índice o posición:

idiomas = ["Inglés", "Español", "Francés"]  
print(idiomas[2]) # Francés

La lista se puede representar así:

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 2 |
| "Inglés" | "Español" | "Francés" |



### Ejercicio 3.0

Define una lista de nombres y muéstralas por pantalla:

nombres = ["Ada", "Bug", "Neko"]  
  
print(nombres) # ["Ada", "Bug", "Neko"]

Resultado:

["Ada", "Bug", "Neko"]



### Ejercicio 3.1

Crea un programa que defina una de 5 números con decimales. Luego crea un bucle que calcule la media de todos los números.

numeros = [3.4, 2.7, 4.3, 6.6, 8.3]  
suma = 0.0  
  
for numero in numeros:  
 suma = suma + numero  
  
media = suma / len(numeros)  
  
print("La media es: ", media)

Resultado:

La media es: 5.0600000000000005

## Extraer partes de la lista

Usando el índice numérico, se pueden sacar partes de una lista, creando una sub-lista de la misma. Por ejemplo "quiero los tres primeros valores de la lista" o "quiero desde el 4º al 6º" o "quiero los dos últimos". Para eso basta con indicar un rango de índices:

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

También se pueden sacar los primeros elementos:

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
numeros[:6] # [1, 2, 3, 4, 5, 6]

O los últimos valores

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
numeros[-4:] # [6, 7, 8, 9]

O simplemente el último de todos:

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
numeros[-1] # [9]

## Añadir y eliminar elementos

Si queremos añadir un elemento a una lista, basta con utilizar la función append:

idiomas = ["Inglés", "Español", "Francés"]  
idiomas.append("Italiano")  
print(idiomas) # ["Inglés", "Español", "Francés", "Italiano"]

Y si queremos eliminar un elemento de la lista, se puede usar la orden del:

idiomas = ["Inglés", "Español", "Francés"]  
del idiomas[1]  
print(idiomas) # ["Inglés", "Francés"]

Y también se puede cambiar el valor de un elemento de la lista:

idiomas = ["Inglés", "Español", "Francés"]  
idiomas[2] = "Italiano"

Y recuerda, para recorrer la lista, podemos usar un bucle for:

idiomas = ["Inglés", "Español", "Francés"]  
for idioma in idiomas:  
 print(idioma)

También se puede recorrer la lista utilizando el índice. Para eso tendremos que utilizar la función range, pasándole la longitud de la lista con len:

idiomas = ["Inglés", "Español", "Francés"]  
for i in range(len(idiomas)):  
 print(idiomas[i])

De todas formas, si no se necesita el índice dentro del bucle, es mejor recorrer la lista sin índice tal y como se hace en el ejemplo anterior.

En otros lenguajes, a las listas se les llama "arrays". Ya los deberías conocer, pero te lo volvemos a recordar.

### Ejercicio 3.2

Define una lista de nombres, muéstrala por pantalla. Añade un elemento y muestra la lista por pantalla. Luego elimina un elemento de la lista y muestra la lista por pantalla.

nombres = ["Ada", "Bug", "Neko"]  
  
print(nombres)  
  
nombres.append("Miranda")  
  
print(nombres)  
  
del nombres[1]  
  
print(nombres)

Resultado:

["Ada", "Bug", "Neko"]  
["Ada", "Bug", "Neko", "Miranda"]  
["Ada", "Neko", "Miranda"]

## Diccionarios

Los diccionarios son conjuntos de datos donde cada elemento tiene una clave y un valor. Dichos de otra manera, son como una lista, pero en lugar de tener un índice numérico como 0, 1, 2,... tienen el valor que tú quieras.

Por ejemplo, podemos definir un diccionario que contenga las edades de varias personas, donde el nombre de la persona es la clave y la edad el valor:

edades = { "Ada": 14, "Bug": 10, "Neko": 2 }  
print(edades["Ada"]) # 14

El diccionario se puede representar así:

|  |  |  |
| --- | --- | --- |
| "Ada" | "Bug" | "Neko" |
| 14 | 10 | 2 |



### Ejercicio 3.3

Define un diccionario llamado telefonos donde guardarás los teléfonos de un par de amigos. La clave será el nombre del amigo y el valor el número de teléfono.

telefonos = {"Ada": 666555333, "Bug": 111000111 }  
  
print(telefonos)   
  
for nombre in telefonos.keys():  
 print(nombre, telefonos[nombre])

Resultado:

{'Bug': 111000111, 'Ada': 666555333}  
Bug 111000111  
Ada 666555333

Para añadir nuevos elementos, se puede asignar un nuevo valor:

edades = {"Ada": 14, "Bug": 10, "Neko": 2}  
  
print(edades) # {'Bug': 10, 'Neko': 2, 'Ada': 14}  
  
edades["Miranda"] = 23;  
  
print(edades) # {'Bug': 10, 'Neko': 2, 'Ada': 14, 'Miranda': 23}  
  
del edades["Bug"]  
  
print(edades) # {'Neko': 2, 'Ada': 14, 'Miranda': 23}  
  
for nombre in edades.keys():  
 print(nombre, edades[nombre])

### Ejercicio 3.4

Utiliza el diccionario telefonos del ejercicio anterior. Solicita al usuario datos para meter una nueva entrada en el diccionario: tendrás que pedir un nombre y un teléfono y luego añadirlo al diccionario. Al finalizar, muestra todos los elementos del diccionario.

telefonos = {"Ada": 666555333, "Bug": 111000111 }  
  
nombre = input("Introduce un nombre: ")  
telefono = input("Introduce un número: ")  
  
telefonos[nombre] = int(telefono)  
  
for nombre in telefonos.keys():  
 print(nombre, telefonos[nombre])

Resultado:

Introduce un nombre: Neko  
Introduce un número: 333222000  
Ada 666555333  
Neko 333222000  
Bug 111000111

Podemos eliminar valores del diccionario con la función del:

edades = {"Ada": 14, "Bug": 10, "Neko": 2}  
print(edades) # {'Bug': 10, 'Neko': 2, 'Ada': 14}  
del edades["Bug"]  
print(edades) # {'Neko': 2, 'Ada': 14}

¿Y si queremos recorrer todos los valores del diccionario? No hay problema, pero tendremos que utilizar una función que nos devuelva todas las claves del diccionario: keys(). Sería así:

edades = {"Ada": 14, "Bug": 10, "Neko": 2}  
for nombre in edades.keys():  
 print(nombre, edades[nombre])

Por pantalla ser vería:

Ada 14  
Bug 10  
Neko 2

### Ejercicio 3.5

Utiliza el diccionario telefonos del ejercicio anterior. Solicita al usuario un nombre, y luego elimina ese elemento del diccionario. Al finalizar, muestra todos los elementos del diccionario.

telefonos = {"Ada": 666555333, "Bug": 111000111 }  
  
nombre = input("Introduce un nombre: ")  
  
del telefonos[nombre]  
  
for nombre in telefonos.keys():  
 print(nombre, telefonos[nombre])

Resultado:

Introduce un nombre: Bug  
Ada 666555333



Nota:

En otros lenguajes, a los diccionarios se les llama hash o hashtables.

## Estructuras de datos combinadas

¡Las estructuras básicas como listas y diccionarios pueden contener valores que también sean listas y diccionarios!

Se pueden crear estructuras de datos anidadas, tan complejas como necesites. Por ejemplo, supongamos que quieres representar los datos de un amigo con el siguiente diccionario:

amigo = {"nombre": "Neko", "edad": 2}

¿Y si quieres tener una estructura de datos que contenga varios amigos? En ese caso, puedes hacer una lista de diccionarios:

amigos = [  
 {"nombre": "Neko", "edad": 2},  
 {"nombre": "Bug", "edad": 10},  
 {"nombre": "Ada", "edad": 14}  
]  
print(amigos[1]) # {"nombre": "Bug", "edad": 10}  
print(amigos[2]["nombre"]) # Ada

### Ejercicio 3.6

Escribe un programa que defina una lista de diccionarios llamada cliente que contenga las claves: nombre, email y edad. El programa debe recorrer la lista y mostrar el nombre y edad de cada cliente.

clientes = [  
 {  
 "nombre": "Juan",  
 "email": "jj@terra.com",  
 "edad": 39  
 },  
 {  
 "nombre": "Pedro",  
 "email": "pp@ozu.es",  
 "edad": 42  
 },  
 {  
 "nombre": "Ana",  
 "email": "ana@ole.com",  
 "edad": 37  
 }  
]  
  
for cliente in clientes:  
 print(f"{cliente['nombre']} {cliente['edad']}")

Resultado:

Juan 39  
Pedro 42  
Ana 37

¿Y si quieres una estructura de datos que contenga el estado de un juego, con sus personajes y sus objetos? Podría hacer un diccionario anidado, donde la clave es el nombre del personaje:

pantalla = {  
 "Mario": { "vida": 10, "objetos": ["seta", "estrella"]},  
 "Luigi": { "vida": 7, "objetos": []},  
 "Toad": { "vida": 15, "objetos": ["seta"]},  
}  
  
print(pantalla["Luigi"]["vida"]) # 7

Pero ¿Cómo sé que tipo de estructura debo diseñar? Depende de cómo la vayas a usar. A veces necesitarás recorrer todos, otras veces necesitarás acceder a un elemento concreto,... según lo que requiera tu programa tendrás que diseñar una estructura concreta.

# Textos

El tipo de dato texto, también llamado cadena o *string*, es fundamental en los programas. Por eso dispone de muchas utilidades para facilitarnos el manejo de este tipo de datos. A continuación, veremos algunas funciones útiles para los textos, pero antes, conviene revelar algo sobre el texto:

## Los textos son listas

Efectivamente, para el ordenador, un texto es como una lista. Una lista o cadena de letras, y la podemos tratar como tal:

texto = "Hola soy Ada"  
texto[1] # "o"

Incluso podemos recorrer el texto como si fuera una lista:

texto = "Ada"  
  
for caracter in texto:  
 print(caracter)

La salida sería:

A  
d  
a

También podemos saber la longitud de un texto con la función len():

texto = "Neko maulla"  
len(texto) # 11

Pero si duda, lo más interesante es que podemos extraer la parte que queramos del texto indicando el inicio y el final:

texto = "Python mola"  
texto[0:6] # "Python"  
texto[7:12] # "mola"

También se pueden sacar los primeros caracteres

texto = "Python mola"  
texto[:6] # "Python"

O los últimos caracteres

texto = "Python mola"  
texto[-4:] # "mola"

O simplemente el último de todos:

texto = "Python mola"  
texto[-1] # "a"

## Mayúsculas/Minúsculas

Tenemos una serie de funciones para convertir texto a mayúsculas o minúsculas:

texto = "Profesora Ada"  
texto.upper() # PROFESORA ADA  
texto.lower() # Profesora Ada

También tenemos una función llamada title(), la cual cambia cada palabra dentro de un texto, poniendo la primera letra en mayúsculas.

texto = "esto es una frase"  
texto.title() # Esto Es Una Frase

## split: de texto a lista

split es una interesante función que parte un texto en cachos y lo convierte en una lista:

texto = "ven conmigo si quieres vivir"  
palabras = texto.split() # ["ven", "conmigo", "si", "quieres", "vivir"]

Por defecto el corte del split se hace en los espacios del texto. Pero se puede indicar cualquier otro separador, por ejemplo el punto y coma ;:

texto = "Ada;Neko;Bug"  
nombres = texto.split(";") # ["Ada", "Neko", "Bug" ]

## Búsquedas

En ocasiones nos interesará buscar una palabra dentro de un texto. Para eso podemos usar la función find. En caso de encontrar la palabra, muestra la posición en la que se empieza esa palabra. Si no lo encuentra, devuelve -1.

palabras = "La mejor profesora es Ada, sin duda"  
encontrado = palabras.find("mejor") # 3  
encontrado = palabras.find("Ada") # 22  
noEncontrado = palabras.find("xxx") # -1

Si queremos saber si un texto empieza de alguna manera, se puede usar startswith()

palabras = "Python es un buen lenguaje"  
empieza = palabras.startswith("Py") # True  
empieza = palabras.startswith("es") # False

Mientras que si queremos saber si un texto acaba de una manera, se puede usar endswith():

palabras = "Python es un buen lenguaje"  
acaba = palabras.endswith("aje") # True  
acaba = palabras.startswith("ajes") # False

## Eliminar sobrantes

Los textos pueden empezar o terminar con espacios en blanco u otros caracteres que quizá nos interese eliminar, como los saltos de línea. Para quitar esas partes sobrantes de un texto, se pueden usar las siguientes funciones.

Con lstrip() se eliminan los espacios al inicio del texto:

texto = " Tengo espacios "  
limpio = texto.lstrip() # "Tengo espacios "

Con rstrip() se eliminan los espacios al inicio del texto:

texto = " Tengo espacios "  
limpio = texto.rstrip() # " Tengo espacios"

Y con strip() quitamos los espacios de ambos lados:

texto = " Tengo espacios "  
limpio = texto.strip() # "Tengo espacios"

Por defecto se quitan espacios, pero podemos indicar cualquier texto que queramos quitar:

texto = "--Texto con guión"  
limpio = texto.lstrip("-") # "Texto con guión"

También los saltos de línea cuando leemos texto desde un fichero o desde la consola:

texto = "Esto tiene un salto de línea\n"  
limpio = texto.rstrip("\n")

## Ejercicios propuestos

### Ejercicio 3.0

Escribe un programa que inicie una lista de 5 números (iniciados a 0), otro de 5 nombres iniciados a mano y otro de valores 5 booleanos (iniciados a false)

nombres = ["Frodo", "Sam", "Merrin", "Pippin"]  
booleanos = [True]\*5  
numeros = [0]\*5  
  
print(nombres)  
print(numeros)  
print(booleanos)

Resultado:

["Frodo", "Sam", "Merrin", "Pippin"]  
[0, 0, 0, 0, 0]  
[True, True, True, True, True]

### Ejercicio 3.1

Escribe un programa que defina una lista de 10 números. Luego debe crear un bucle que en las **posiciones** pares meta un 0.

numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
for i in range(len(numeros)):  
 if i % 2 == 0:  
 numeros[i] = 0  
  
print(numeros)

Resultado:

[0, 2, 0, 4, 0, 6, 0, 8, 0, 10]

### Ejercicio 3.2

Escribe un programa para gestionar una lista, que muestre al usuario un menú con 4 opciones: (1) meter elemento, (2) eliminar, (3) mostrar y (4) salir. El menú debe mostrarse mientras el usuario no meta la opción 4. Si elige la opción 1, se hace push de un valor cualquiera, si elige 2 se hace pop, si elige 3 se muestra el contenido dla lista.

numeros = []  
opcion = -1  
  
while opcion != "4":  
 print("Elige opción")  
 print("1. Meter elemento.")  
 print("2. Sacar elemento.")  
 print("3. Mostrar lista.")  
 print("4. Salir.")  
 opcion = input("Elige opción: ")  
  
 if "1":  
 numeros.append(0)  
 elif "2":  
 numeros.pop()  
 elif "3":  
 print(numeros)  
 elif "4":  
 print("Hasta otra")  
 else:  
 print("Opción desconocida")

Resultado:

Elige opción  
1. Meter elemento.  
2. Sacar elemento.  
3. Mostrar lista.  
4. Salir.  
Elige opción: 3  
[]

### Ejercicio 3.3

Escribe un programa que solicite palabras al usuario y las vaya concatenando para construir una frase, hasta que el usuario escriba un punto (.). Entonces el programa deberá mostrar la frase creada. Si el usuario no escribe nada, no se debe concatenar nada.

frase = ""  
palabra = ""  
  
while palabra != ".":  
 palabra = input("Escribe una palabra: ")  
  
 if palabra != "." or palabra != "":  
 frase = frase + " " + palabra  
  
print("Frase creada:", frase)

Resultado:

Escribe una palabra: Hola  
Escribe una palabra: qué  
Escribe una palabra: tal  
Escribe una palabra: .  
Hola qué tal

### Ejercicio 3.4

Escribe un programa que solicite al usuario su nombre, su lugar de nacimiento y su año de nacimiento. Luego debe mostrar una frase con toda esa información utilizando la interpolación o plantillas de cadenas.

nombre = input("Escribe tu nombre: ")  
lugar = input("Escribe tu lugar de nacimiento: ")  
fecha = input("Escribe tu año de nacimiento: ")  
  
mensaje = f"Te llamas {nombre} naciste en {lugar} en {fecha}"  
  
print(mensaje)

Resultado:

Escribe tu nombre: Ada  
Escribe tu lugar de nacimiento: Teverga  
Escribe tu año de nacimiento: 2006  
Te llamas Ada naciste en Teverga en 2006

### Ejercicio 3.5

Escribe un programa que solicite al usuario una frase. Luego debe solicitar una palabra de esa frase, y como resultado, el programa devolverá la misma frase con esa palabra en mayúsculas:

frase = input("Escribe una frase: ")  
palabra = input("Escribe una palabra de esa frase: ")  
  
posicion = frase.index(palabra)  
  
if posicion != -1:  
 inicio = frase[0:posicion]  
 final = frase[posicion + len(palabra):len(frase)]  
 resultado = f"{inicio}{palabra.upper()}{final}"  
  
 print(resultado)  
else:  
 print(palabra, "no está en la frase.")

Resultado:

Escribe una frase: Qué buena es la profa Ada  
Escribe una palabra de esa frase: buena  
Qué BUENA es la profa Ada

### Ejercicio 3.6

Crea un programa que defina una lista de 5 nombres y luego utilice un bucle para mostrar los nombres de uno en uno.

nombres = ["Frodo", "Merrin", "Sam", "Pip", "Bilbo"]  
  
for nombre in nombres:  
 print(nombre)  
  
# variante:  
for i in range(len(nombres)):  
 print(nombres[i])

Resultado:

Frodo  
Merrin  
Sam  
Pip  
Bilbo

### Ejercicio 3.7

Crea un programa que defina una lista de 10 números enteros. Luego crea otro bucle que calcule que incremente en uno cada uno de los elementos y los muestre.

numeros = [3, 5, -4, 2, 1, 4, 0, 6, 9, 8, 3]  
  
for i in range(len(numeros)):  
 numeros[i] = numeros[i] + 1  
  
print(numeros)  
  
  
# Alternativa para la suma:  
# numerosIncrementados = numeros.map( numero => numero + 1 )

Resultado:

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

### Ejercicio 3.8

Crea un programa que defina una lista de 10 números enteros Luego crea un bucle que determine si en la lista hay algún elemento repetido. Con que encuentre uno repetido es suficiente.

numeros = [3, 5, -4, 2, 1, 4, 0, 6, 9, 8, 3]  
repetido = False  
i = 0  
j = 0  
  
while i < len(numeros) and not repetido:  
 while j < len(numeros):  
 if numeros[i] == numeros[j]:  
 repetido = True  
 break  
 j = j + 1  
 i = i + 1  
  
if repetido:  
 print("Hay un número repetido")  
else:  
 print("No hay un número repetido")

Resultado:

Hay un número repetido

### Ejercicio 3.9

Crea un programa que defina una lista iniciado con 10 números enteros. Luego crea otro bucle que contabilice el total de números positivos, negativos y los que sean 0.

numeros = [3, 5, -4, 2, 1, 4, 0, 6, -9, 8, 3]  
  
positivos = 0  
negativos = 0  
ceros = 0  
  
for numero in numeros:  
 if numero > 0:  
 positivos = positivos + 1  
 elif numero < 0:  
 negativos = negativos + 1  
 else:  
 ceros = ceros + 1  
  
print("Positivos: ", positivos)  
print("Negativos: ", negativos)  
print("Ceros: ", ceros)

Resultado:

Positivos: 8  
Negativos: 2  
Ceros: 1

### Ejercicio 3.10

Crea un programa que defina una lista de dos dimensiones de 5x10 elementos. Crea un bucle que inicie los valores de la lista usando números aleatorios. Para crear números aleatorios importa la librería random y utiliza la función random.randint(), tal y como se muestra aquí:

import random  
random.randint(0, 30); # número aleatorio entre 0 y 30

Después de eso crea otro bucle que si encuentra el número 15 en algún elemento interrumpa el bucle y muestre la posición en la que está.

import random  
  
matriz = [([0] \* 10)] \* 5  
  
print(matriz)  
  
for i in range(len(matriz)):  
 random.seed()  
 for j in range(len(matriz[i])):  
 matriz[i][j] = random.randint(0, 30)  
  
print(matriz)  
  
for i in range(len(matriz)):  
 for j in range(len(matriz[i])):  
 if matriz[i][j] == 15:  
 print("Encontrado 15 en ", i, )

### Ejercicio 3.11

Crea un programa que defina una lista iniciado con 10 números enteros. En un bucle muestra por pantalla todos los elementos. Luego crea otro bucle que baraje los elementos usando el método random del ejercicio anterior en los índices. Luego muestra el resultado.

import random  
  
numeros = [4, 7, -3, 7, 1, 11, 9, 0, 5, 8]  
  
print(numeros)  
  
for i in range(len(numeros)):  
 indiceAleatorio = random.randint(0, len(numeros) - 1)  
 anterior = numeros[i]  
 numeros[i] = numeros[indiceAleatorio]  
 numeros[indiceAleatorio] = anterior  
  
print(numeros)

Resultado:

[5, 4, 11, 7, 1, -3, 0, 9, 7, 8]

# Funciones

|  |  |
| --- | --- |
| miau() | dormir() |
|  |  |

Las funciones son pequeños programas dentro de los programas. Esta sería una función que simplemente saca un saludo por pantalla:

def saludo ():  
 print "Hola"

Como se puede ver, una función se define utilizando la palabra def seguida del nombre de la función, en este caso saludo y la lista de parámetros (), la cual está vacía en este caso. En el cuerpo de la función, podemos poner las instrucciones que queramos.

Y siguiendo con el estilo de Python, observa que el código que va dentro de la función va precedido de una tabulación o espacios.

Una vez definida esa función, cada vez que la utilicemos se ejecutará el código que hay en ella:

saludo()

Lo cual mostrará:

Hola

### Ejercicio 4.0

Escribe un programa con tres funciones llamadas días, tardes y noches. Cada una debe mostrar un saludo distinto, "Buenos días", "Buenas tardes" y "Buenas noches" respectivamente. Añade las llamadas de a las tres funciones.

def dias ():  
 print("Buenos días")  
  
def tardes ():  
 print("Buenos tardes")  
  
def noches ():  
 print("Buenos noches")  
  
dias()  
tardes()  
noches()

Resultado:

Buenos días  
Buenos tardes  
Buenos noches

## Parámetros

Las funciones pueden recibir parámetros. Estos se convierten en variables dentro de la función y nos permiten que la función haga cosas diferentes según lo que les pasemos.

Por ejemplo, podemos crear una función que salude a alguien:

def saluda(persona):  
 print("Hola", persona)

Dentro de la función, el valor de persona será distinto según lo que se le pase en la llamada Se podría usar así:

saluda("Neko") # persona será "Neko"  
saluda("Ada") # persona será "Ada"

Lo que resultaría en:

Hola Neko  
Hola Ada



### Ejercicio 4.1

Escribe un programa con una función llamada cuadrado que tenga un parámetro a. La función debe multiplicar a \* a y mostrar el resultado por pantalla.

def cuadrado (a):  
 print(a \* a)  
  
  
cuadrado(3)

Resultado:

9

### Parámetro con valor por defecto

Los parámetros de una función pueden tener un valor por defecto. Es decir, se les puede asignar un valor concreto y si no se pasa ese parámetro en la llamada, el parámetro tomará ese valor por defecto.

def saluda(persona, veces = 3):  
 for i in range(veces):  
 print("Hola", persona)

Si se le llama con:

saluda("Neko", 2)

Se vería:

Hola Neko  
Hola Neko

En cambio, si no pasamos el segundo parámetro, vecestomará el valor por defecto 3:

saluda("Bug")

Sería:

Hola Bug  
Hola Bug  
Hola Bug

### Parámetros infinitos

Las funciones en Python permiten que les pases un número indeterminado de parámetros. Suena un poco raro, pero resulta útil. Imagina que quieres crear una función que sume los **todos** parámetros que le pases.

def sumar(\*valores):  
 resultado = 0  
  
 for valor in valores:  
 resultado = resultado + valor  
   
 return resultado

Si te fijas, hemos definido el parámetro valores con un \* por delante. Con eso le estamos indicando que no se trata de un único parámetro, sino una lista que puede tener cualquier longitud.

Por tanto, a esa función la podemos llamar así:

sumar(3, 4, 5) # 12  
sumar(2, 45) # 47



### Ejercicio 4.2

Escribe un programa con una función llamada unir que reciba parámetros indefinidos o dinámicos. La función unir debe tomar los parámetros y retornar un texto con todos ellos formando una frase.

def unir(\*palabras):  
 frase = ""  
  
 for palabra in palabras:  
 frase = frase + " " + palabra  
   
 return frase  
  
  
print(unir("Hola", "qué", "tal"))

Resultado:

Hola qué tal

## Retorno

Las funciones pueden hacer todas las operaciones que sean necesarias, pero son más útiles cuando devuelven un resultado.

Para eso se utiliza la instrucción return, la cual solo puede devolver un dato concreto o una estructura de datos.

Por ejemplo, una función que calcule la suma de dos valores:

def suma(a, b):  
 resultado = a + b  
 return resultado

Se puede abreviar un poco haciendo directamente:

def suma(a, b):  
 return a + b

Se podría usar la función así:

print(suma(40, 2)) # 42

Debes tener en cuenta un par de cosas con el return:

1. Una vez se hace return, el programa sale de la función.
2. Puedes tener más de un return en la función, pero solo se ejecutará uno de ellos.



### Ejercicio 4.3

Escribe un programa con una función llamada dividir que reciba los parámetros a y b. La función debe retornar la división entre los dos números.

def dividir (a, b):  
 return a / b  
  
  
print(dividir(32, 4))

Resultado:

8

Otro ejemplo, una función que multiplique un valor varias veces. Si el número de veces es menor que 1 devolverá un 0:

def multiplica(numero, veces):  
 if (veces > 0):   
 total = 1  
 for i in range(veces):  
 total = total \* numero  
 else:  
 return 0  
  
multiplica(2, 3) # 8

Nota:

La función anterior queda más clara así.

def multiplica(numero, veces):  
 if (veces < 1): return 0   
  
 total = 1  
 for i in range(veces):  
 total = total \* numero

## Llamadas anidadas

No tengas miedo a anidar las llamadas de funciones. Es algo muy natural en programación. Imagina que tenemos esta función que se encarga de solicitar un número al usuario y devuelve el ńumero introducido:

def leeNumero():  
 numero = input("Escribe número: ")  
 return int(numero)

Supongamos que también tenemos otra función para restar números:

def restar (a, b)  
 return a - b

Si queremos hacer un programa que solicite dos números al usuario y los reste podríamos hacerlo así:

print(restar(leeNumero(), leeNumero()))

En lugar de poner un número o una variable como parámetro, podemos poner una llamada a una función.

Esas funciones leeNumero(), devolverán algún valor. En cuanto se llame al primer leeNumero() será como tener:

print(restar(5, leeNumero()))

Y luego tras llamar al segundo leeNumero() :

print(restar(5, 2)

Luego se llamará a restar y devolverá un valor:

print(3)

Y finalmente se llamará a print y veremos el resultado:

3



### Ejercicio 4.4

Escribe dos funciones. Una que reciba un número y devuelva el mismo número en positivo.

def positivo(valor)

Y otra función que reciba dos números y calcule la potencia entre ellos.

def potencia(valor1, valor2)

Debes llamar a la segunda función pasando como parámetros:

potencia(positivo(-5), positivo(4))

Nota:

No utilices funciones preexistentes.

def positivo (valor):  
 if valor < 0:  
 return -valor  
  
 return valor  
  
  
def potencia (valor1, valor2):  
 return valor1 \*\* valor2  
  
print(positivo(-1))  
print(potencia(2, 3))  
print(potencia(positivo(2), positivo(4)))  
potencia(positivo(-5), positivo(4))

Resultado:

1  
8  
16

## ¿Por qué usar funciones?

Ok, ¿para qué necesitamos crear funciones? Pues lo cierto es... que son fundamentales.

Hasta ahora hemos visto programas que son una simple secuencia de órdenes, las cuales se ejecutan desde el principio hasta el final.

Eso está bien cuando los programas son simples y tienen que hacer pocas cosas, pero cuando el programa tiene que hacer algo más complejo es muy probable que tengas que usar funciones, por varios motivos.

### Motivo 0: divide y vencerás

Los programas tratan de resolver problemas ofreciendo una solución. En ocasiones los problemas pueden ser muy complejos de afrontar. Las funciones te permiten abordar esos problemas por partes. Cada función te puede dar la solución para una parte del problema. Por lo tanto, puedes dividir el problema en muchos pequeños problemas y solucionar cada problema con una función. Escribir el código en funciones es el primer paso que te permitirá diseñar programas más complejos.

### Motivo 1: no repetir el código

Si tu programa tiene que hacer alguna cosa varias veces, tendrías que repetir el código tantas veces como fuera necesario. Imagina que tienes que recibir varios datos del usuario, y que cada vez que lo haces tienes que comprobar que el dato no está vacío:

dato = ""  
  
while dato == "":  
 dato = input("Por favor, introduce un dato: ")  
 if dato == "":  
 print("¡El dato está vacío!)

Si solicitas al usuario 3 datos, ¡tendrías que repetir ese código 3 veces! En cambio si creas una función con ese mismo código, solo lo tendrás que escribir una vez y luego podrás usarlo tantas veces como necesites.

Nota:

NO repetir el código es una de las reglas más importantes que debe seguir todo buen programador. Incluso puedes aplicar la **regla del 3**: a la tercera vez que tengas que repetir algo, tienes que automatizarlo.

### Motivo 2: reutilizar código

Además de no repetir el código, una función nos permite que un mismo código sirva para distintos tipos de datos. ¡Para eso se utilizan los parámetros!

def saludar(saludo, veces):  
 for i in range(veces):  
 print(saludo)

Se le puede llamar con distintos valores:

saludar("Holi", 3)  
saludar("Aupa", 1)

Sería:

Holi  
Holi  
Holi  
Aupa

### Motivo 3: facilitar el mantenimiento

Si el código está en un único sitio, es más fácil corregirlo, cambiarlo, mejorarlo y mantenerlo en general. Crear un programa desde cero es muy bonito, pero el verdadero trabajo es mantener el código a lo largo del tiempo. Si tenemos nuestras funciones bien definidas, nos ahorraremos *muuuuucho* trabajo.

### Motivo 4: permite hacer tests

Quizá seas un poco joven para esto, pero las verdaderas *pros* como yo testeamos nuestros programas. ¿Qué significa eso? Que escribimos programas cuya única función es comprobar que nuestros programas hacen lo que deben. Si tu código tiene funciones, podrás escribir tests para comprobar que esas funciones hacen lo que deben.

En realidad, cuando ya eres una experta, lo suyo es que escribas el test ¡antes que la función en si misma! Veremos un poco más sobre los test más adelante.

## Cómo hacer buenas funciones

Cualquiera puede escribir funciones y agrupar el código en pequeñas partes. Pero si quieres escribir funciones como un *pro*, tienes que procurar lo siguiente:

* Una función **debe hacer solo una cosa**. Es mejor tener muchas pequeñas funciones que pocas funciones haciendo muchas cosas. Si tu función no cabe en la pantalla o pasa de 24 líneas, quizá debas dividirla en pequeñas partes.
* Una función **no debería cambiar nada que haya fuera**. Si no quieres tener sorpresas, una función no debería liarla dentro del programa.
* Una función **debería retornar algo**, y ese algo siempre debería ser lo mismo para determinados parámetros.

## Ejercicios propuestos

### Ejercicio 4.0

Escribe una función que reciba dos números, detecte cuál es más grande y muestre la diferencia entre ellos.

def diferencia (valor1, valor2):  
 diferencia = 0  
  
 if valor1 > valor2:  
 diferencia = valor1 - valor2  
 else:  
 diferencia = valor2 - valor1  
  
 print("La diferencia es: ", diferencia)  
  
diferencia(10, 5)  
diferencia(4, 12)

Resultado:

La diferencia es: 5  
La diferencia es: 8

### Ejercicio 4.1

Escribe un programa que reciba dos números y un signo de operación aritmética: +, -, \*, /. La función debe retornar el resultado de esa operación entre los dos números.

def calcular (valor1, valor2, op):  
 if op == "+": return valor1 + valor2  
 elif op == "-": return valor1 - valor2  
 elif op == "\*": return valor1 \* valor2  
 elif op == "/": return valor1 / valor2  
 else: return "Operación desconocida"  
  
resultado = calcular(4, 6, "\*")  
print(resultado)  
  
resultado = calcular(5, 5, "-")  
print(resultado)  
  
resultado = calcular(4, 3, "@")  
print(resultado)

Resultado:

24  
0  
Operación desconocida

### Ejercicio 4.2

def saludo(momentoDelDia) Esta función recibe como parámetro un momento del día: "mañana", "tarde" o "noche" y debe devolver el correspondiente saludo: "Buenos días", "Buenas tardes", y "Buenas noches" respectivamente.

def saludo (momento):  
 if momento == "mañana": return "Buenos días"  
 elif momento == "tarde": return "Buenas tardes"  
 elif momento == "noche": return "Buenas noches"  
 else: return ""  
  
print(saludo("tarde"))  
  
def saludo2 (momento):  
 return {  
 "mañana": "Buenos Días",  
 "tarde": "Buenas tardes",  
 "noche":"Buenas noches"  
 }[momento]  
  
print(saludo2("tarde"))

Resultado:

Buenas tardes  
Buenas tardes

### Ejercicio 4.3

def iniciarConNumero (numeros, numero)

Esta función debe iniciar todos los elementos del array números con el número que pasamos como parámetro.

def iniciarConNumero (longitud, numero):  
 numeros = []  
 for i in range(longitud):  
 numeros.append(numero)  
 return numeros  
  
def iniciarConNumero1 (longitud, numero): return [numero] \* longitud  
  
print(iniciarConNumero(10, 3))  
print(iniciarConNumero1(10, 3))

Resultado:

[3, 3, 3, 3, 3, 3, 3, 3, 3, 3]  
[3, 3, 3, 3, 3, 3, 3, 3, 3, 3]

### Ejercicio 4.4

def aleatorio (max) Esta función debe devolver un número aleatorio entre 0 y el valor máximo que se pasa como parámetro.

import random  
  
def aleatorio (max):  
 return random.randint(0, max)  
  
print(aleatorio(5))

Resultado:

3

### Ejercicio 4.5

def generarNombre (silabas)

Una función que dado un número de sílabas genere un nombre (alternar consonantes y vocales) aleatorio. Puedes utilizar la función del ejercicio anterior.

import random  
  
def aleatorio (max):  
 return random.randint(0, max - 1)  
  
  
def generarNombre (silabas):  
 vocales = ["a", "e", "i", "o", "u"]  
 consonantes = ["b","c","d","f","g","h","j","k","l","m","n","ñ","p","q","r","s","t","v","w","x","y","z"]  
 nombre = ""  
  
 for i in range(silabas):  
 consonante = consonantes[aleatorio(len(consonantes))]  
 vocal = vocales[aleatorio(len(vocales))]  
 nombre = nombre + consonante + vocal  
  
 return nombre  
  
  
print(generarNombre(3))

Resultado:

xamozu

### Ejercicio 4.6

def generarPassword(length)

Una función que dada una longitud genere una cadena o *string* con caracteres aleatorios. Puedes usar un array de *strings* con caracteres e ir sacando caracteres aleatorios del array para generar un nombre. Para generar números aleatorios:

import random  
  
def aleatorio (max):  
 return random.randint(0, max - 1)  
  
def generarPassword (longitud):  
 caracteres = ["a","b","c","d","e","f","g","h","i","j","k","l",  
 "m","n","ñ","o","p","q","r","s","t","u","v","w","x","y","z",  
 "0","1","2","3","4","5","6","7","8","9",".","-","\_","!","$"]  
 password = ""  
  
 for i in range(longitud):  
 caracter = caracteres[aleatorio(len(caracteres))]  
 password = password + caracter  
  
 return password  
  
  
print(generarPassword(8))

Resultado:

\_!5\_flg$

### Ejercicio 4.7

Crea una función llamada factura(productos, cantidades, precios) que recibe tres arrays del mismo tamaño con los siguientes contenidos:

1. productos: nombres de productos.

2. cantidades: números enteros indicando cantidad.

3. precios: números con decimales indicando el precio de cada producto.

La función debe recorrer cada producto y calcular el precio total según su cantidad y precio. Se debe mostrar ese precio total de cada producto y al final del programa, se debe mostrar el precio total.

def factura(productos, cantidades, precios):  
 factura = "FACTURA\n-------------------\n"  
 total = 0  
  
 for i in range(len(productos)):  
 factura = factura + productos[i]  
 factura = factura + " x " + str(cantidades[i])  
 factura = factura + " : " + str(precios[i]) + "\n"  
  
 total = total + (cantidades[i] \* precios[i])  
  
 factura = factura + "\n-----------------------\n"  
 factura = factura + "Total: " + str(total)  
  
 return factura  
  
  
# Ejemplo de llamada  
totalFactura = factura(["Pan","Huevos","Harina"],[1,6,2],[1.2, 0.2, 0.8])  
print(totalFactura)

Resultado:

FACTURA  
-------------------  
Pan x 1 : 1.2  
Huevos x 6 : 0.2  
Harina x 2 : 0.8  
  
-----------------------  
Total: 4.0

### Ejercicio 4.8

Este va a ser más complicado. Crea un programa con una función para generar personajes de juegos.

def generarAtributos (nivelCompensacion)

Esta función debe definir tres variables: fuerza, velocidad e inteligencia. El programa lo que debe hacer es repartir 20 puntos entre las tres variables. O dicho de otra forma, entre las tres variables deben sumar 20. El parámetro nivelCompensacion debe servir para indicar si se reparten puntos muy diferenciados o igualados, cuanto más alto el valor más descompensado, es decir, la diferencia entre atributos es mayor; cómo hacerlo es cosa del programador.

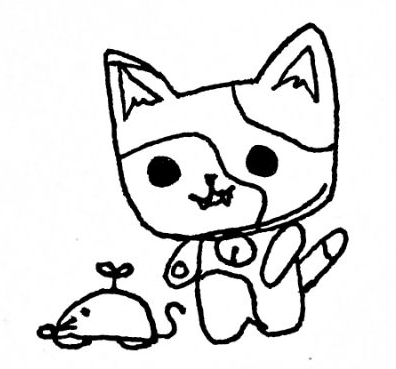
Al final el programa debe mostrar un resumen de los puntos asignados.

import random  
  
def aleatorio (max):  
 return random.randint(0, max)  
  
def generarAtributos (nivelCompensacion):  
 darPuntosA = 0  
 inteligencia = 0  
 fuerza = 0  
 velocidad = 0  
  
 puntosRestantes = 20  
 puntos = 0  
  
 while puntosRestantes > 0:  
 if nivelCompensacion > puntosRestantes:  
 puntos = puntosRestantes  
 puntosRestantes = 0  
 else:  
 puntos = aleatorio(nivelCompensacion+1)  
 puntosRestantes = puntosRestantes - puntos  
  
 darPuntosA = aleatorio(3)  
  
 if darPuntosA == 0:  
 inteligencia = inteligencia + puntos  
 elif darPuntosA == 1:  
 fuerza = fuerza + puntos  
 elif darPuntosA == 2:  
 velocidad = velocidad + puntos  
  
 print("\nValores asignados por compensación: ", nivelCompensacion)  
 print("Inteligencia: ", inteligencia)  
 print("Fuerza: ", fuerza)  
 print("Velocidad: ", velocidad)  
  
  
generarAtributos(3)

Resultado:

Valores asignados por compensación: 3  
Inteligencia: 3  
Fuerza: 1  
Velocidad: 10

# Clases



Las clases nos permiten aplicar una técnica llamada programación orientada a objetos. Es otra estrategia para resolver problemas complejos. Con las funciones, dividimos un problema en pequeños problemas. En cambio, con la programación orientada a objetos, lo que tratamos de hacer es dividir el problema en clases. ¿Pero cómo? representando todo aquello que forma parte del problema utilizando clases.

Imagina que tuviéramos que hacer el programa de un juego de carreras como Mario Kart. Utilizando la programación orientada a objetos podríamos representar los elementos del juego con clases como:

* **Personaje**, con su nombre y otras propiedades.
* **Coche**, con sus características de velocidad, resistencia, funciones de aceleración, etc.
* **Circuito**, con su longitud, sus túneles, sus premios, etc.

## Cómo crear clases

Una clase es una estructura de programación que nos permite representar una entidad con sus propiedades y métodos. Es decir, una clase:

* Tiene **propiedades**: variables propias.
* **Hace** cosas: funciones.

Por ejemplo, la siguiente clase representa a un gato muy simple, con una función para maullar:

class Gato:  
 def maulla(self):  
 print("Miau")

Como se puede ver, para definir la clase utilizamos la palabra class seguida del nombre de la clase, con la primera letra en mayúscula. Todo lo que vaya dentro de ese bloque será parte de la clase.

Por otro lado, debes tener en cuenta de que **todas** las funciones de una clase deben tener el parámetros self, aunque no se use. Ese parámetro se refiere a la propia clase, y se utiliza para referirse a las propiedades y funciones de ella misma, como veremos ahora.

## Clase vs instancia

La clase no es más que la definición del gato. Pero para crear un gato en nuestro programa, debemos crear una instancia. Se hace así:

gato = Gato()  
gato.maulla()

Por pantalla veremos:

Miau

La instancia es un objeto concreto. La clase solamente es solamente la definición: un gato tiene un nombre y maulla. La instancia es un objeto concreto: Neko.



### Ejercicio 5.0

Escribe un programa que defina una clase llamada Persona que contenga los métodos dormir, comer y saludar. Dentro de cada método debe sacar algún texto por consola. Crea una instancia de la clase y llama a los distintos métodos.

class Persona:  
 def dormir (self):  
 print("ZZZZZ...")  
  
 def comer (self):  
 print("Ñam, Ñam...")  
  
 def saludar (self):  
 print("Hola, qué tal!")  
  
  
persona = Persona()  
  
persona.dormir()  
persona.comer()  
persona.saludar()

Resultado:

ZZZZZ...  
Ñam, Ñam...  
Hola, qué tal!

## Función constructora

Ese gato que hemos definido antes hace más bien poco. Vamos a darle una propiedad nombre. Además, vamos a crear una función especial que debe llamarse \_\_init\_\_.

\_\_init\_\_ es lo que se conoce como **función constructora**. Esta función se llama cuando se crea un objeto de la clase, y por tanto es el lugar ideal para iniciar las propiedades de la clase:

class Gato:  
 def \_\_init\_\_(self, nombre):  
 self.nombre = nombre  
   
 def maulla(self):  
 print("Miau, soy", self.nombre)

Ahora, cuando creemos objetos de la clase Gato le pasaremos un nombre y este se quedará como propiedad:

gato = Gato("Pixi")  
gato.maulla()  
  
otroGato = Gato("Cheto")  
otroGato.maulla()

Y en este caso veremos:

Miau, soy Pixi  
Miau, soy Cheto

### Ejercicio 5.1

Escribe un programa que defina una clase llamada Hola. La clase debe tener una función constructora con una propiedad llamada saludo. Esa propiedad se iniciará con la palabra "Hola". Además añadirás un método llamado decirHola, el cual mostrará por pantalla el contenido de la propiedad saludo.

class Hola:  
 def \_\_init\_\_(self):  
 self.saludo = "Hola"  
  
 def decirHola (self):  
 print(self.saludo)  
  
  
hola = Hola()  
hola.decirHola()

Resultado:

Hola

## Herencia

La herencia es un mecanismo que tienen las clases para reutilizar código. Supongamos que queremos hacer una clase que represente a un cachorro de gato. Queremos que haga lo mismo que la clase Gato pero que además ronronee. La clase cachorro podría heredar de la clase Gato, de la siguiente manera:

class Cachorro(Gato):  
 def ronronea(self):  
 print("Purrrr")

Ahora podemos hacer lo siguiente. Crear un objeto Cachorro, con las mismas propiedades que la clase Gato. De forma automática, heredará la propiedad nombre y el método maulla:

gatito = Cachorro("Lucifur")   
gatito.ronronea()  
gatito.maulla()

Se vería como:

Purrrr  
Miau, soy Lucifur

### super()

Cuando creas una subclase o una clase hija de otra, desde la clase que hereda puedes utilizar la función super() para llamar a funciones de la clase heredada.

Por ejemplo, en el caso anterior, desde la subclase Cachorro podríamos añadir un constructor propio y también llamar al constructor de la super-clase Gato:

class Cachorro(Gato):  
 def \_\_init\_\_(self, nombre):  
 super().\_\_init\_\_(nombre)  
 print("Gatete creado")  
  
 def ronronea(self):  
 print("Purrrr")



### Ejercicio 5.2

Escribe un programa que defina la clase Comida con el atributo nombre. Crea una subclase llamada Fruta que extienda la clase Comida con un constructor que recibe nombre y vitaminas, y un método llamado info que devuelva toda su información. vitaminas es una lista de nombres. Crea una instancia para probar la clase Fruta.

class Comida:  
 def \_\_init\_\_(self, nombre):  
 self.nombre = nombre  
  
  
class Fruta(Comida):  
 def \_\_init\_\_(self, nombre, vitaminas):  
 super().\_\_init\_\_(nombre)  
 self.vitaminas = vitaminas  
  
 def info (self):  
 # return f"{self.nombre} {self.vitaminas}";  
 return self.nombre + " " + str(self.vitaminas)  
  
postre = Fruta("Kiwi", ["A", "C"])  
print(postre.info())

Resultado:

Kiwi ['A', 'C']

## Encapsulación

En el ejemplo del gato, se puede hacer acceder a la propiedad nombre de forma directa. Para eso, en los objetos basta con poner algo así:

objeto.nombrePropiedad

El gato tiene una propiedad llamada nombre.

miGato = Gato("Pixi")  
print(miGato.nombre) # Pixi  
miGato.nombre = "Pixel"  
miGato.maulla() # Miau, soy Pixel

Acceder a una propiedad de forma tan directa no está mal, pero las buenas programadoras como yo tratamos de **encapsular** la clase. ¿Qué significa eso? Que no se permite que se pueda acceder o cambiar directamente las propiedades o métodos de la misma. Solo aquello que sea necesario para quienes usen la clase. Dicho de otra forma, los programadores deben tratar de crear clases que parezcan "cajas negras". En el caso de las propiedades como el nombre, en Python se puede añadir los siguientes métodos:

* Un método para **devolver** el valor de la propiedad nombre, también conocido como "getter":

@property   
 def nombre ():  
 return self.\_nombre

* Un método para poder **modificar** el valor del nombre, también conocido como "setter":

@nombre.setter  
 def nombre (nombre):  
 if nombre != "":  
 self.\_nombre = nombre

La clase quedaría así:

class Gato:  
 def \_\_init\_\_(self, nombre):  
 self.\_nombre = nombre  
   
 @property   
 def nombre ():  
 return self.\_nombre  
   
 @nombre.setter  
 def nombre (nombre):  
 if nombre != "":  
 self.\_nombre = nombre  
  
 def maulla(self):  
 print("Miau, soy", self.nombre)

Observa que la propiedad nombre es ahora \_nombre. Es una manera de expresar que esa propiedad es "privada", y que no se debería acceder a ella desde fuera de la clase.

Ahora cuando usemos la clase Gato e intentemos acceder a la propiedad . nombre, se hará a través de esas nuevas funciones.

miGato = Gato("Pixi")  
print(miGato.nombre) # llama al método `def nombre ():`  
miGato.nombre = "Pixel" # llama al método `def nombre (nombre):`  
miGato.maulla() # Miau, soy Pixel



### Ejercicio 5.3

Escribe un programa que defina la clase Vehiculo con el atributo matricula, con métodos get/set y otro método llamado arrancar. Crea una subclase llamada Coche que extienda la clase Vehículo con un constructor que recibe matricula, modelo y color, y una función llamada info que devuelva toda su información. Crea una instancia para probar la clase Coche.

class Vehiculo:  
 def \_\_init\_\_(self, matricula):  
 self.\_matricula = matricula  
  
 @property  
 def matricula (self):  
 return self.\_matricula  
  
 @matricula.setter  
 def matricula (self, matricula):  
 self.\_matricula = matricula  
  
 def arrancar (self):  
 print("Arrancando ", self.\_matricula)  
  
  
class Coche(Vehiculo):  
 def \_\_init\_\_(self, matricula, modelo, color):  
 super().\_\_init\_\_(matricula)  
 self.\_modelo = modelo  
 self.\_color = color  
  
 def info (self):  
 return f"{self.matricula} {self.\_modelo} {self.\_color}";  
  
  
coche = Coche("0042ASI", "Opel Corsa", "Blanco")  
coche.arrancar()  
print(coche.info())

Resultado:

Arrancando 0042ASI  
0042ASI Opel Corsa Blanco

¿Qué ventaja puede tener la encapsulación? Básicamente que desde "fuera" no se pueda manipular la clase sin control. De ahí que sea como una caja negra, como una videoconsola. Si para jugar un juego tuvieras que abrirla y soldar las conexiones a mano probablemente te acabarías cargando la consola. Por eso los aparatos se diseñan como cajas negras, solo te permiten manipularlas desde fuera.

En el caso de la clase Gato, no permitimos que el nombre se pueda cambiar directamente. A través de la función "setter" podemos controlar que el nombre que se quiera asignar sea correcto.

## Clases que contiene otras clases

Con la programación orientada a objetos, tratamos de representar cosas del mundo real a través de clases. Y esas clases pueden estar relacionadas unas con otras.

Por ejemplo, un Colegio puede tener Aulas, un aula puede tener Alumnos y un Profesor, etc. Las clases pueden contener por tanto, propiedades que en realidad son otras clases o incluso listas de otras clases.

class Alumno:  
 def \_\_init\_\_ (self, nombre):  
 self.\_nombre = nombre  
  
class Aula:  
 def \_\_init\_\_ (self):  
 self.\_alumnos = []  
   
 def meteAlumno (self, alumno):  
 self.\_alumnos.append(alumno)  
  
 def pasarLista (self):  
 for alumno in self.\_alumnos:  
 print(alumno.\_nombre)  
  
alumno1 = Alumno("Gumball")  
alumno2 = Alumno("Darwin")  
  
aula = Aula()  
  
aula.meteAlumno(alumno1)  
aula.meteAlumno(alumno2)  
aula.pasarLista()

Los diseños pueden ser tan complejos como sea necesario para representar lo que necesitemos.



### Ejercicio 5.4

Escribe un programa que defina la clase Piloto con el atributo nombre y las funciones get/set. Crea también una clase llamada Aeroplano con el atributo modelo, piloto y copiloto, con funciones get/set para el modelo y otro método llamado volar. Crea una instancia para probar ambas clases.

class Piloto:  
 def \_\_init\_\_(self, nombre):  
 self.\_nombre = nombre  
  
 @property  
 def nombre (self):  
 return self.\_nombre  
  
 @nombre.setter  
 def nombre (self, nombre):  
 self.\_nombre = nombre  
  
class Aeroplano:  
 def \_\_init\_\_(self, modelo, piloto, copiloto):  
 self.\_modelo = modelo  
 self.\_piloto = piloto  
 self.\_copiloto = copiloto  
  
 @property  
 def modelo (self):  
 return self.\_modelo  
  
 @modelo.setter  
 def modelo (self, modelo):  
 self.\_modelo = modelo  
  
 def volar (self):  
 return f"Volando {self.\_modelo} {self.\_piloto.nombre} con {self.\_copiloto.nombre}"  
  
  
piloto1 = Piloto("Han Solo")  
piloto2 = Piloto("Murdock")  
avioneta = Aeroplano("AirBluff 727", piloto1, piloto2)  
  
print(avioneta.volar())

Resultado:

Volando AirBluff 727 Han Solo con Murdock

## Métodos estáticos

Normalmente, para poder utilizar una clase siempre creamos una instancia de la misma, como hacíamos en el ejemplo anterior:

alumno1 = Alumno("Gumball")

En determinadas ocasiones, puede que nos interese crear una clase de la que no queramos hacer copias y que solo sirva para hacer una tarea concreta, como si fuera una función.

Por ejemplo, podemos hacer una clase que dado un nombre, le dé un formato correcto, con la primera letra mayúscula y el resto de letras en minúsculas:

class Formato:  
 @staticmethod  
 def formato (nombre):  
 return nombre[0].upper() + nombre[1:].lower()  
  
print(Formato.formato("gUmBaLl"))

Resultado:

Gumball

### Ejercicio 5.5

Escribe un programa que defina una clase llamada Numero y una función estática llamado aleatorio(max). Esta función debe devolver un número dentro del intervalo 0 y max.

import random  
  
class Numero:  
 @staticmethod  
 def aleatorio (max):  
 return random.randint(0, max)  
  
for i in range(5):  
 print(Numero.aleatorio(10))

Resultado:

4  
3  
0  
9  
1

Meter métodos estáticos en una clase puede resultar útil cuando queremos agrupar unas funciones en una única clase, y no queremos crear distintas instancias, sino utilizar funciones concretas.

## Ejercicios propuestos

### Ejercicio 5.0

Escribe un programa que defina una clase llamada Instrumento. El constructor debe tener los parámetros nombre y tipo, que se asignarán a los atributos \_nombre y \_tipo respectivamente. Además debes añadir una función llamada tocar que simplemente sacará un mensaje y otra llamada info que devolver un texto con la información de los atributos. Crea una instancia de la clase y llama a sus funciones.

class Instrumento:  
 def \_\_init\_\_(self, nombre, tipo):  
 self.\_nombre = nombre  
 self.\_tipo = tipo  
  
 def tocar (self):  
 print("Tocando ", self.\_nombre)  
  
 def info (self):  
 return f"{self.\_nombre} {self.\_tipo}"  
  
  
miGuitarra = Instrumento("Guitarra", "cuerda")  
miGuitarra.tocar()  
print(miGuitarra.info())

Resultado:

Tocando Guitarra  
Guitarra cuerda

### Ejercicio 5.1

Escribe un programa que defina una clase llamada NombreFormateado, con un constructor que recibe un nombre y un apellido y una función llamada formatear que debe devolver el nombre y el apellido en este formato: "Nombre Apellido", es decir con la primera letra en mayúscula, el resto en minúscula y separados por comas. Crea las funciones auxiliares que consideres oportunas.

class NombreFormateado:  
 def \_\_init\_\_(self, nombre, apellido):  
 self.\_nombre = nombre  
 self.\_apellido = apellido  
  
 def formatear (self):  
 return self.\_corregir(self.\_nombre) + " " + self.\_corregir(self.\_apellido)  
  
 def \_corregir (self, cadena):  
 return self.\_primero(cadena) + self.\_resto(cadena)  
  
 def \_primero (self, cadena):  
 return cadena[0].upper()  
  
 def \_resto (self, cadena):  
 return cadena[1:len(cadena)].lower()  
  
  
formateador = NombreFormateado("JUAN", "PÉREZ")  
print(formateador.formatear())

Resultado:

Juan Pérez

### Ejercicio 5.2

Escribe un programa que defina una clase llamada Sumador, la cual se instancia con dos números. Incluye funciones get y set para ambos, y debes controlar que si se les intenta asignar un valor negativo se asigne 0. Además tendrán la función sumar que devolverá la suma de ambos números.

class Sumador:  
 def \_\_init\_\_(self, valor1, valor2):  
 self.valor1 = valor1  
 self.valor2 = valor2  
  
 @property  
 def valor1 (self):  
 return self.\_valor1  
  
 @valor1.setter  
 def valor1 (self, valor1):  
 if valor1 > 0:  
 self.\_valor1 = valor1  
 else:  
 self.\_valor1 = 0  
  
 @property  
 def valor2 (self):  
 return self.\_valor2  
  
 @valor2.setter  
 def valor2 (self, valor2):  
 if valor2 > 0:  
 self.\_valor2 = valor2  
 else:  
 self.\_valor2 = 0  
  
 def sumar (self):  
 return self.\_valor1 + self.\_valor2  
  
  
sumador = Sumador(28, 14)  
print(sumador.sumar())  
  
sumador.valor1 = 600  
sumador.valor2 = 66  
print(sumador.sumar())

Resultado:

42  
666

### Ejercicio 5.3

Crea un programa con una clase llamada Moneda. La clase debe tener un constructor vacío y una única función llamado tirar cuyo resultado debe ser un *string* elegido al azar entre "cara" o "cruz". Crea una instancia de la clase para probarla.

import random  
  
def aleatorio (max):  
 return random.randint(0, max)  
  
class Moneda:  
 def tirar (self):  
 lados = ["cara", "cruz"]  
 numero = aleatorio(1)  
  
 return lados[numero]  
  
moneda = Moneda()  
  
for i in range(10):  
 print(moneda.tirar())

Resultado:

cara  
cara  
cruz  
cruz  
cruz  
...

### Ejercicio 5.4

Crea un programa con una clase llamada Dado para simular el comportamiento de un dado de N caras. Crea una instancia de la clase para probarla.

* ​constructor def \_\_init\_\_(self, lados, admiteCero=False): con el atributo lados: atributo que guarda el número de caras y el atributo admiteCero = False: atributo que nos dice si el dado puede devolver el valor 0. Por defecto vale False.
* ​setter def lados (self, lados) : función setter con parámetro, establece el atributo lados.
* ​setter def admiteCero (self, lados, admiteCero): función con parámetros, establece los dos atributos.
* ​def tirar (self): función que simula el lanzamiento del dado y retorna un el resultado. Debe tener en cuenta al atributo admiteCero. Crea instancias que genere un dado de 6 caras, un dado de 10 caras y un dado de 20 que permita ceros, y haz 100 lanzamientos de cada uno:

import random  
  
def aleatorio (max):  
 return random.randint(0, max)  
  
class Dado:  
 def \_\_init\_\_(self, lados = 6, admiteCero = False):  
 self.\_lados = lados  
 self.\_admiteCero = admiteCero  
  
 @property  
 def lados (self):  
 return self.\_lados  
  
 @lados.setter  
 def lados (self, lados):  
 self.\_lados = lados  
  
 @property  
 def admiteCero (self):  
 return self.\_admiteCero  
  
 @admiteCero.setter  
 def admiteCero (self, admiteCero):  
 self.\_admiteCero = admiteCero  
  
 def tirar (self):  
 numero = aleatorio(self.\_lados)  
  
 if not self.\_admiteCero:  
 numero = numero + 1  
  
 return numero  
  
  
  
dado = Dado()  
for i in range(10):  
 print(dado.tirar())

Resultado:

4  
3  
2  
4  
3  
...

### Ejercicio 5.5

Crea un programa que contenga dos clases:

* Clase Jugador, que contiene los atributos nombre, puesto y dorsal. También tiene un constructor con todos esos parámetros y una función llamada informe que retorne todos los atributos.
* Clase Equipo, que contiene los atributos nombre, fundacion, presupuesto y una lista para guardar instancias de la clase Jugador. Debe tener un constructor con los atributos nombre, fundacion, presupuesto, sus get/set, una función informe y otras dos funciones:
* def fichar(self, jugador) para añadir jugadores a la lista.
* def mostrarJugadores(self), para devolver una cadena con todos los datos de los jugadores Además, añade el código necesario para crear dos jugadores y un equipo, al que añadirás los jugadores y los mostrarás.

class Jugador:  
 def \_\_init\_\_(self, nombre, posicion, dorsal):  
 self.\_nombre = nombre  
 self.\_posicion = posicion  
 self.\_dorsal = dorsal  
  
 def informe (self):  
 return f"{self.\_nombre} {self.\_posicion} {self.\_dorsal}"  
  
  
class Equipo:  
 def \_\_init\_\_ (self, nombre, fundacion, presupuesto):  
 self.\_nombre = nombre  
 self.\_fundacion = fundacion  
 self.\_presupuesto = presupuesto  
 self.\_jugadores = []  
  
 def ficharJugador (self, jugador):  
 self.\_jugadores.append(jugador)  
  
 def mostrarJugadores (self):  
 for jugador in self.\_jugadores:  
 print(jugador.informe())  
  
  
jugador1 = Jugador("Maradona", "Delantero", 10)  
jugador2 = Jugador("Beckenbauer", "Defensa", 4)  
  
print(jugador1.informe())  
  
equipo = Equipo("New Team", 1983, 39944.45)  
equipo.ficharJugador(jugador1)  
equipo.ficharJugador(jugador2)  
  
equipo.mostrarJugadores()

Resultado:

Maradona Delantero 10  
Beckenbauer Defensa 4

### Ejercicio 5.6

Crea un programa que incluya una serie de clases.

* 1 - Clase Dispositivo: tiene los atributos nombre, marca y precio. Un constructor usando los atributos, los set y get y una función toString mostrando los atributos.
* 2 - Clase Movil: es una subclase de Dispositivo, hay que añadir el atributo numero. Crea el constructor y el método def toString (self) aprovechando los de la superclase. Añade la función def llamar (self, numero), que saque por pantalla una cadena diciendo "llamando numero".
* 3 - Clase Ordenador: es una subclase de Dispositivo, hay que añadir el atributo procesador. Crea el constructor y la función def toString (self) aprovechando los de la superclase Además, añade el código necesario para crear un móvil y un ordenador y los muestras.

class Dispositivo:  
 def \_\_init\_\_(self, nombre, precio):  
 self.\_nombre = nombre  
 self.\_precio = precio  
  
 def get\_nombre ():  
 return self.\_nombre  
  
 def set\_nombre (nombre):  
 self.\_nombre = nombre  
  
 def get\_precio ():  
 return self.\_precio  
  
 def set\_precio (precio):  
 self.\_precio = precio  
  
 def toString (self):  
 return f"{self.\_nombre} {self.\_precio}";  
  
  
class Movil(Dispositivo):  
 def \_\_init\_\_(self, nombre, precio, numero):  
 super().\_\_init\_\_(nombre, precio)  
 self.\_numero = numero  
  
 @property  
 def numero (self):  
 return self.\_numero  
  
 @numero.setter  
 def numero (self, numero):  
 self.\_numero = numero  
  
 def toString (self):  
 return f"{super().toString()} {self.\_numero}"  
  
 def llamar (numero):  
 print("Llamando a", numero)  
  
  
class Ordenador(Dispositivo):  
 def \_\_init\_\_(self, nombre, precio, procesador):  
 super().\_\_init\_\_(nombre, precio)  
 self.\_procesador = procesador  
  
 @property  
 def procesador (self):  
 return self.\_procesador  
  
 @procesador.setter  
 def procesador (self, procesador):  
 self.\_procesador = procesador  
  
 def toString (self):  
 return f"{super().toString()} {self.\_procesador}"  
  
  
ordenador = Ordenador("Dell", 4553.4, "Lentium 4")  
telefono = Movil("Chanmhung", 434.4, 665745345)  
  
print("Ordenador: ", ordenador.toString())  
print("Teléfono: ", telefono.toString())

Resultado:

Ordenador Dell 4553.4 Lentium 4  
Teléfono Chanmhung 434.4 665745345

### Ejercicio 5.7

Vamos a crear el proyecto Caperucita en el que la protagonista gestiona una Cesta de comida. La comida será de varios tipos. Estas son las clases que se deben hacer,

* 1 - Clase Comida: tiene los atributos nombre y peso. Un constructor usando los atributos, los set y get y una función toString mostrando los atributos.
* 2 - Clase Fruta: es una subclase de Comida, y hay que añadir el atributo vitamina. Crea el constructor y la función toString aprovechando los de la superclase.
* 3 - Clase Caramelo: es una subclase de Comida y hay que añadir el atributo calorias. Crea el constructor y la funciñon toString aprovechando los de la superclase.
* 4 - Clase Cesta, tiene un atributo llamado alimentos que es un array de elementos tipo Comida (Fruta o Caramelo). Se inicia en el constructor. Tiene tres funciones:
  + def meterComida(self, comida) para meter una comida en la cesta,
  + def pesoTotal(self) devuelve el peso total de la comida de la cesta.
  + def toString(self) para mostrar toda la comida de la cesta. Además, añade el código necesario para crear instancias de Fruta, Caramelo y añádelos a la instancia de Cesta.

class Comida:  
 def \_\_init\_\_(self, nombre, peso):  
 self.\_nombre = nombre  
 self.\_peso = peso  
  
 @property  
 def nombre (self):  
 return self.\_nombre  
  
 @nombre.setter  
 def nombre (self, nombre):  
 self.\_nombre = nombre  
  
 @property  
 def peso (self):  
 return self.\_peso  
  
 @peso.setter  
 def peso (self, peso):  
 self.\_peso = peso  
  
 def toString (self):  
 return f"{self.\_nombre} {self.\_peso}"  
  
  
class Fruta(Comida):  
 def \_\_init\_\_(self, nombre, peso, vitamina):  
 super().\_\_init\_\_(nombre, peso)  
 self.\_vitamina = vitamina  
  
 @property  
 def vitamina (self):  
 return self.\_vitamina  
  
 @vitamina.setter  
 def vitamina (self, vitamina):  
 self.\_vitamina = vitamina  
  
 def toString (self):  
 return f"{super().toString()} {self.\_vitamina}"  
  
  
class Caramelo(Comida):  
 def \_\_init\_\_(self, nombre, peso, calorias):  
 super().\_\_init\_\_(nombre, peso)  
 self.\_calorias = calorias  
  
 @property  
 def calorias (self):  
 return self.\_calorias  
  
 @calorias.setter  
 def calorias (self, alorias):  
 self.\_calorias = calorias  
  
 def toString (self):  
 return f"{super().toString()} {self.\_calorias}"  
  
  
class Cesta:  
 def \_\_init\_\_(self):  
 self.\_alimentos = []  
  
 def meterComida (self, comida):  
 self.\_alimentos.append(comida)  
  
 def pesoTotal (self):  
 total = 0  
 for alimento in self.\_alimentos:  
 total += alimento.peso  
  
 return total  
  
 def toString (self):  
 info = ""  
 for alimento in self.\_alimentos:  
 info = info + alimento.toString() + "\n"  
  
 return info  
  
  
chicle = Caramelo("Cheiw", 0.2, 100)  
gominola = Caramelo("Fresa", 0.3, 210)  
pera = Fruta("Pera", 0.1, "B")  
manzana = Fruta("Manzana", 0.15, "A")  
  
cesta = Cesta()  
cesta.meterComida(chicle)  
cesta.meterComida(gominola)  
cesta.meterComida(pera)  
cesta.meterComida(manzana)  
  
print("Contenido cesta: ", cesta.toString())  
print("Peso total:", cesta.pesoTotal())

# Excepciones



A estas alturas puede que ya seas excelente en programación. Pero aún así, tus programas pueden seguir fallando, porque hay cosas que escapan al control de tu programa y hacen que tu programa deje de funcionar. Por ejemplo, si tu programa espera que el usuario escriba un número pero este escribe letras o no escribe nada, tu programa fallará. Si tu programa tiene que leer un fichero pero este fichero no existe, tu programa fallará. Si tu programa necesita conectarse a la red pero tu ordenador no está conectado, tu programa fallará. Como puedes ver, hay situaciones sobre las que el programa no puede tener control. Por suerte tenemos un mecanismo que nos permite que si se produce una de esas sesiones, al menos nuestro programa no falle y termine sin más. Y ese mecanismo son las Excepciones.

## Excepciones en Python

Por ejemplo, supongamos que tenemos un programa muy simple como el siguiente, en el que se le pide un número al usuario y se hace una multiplicación:

valor = input("Introduce un número: ")  
valor = int(valor)  
cuadrado = valor \* valor  
print("El cuadrado es: ", cuadrado)

Si el usuario introduce lo que no debe, veremos lo siguiente:

Introduce un número: x  
Traceback (most recent call last):  
 File "saltaexcepcion.py", line 2, in <module>  
 valor = int(valor)  
ValueError: invalid literal for int() with base 10: 'x'

Mediante una excepción, podemos evitar que el programa falle estrepitosamente y al menos mostrar un mensaje de error al usuario. La excepción es una estructura más en el lenguaje, y tiene la siguiente forma:

try:  
 código\_que\_puede\_fallar  
except:  
 código\_que\_se\_ejecuta\_si\_hay\_error

Veamos el ejemplo anterior, protegiendo el código sensible dentro del bloque de excepción:

valor = input("Introduce un número: ")  
  
try:  
 valor = int(valor)  
 cuadrado = valor \* valor  
 print("El cuadrado es: ", cuadrado)  
except:  
 print("Error al convertir dato!")

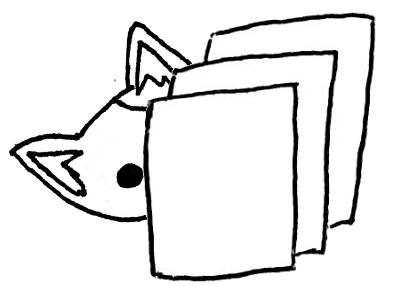
Ahora si se mete un dato incorrecto, veremos lo siguiente:

Introduce un número: x  
Error al convertir dato!

También se podría mejorar el programa para que volviera a intentar solicitar el valor y no terminar.

Existen excepciones específicas según el tipo de error, las cuales se pueden usar para mostrar un mensaje de error más específico.

# Manejo de ficheros



Hasta ahora hemos manejado pocos datos, lo que el usuario escribe por pantalla o lo que tenemos en variables. Pero si queremos utilizar cantidades más grandes de datos, podemos leer y escribir en ficheros. Existen toda clase de ficheros, desde texto, multimedia (música, vídeo) hasta binarios. Todos ellos se pueden manejar desde un programa. Como una introducción, vamos a ver cómo podemos manejar ficheros de texto.

## Lectura de ficheros

Para poder leer un fichero, necesitamos por un lado que exista ese fichero, luego ya lo podemos abrir y leer. En el siguiente código, se lee un fichero que está en el mismo sitio que el propio programa:

fichero = open("texto.txt", "r")  
contenido = fichero.read()  
print(contenido)  
fichero.close()

A tener en cuenta:

* Para leer el fichero, primero hay que abrirlo con open
* Al abrir el fichero hay que indicar su nombre y si está en otro directorio, la ruta hacia el mismo. Si estuviera dentro de carpeta fichero, la ruta sería fichero/texto.txt.
* El parámetro "r" indica que el fichero lo leemos solo en modo lectura o read. El fichero de texto podría ser algo así, y el programa mostraría eso mismo por pantalla.

Este es un texto  
de varias líneas  
Y se puede leer  
muy fácilmente

### ¿Leyendo línea a línea?

En el ejemplo anterior, hemos leído todo el contenido del fichero de golpe, guardándolo en una variable de texto. Pero a veces, puede que nos interese leer el fichero línea a línea. Para ello debemos utilizar la función readline como se ve a continuación:

fichero = open("texto.txt", "r")  
lineas = fichero.readline()  
for linea in lineas:  
 print(linea)  
  
fichero.close()

## Ficheros JSON

Los ficheros de texto simples como el anterior pueden contener información, pero no son datos muy manejables para un programa. Si queremos leer o guardar datos que un programa pueda manipular fácilmente, conviene usar algún formato concreto. Uno de los formatos más populares en programación es el formato JSON. Es un formato que se parece a las estructuras de diccionario en Python. Incluso también tiene la opción de representar listas como las del lenguaje. El siguiente contenido está en formato JSON. Se trata de una lista que contiene varios objetos. Si te fijas, los objetos en JSON *¡son iguales a los diccionarios en Python!*

[  
 {"id": 66, "nombre": "Ada"},  
 {"id": 2, "nombre": "Neko"},  
 {"id": 4, "nombre" : "Bug"}  
]

Lo bueno de ese formato es que se puede importar a nuestro programa Python fácilmente, siempre que sea correcto, claro.

Para poder leer ese contenido y convertir esos datos en una lista de diccionarios, utilizaremos la librería json. Mediante la función json.load podremos cargar de forma automática el contenido de ese fichero en una variable. A partir de ahí podremos manejar todo ese contenido como una lista, ¡donde cada elemento es un diccionario!:

import json  
  
fichero = open("texto.json", "r")  
contenido = json.load(fichero)  
  
for personaje in contenido:  
 print(personaje["nombre"])  
  
fichero.close()

Por pantalla veríamos:

Ada  
Neko  
Bug

## Escritura de ficheros

Para escribir ficheros, el proceso es similar pero debemos hacer dos cosas:

* Abrir el fichero en modo escritura.
* Utilizar la función write para escribir contenido.

Con el siguiente código, escribiremos un par de líneas de texto en el fichero:

fichero = open("texto.txt", "w")  
fichero.write("Escribo una línea\n")  
fichero.write("Escribo otra línea\n")  
fichero.close()



¡OJO!

Si escribimos de esa manera, estaremos machacando el contenido del fichero. Contenido del fichero:

Escribo una línea  
Escribo otra línea

Si queremos escribir añadiendo contenido, debemos abrir el fichero en modo "a":

fichero = open("texto.txt", "a")  
fichero.write("Añado una línea\n")  
fichero.write("Añado otra línea\n")  
fichero.close()

Ahora el contenido del fichero sería:

Escribo una línea  
Escribo otra línea  
Añado una línea  
Añado otra línea

## Escribir en un fichero json

En el caso de un fichero en formato JSON, lo que nos tiene que preocupar es que al momento de escribir, convirtamos nuestros datos a texto. Por suerte para eso hay una función que lo hace de forma automática: json.dumps()

En el siguiente ejemplo, se carga el contenido de un fichero json dentro de una variable. Luego añadimos un elemento a esa lista. Abrimos el fichero otra vez, en modo escritura, y hacemos un write utilizando json.dumps para convertir el contenido en texto:

import json  
  
fichero = open("texto.json", "r")  
contenido = json.load(fichero)  
fichero.close()  
  
personaje = { "id": 666, "nombre": "Gumball" }  
contenido.append(personaje)  
  
fichero = open("texto.json", "w")  
fichero.write(json.dumps(contenido))  
fichero.close()

# Librerías



Conforme los programas se hacen más y más complejos, es probable que tengamos que definir muchas funciones, o separar el diseño en clases, etc. Pese a que podríamos tener todo dentro del mismo fichero, no sería la mejor forma de organizar nuestro código. Lo ideal es que separemos cada clase en su propio fichero y cada función o grupo de funciones en su propio fichero.

Una vez organizado el código en ficheros e incluso en carpetas, ya podemos reutilizarlos en otros ficheros. Veamos un ejemplo simple.

Definimos la siguiente función en un fichero llamado mates.py:

def sumar (a, b):  
 return a + b  
  
def restar (a, b):  
 return a - b  
  
def incrementar (a):  
 return a - 1

Ahora podemos incluir ese fichero en otro programa mediante la orden import. Si están en el mismo directorio se puede hacer simplemente:

import mates  
  
valor1 = 5  
valor2 = 10  
  
resultado = mates.sumar(valor1, valor2)  
print(resultado) # 15

### Ejercicio 6.0

En un ejercicio anterior, se proponía hacer un generador de contraseñas. Utiliza el mismo código pero créalo dentro de un fichero. Crea otro fichero donde debes importar ese código y utilizarlo.

Fichero generar.py:

import random  
  
def aleatorio (max):  
 return random.randint(0, max - 1)  
  
def generarPassword (longitud):  
 caracteres = ["a","b","c","d","e","f","g","h","i","j","k","l",  
 "m","n","ñ","o","p","q","r","s","t","u","v","w","x","y","z",  
 "0","1","2","3","4","5","6","7","8","9",".","-","\_","!","$"]  
 password = ""  
  
 for i in range(longitud):  
 caracter = caracteres[aleatorio(len(caracteres))]  
 password = password + caracter  
  
 return password

Y lo usamos en fichero 1.py:

import generar  
  
password = generar.generarPassword(8)  
  
print(password)

Resultado:

g3ep-ahx

Con las clases se puede hacer exactamente lo mismo. Supongamos que tenemos una clase llamada LectorPantalla en un fichero llamado lector\_pantalla.py. Es una clase que nos permite leer datos desde la consola:

class LectorPantalla:  
 def leerEntero (self, mensaje = "Introduce un número: "):  
 numero = input(mensaje)  
 return int(numero)  
  
 def leerTexto (self, mensaje = "Introduce texto: ")  
 texto = input(mensaje)  
 return texto

Ahora podemos reutilizar esa clase en otro fichero, junto con nuestro fichero mates.

import lector\_pantalla  
import mates  
  
lector = lector\_pantalla.LectorPantalla()  
valor1 = lector.leerEntero()  
  
print(mates.incrementar(valor1))

Por pantalla se podría ver algo así:

Introduce un número: 6  
7



### Ejercicio 6.1

Define una clase llamada Menu que tenga las siguientes funciones:

* def init\_\_(self): recibe como parámetro una lista de opciones (*strings*).
* def mostrar (self): muestra las opciones precedidas de un número llamando a print.
* def seleccionar(self, numero): devuelve True si el número seleccionado está en el menú, en caso contrario devuelve False. Luego importa esa clase en el fichero 2.py y utilízala.

Fichero menu.py:

class Menu:  
 def \_\_init\_\_ (self, opciones):  
 self.\_opciones = opciones  
  
 def mostrar (self):  
 for i in range(len(self.\_opciones)):  
 print(f"{i+1} {self.\_opciones[i]}")  
  
 def seleccionar (self, numero):  
 return numero > 0 and numero <= len(self.\_opciones)

Fichero 2.py:

import menu  
miMenu = menu.Menu(["Mostrar", "Eliminar", "Salir"])  
  
miMenu.mostrar()  
  
if miMenu.seleccionar(1):  
 print("Opción 1 presente")  
else:  
 print("Opción 1 no presente")

Resultado:

1 Mostrar  
2 Eliminar  
3 Salir  
Opción 1 presente

## Ejercicios propuestos

### Ejercicio 6.0

Crea una clase llamada Fichero con las siguientes funciones:

* def \_\_init\_\_(self, fichero): recibe como parámetro el fichero a abrir.
* def leer(self): devuelve una cadena con el contenido del fichero.
* def escribir(self, contenido): escribe en el fichero el contenido que se le pasa como parámetro. Luego debes utilizar la clase importándola en otro fichero.

Fichero fichero.py:

class Fichero:  
 def \_\_init\_\_ (self, nombreFichero):  
 self.\_nombreFichero = nombreFichero  
  
 def leer (self):  
 fichero = open(self.\_nombreFichero, "r")  
 datos = fichero.read()  
 fichero.close()  
  
 return datos  
  
 def escribir (self, contenido):  
 fichero = open(self.\_nombreFichero, "w+")  
 fichero.write(contenido)  
 fichero.close()

Fichero 6.0.py:

from datetime import date  
  
import fichero  
  
miFichero = fichero.Fichero("6.0.txt")  
  
print("Contenido anterior: ", miFichero.leer())  
  
miFichero.escribir("Contenido cambiado!!! " + str(date.today()))  
print("Contenido: ", miFichero.leer())

Fichero de texto 3.txt:

Este es el contenido actual.

Resultado:

Contenido anterior: Contenido cambiado!!! 2020-08-18  
Contenido: Contenido cambiado!!! 2020-08-23

### Ejercicio 6.1

Crea una clase llamada Listado con las siguientes funciones:

* def \_\_init\_\_(self, fichero): recibe como parámetro el nombre de un fichero json cuyo contenido debe cargar en una lista llamada \_datos que se definirá como atributo. El contenido debe ser una lista de diccionarios con el formato {"id": 1, "nombre": "Juan"}
* def existe(self, nombre): debe devolver True/False si el nombre que se pasa como parámetro existe en la lista.
* def aMinusculas(self): debe pasar todos los nombres de la lista a minúsculas.
* def posicion(self, nombre): debe devolver la posición donde se encuentra ese nombre. Luego debes utilizar la clase importándola en otro fichero.

Fichero listado.py:

import json  
  
class Listado:  
 def \_\_init\_\_ (self, nombreFichero):  
 contenido = open(nombreFichero, "r")  
 self.\_datos = json.load(contenido)  
 contenido.close()  
  
 def existe (self, nombre):  
 for dato in self.\_datos:  
 if dato["nombre"] == nombre:  
 return True  
  
 return False  
  
 def aMinusculas (self):  
 self.\_datos = list(map(lambda dato: { "id": dato["id"], "nombre": dato["nombre"].lower() }, self.\_datos))  
  
 def posicion (self, nombre):  
 i = 0  
 for dato in self.\_datos:  
 if dato["nombre"] == nombre:  
 return i  
 i += 1  
 return -1  
  
 def print (self):  
 for dato in self.\_datos:  
 print(dato)

Fichero 6.1.py:

import listado  
miListado = listado.Listado("6.1.json")  
  
if miListado.existe("eugene"):  
 print("Existe!")  
  
miListado.aMinusculas()  
miListado.print()  
  
if miListado.existe("eugene"):  
 print("Existe!")  
 print(miListado.posicion('eugene'))

Fichero 6.1.json:

[  
 {  
 "id": 3,  
 "nombre": "Juan"  
 },  
 {  
 "id": 5,  
 "nombre": "Eugene"  
 },  
 {  
 "id": 10,  
 "nombre": "Paul"  
 }  
]

Resultado:

{'id': 3, 'nombre': 'juan'}  
{'id': 5, 'nombre': 'eugene'}  
{'id': 10, 'nombre': 'paul'}  
Existe!  
1

### Ejercicio 6.2

Crea una clase llamada Tareas con las siguientes funciones:

* def \_\_init\_\_(self): debe abrir el fichero llamado tareas.json y cargar en una lista los diccionarios que tendrán el siguiente formato: {id: 1, tarea: "Aprende algo"}. Esa lista será un atributo.
* def crear(self, tarea): genera un nuevo objeto y lo guarda en la lista.
* def eliminar(self, id): elimina una tarea de la lista que tenga ese id.
* def guardar(self): guarda la lista en el fichero tareas.json.
* def mostrar(self): devuelve todas las tareas en un string Luego debes utilizar la clase importándola en otro fichero.

Fichero tareas.py

import json  
  
class Tareas:  
 def \_\_init\_\_ (self):  
 fichero = open("tareas.json", "r")  
 self.\_tareas = json.load(fichero)  
 fichero.close()  
  
 def crear (self, id, tarea):  
 nueva = { "id": id, "tarea": tarea };  
 self.\_tareas.append(nueva)  
  
 def guardar (self):  
 fichero = open("tareas.json", "w")  
 fichero.write(json.dumps(self.\_tareas))  
 fichero.close()  
  
 def eliminar(self, id):  
 self.\_tareas = list(filter(lambda dato: dato["id"] != id, self.\_tareas))  
  
 def mostrar (self):  
 resultado = ""  
 for dato in self.\_tareas:  
 resultado += json.dumps(dato) + "\n"  
  
 return resultado

Fichero tareas.json: javascript [ {"tarea": "Aprender Go", "id": 3}, {"tarea": "Investigar Rust", "id": 5}, {"tarea": "Dormir más", "id": 10} ]

Fichero 6.2.py:

import tareas  
misTareas = tareas.Tareas()  
  
print(misTareas.mostrar(), "\n---")  
  
misTareas.crear(2, "Comer")  
print(misTareas.mostrar(), "\n---")  
  
misTareas.eliminar(2)  
print(misTareas.mostrar(), "\n---")  
  
misTareas.crear(66, "Leer")  
print(misTareas.mostrar(), "\n---")  
misTareas.guardar()

Resultado:

{"tarea": "Aprender Go", "id": 3}  
{"tarea": "Investigar Rust", "id": 5}  
{"tarea": "Dormir m\u00e1s", "id": 10}  
   
---  
{"tarea": "Aprender Go", "id": 3}  
{"tarea": "Investigar Rust", "id": 5}  
{"tarea": "Dormir m\u00e1s", "id": 10}  
{"tarea": "Comer", "id": 2}  
   
---  
{"tarea": "Aprender Go", "id": 3}  
{"tarea": "Investigar Rust", "id": 5}  
{"tarea": "Dormir m\u00e1s", "id": 10}  
   
---  
{"tarea": "Aprender Go", "id": 3}  
{"tarea": "Investigar Rust", "id": 5}  
{"tarea": "Dormir m\u00e1s", "id": 10}  
{"tarea": "Leer", "id": 66}  
   
---

### Ejercicio 6.3

Crea una clase llamada Jugador que contenga lo siguiente:

* def \_\_init\_\_(self, nombre, dorsal): asigna los parámetros a los atributos \_nombre y \_dorsal.
* Métodos get/set para nombre y dorsal.
* def info(self): devuelve un *string* con la información del jugador.

Crea una clase llamada Equipo que contenga lo siguiente:

* def cargar(self): debe abrir un fichero llamado jugadores.json que contendrá una lista de diccionarios de jugador [{nombre: "Pele", dorsal: 10},{…},…]. Y debe crear por cada objeto del fichero una instancia de la clase Jugador y meterla en una lista llamado this.\_jugadores.
* def mostrar(self): debe mostrar toda la lista de jugadores.
* def fichaje(self, nombre, dorsal): debe introducir un jugador nuevo en la lista, creando una instancia de Jugador. En la clase Equipo tendrás que importar la clase Jugador para poder utilizarla.

Fichero jugador.py:

class Jugador:  
 def \_\_init\_\_ (self, nombre, dorsal):  
 self.\_nombre = nombre  
 self.\_dorsal = dorsal  
  
 @property  
 def nombre (self):  
 return self.\_nombre  
  
 @nombre.setter  
 def nombre (self, nombre):  
 self.\_nombre = nombre  
  
 @property  
 def dorsal (self):  
 return self.\_dorsal  
  
 @dorsal.setter  
 def dorsal (self, dorsal):  
 self.\_dorsal = dorsal  
  
 def toString (self):  
 return f"{self.\_nombre} {self.\_dorsal}"

Fichero equipo.py:

import json  
import jugador  
  
class Equipo:  
 def cargar (self):  
 contenido = open("./jugadores.json")  
 jugadores = json.load(contenido)  
 print("Loaded: ", jugadores)  
 self.\_jugadores = []  
 for j in jugadores:  
 self.\_jugadores.append(jugador.Jugador(j["nombre"], j["dorsal"]))  
  
 def fichaje (self, nombre, dorsal):  
 nuevoFichaje = jugador.Jugador(nombre, dorsal)  
 self.\_jugadores.append(nuevoFichaje)  
  
 def mostrar (self):  
 for jugador in self.\_jugadores:  
 print(jugador.toString())

Fichero jugadores.json:

[  
 {  
 "nombre": "Maradona",  
 "dorsal": 10  
 },  
 {  
 "nombre": "Pele",  
 "dorsal": 8  
 }  
]

Fichero 6.3.py:

import equipo  
miEquipo = equipo.Equipo()  
  
miEquipo.cargar()  
miEquipo.mostrar()  
miEquipo.fichaje("Gento", 11)  
miEquipo.mostrar()

Resultado:

console Loaded: [{'dorsal': 10, 'nombre': 'Maradona'}, {'dorsal': 8, 'nombre': 'Pele'}] Maradona 10 Pele 8 Maradona 10 Pele 8 Gento 11# Apéndices

# Sobre Python

¿Por qué hemos elegido Python? Por sus muchas virtudes. Es un lenguaje interpretado y con una sintaxis muy sencilla, lo cual lo hace muy sencillo de aprender. No hay que preocuparse (mucho) de los tipos y de las rigideces del lenguaje. La única que debemos tener en cuenta es el **de respetar las tabulaciones** en cada bloque.

El objetivo no es aprender el lenguaje en si, lo esencial es aprender a programar, y Python facilita esa tarea.

Además, se trata de un lenguaje muy útil, muy extendido y utilizado profesionalmente. Por si eso fuera poco, es muy apreciado por los desarrolladores, lo cual supone una comunidad inmensa de personas aportando código, librerías y ayuda en la web.

Python tiene dos versiones algo diferenciadas, la 2 y la 3. En este libro hemos procurado utilizar la sintaxis y el estilo de la 3, por utilizar la que está más al día y porque la 2 está en desuso.

## Instalando Python localmente

Basta con ir al [site de python](https://www.python.org) (https://python.org) y descargar el instalador de la versión 3. La instalación varía un poco en función de tu sistema pero básicamente sería la siguiente:

* **Windows**: descargar el instalador, ejecutarlo y confirmar cada paso.
* **Mac**: exactamente igual.
* **Linux**: probablemente lo tengas instalado de serie o probablemente no necesites que te digan cómo instalarlo ;)

## Editores de código

Si quieres utilizar un editor para el código, dispones de muchos donde elegir pero destacaremos los siguientes:

* [pycharm](http://www.jetbrains.com/pycharm/) Un editor profesional y gratuíto.
* [atom](https://atom.io/) Editor ligero de Github.
* [code](https://vscode.io) Editor ligero de Microsoft.
* [pydev](http://pydev.org)
* [sublime](http://www.sublimetext.com) Editor ligero.

## Test unitarios

Los test unitarios son programas que comprueban que las funciones están bien hechas. Básicamente son programas que ejecutan las funciones y comprueban que el resultado que tienen es el correcto. Existen diferentes librerías para hacer tests, aunque en Python existe unittest por defecto, por tanto no hace falta instalar nada.

Supongamos que tenemos la siguiente clase que representa una calculadora:

class Calculadora:  
 def sumar (self, a, b):  
 return a + b  
   
 def restar (self, a, b):  
 return a - b  
   
 def multiplicar (self, a, b):  
 return a \* b  
   
 def dividir (self, a, b):  
 return a / b

Para hacer los test unitarios de esa clase, bastaría con crear esta otra clase, la cual hereda de una clase de testeo de la librería unittest. También tenemos que importar la propia clase calculadora ya que la tenemos que poner a prueba.

Cada función de la clase hace una comprobación de cada función de calculadora. Se pueden hacer tantos tests como creas conveniente para probar que las funciones hacen lo que deben. Como puedes ver, cada test básicamente ejecuta una función y comprueba que el resultado es el esperado con assertEqual:

def test\_sumar(self):  
 c = calculadora.Calculadora()  
 self.assertEqual(c.sumar(40, 2), 42)

Fichero calculadora.test.py:

import unittest  
import calculadora  
  
class TestStringMethods(unittest.TestCase):  
 def test\_sumar(self):  
 c = calculadora.Calculadora()  
 self.assertEqual(c.sumar(40, 2), 42)  
  
 def test\_restar(self):  
 c = calculadora.Calculadora()  
 self.assertEqual(c.restar(40, 2), 38)  
  
 def test\_multiplicar(self):  
 c = calculadora.Calculadora()  
 self.assertEqual(c.multiplicar(40, 2), 80)  
  
 def test\_dividir(self):  
 c = calculadora.Calculadora()  
 self.assertEqual(c.dividir(40, 2), 20)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Ahora basta con ejecutar el fichero de tests y veremos lo siguiente:

python3 calculadora.test.py  
  
....  
----------------------------------------------------------------------  
Ran 4 tests in 0.000s  
  
OK

## Iniciando un proyecto Python localmente

Una forma recomendada de iniciar un proyecto python sería utilizar el paquete virtualenv, el cual instalaremos con el gestor de paquetes pip3. virtualenv crea una carpeta de proyecto que puede funcionar independientemente de nuestro sistema. Eso hace que el proyecto sea más flexible y más fácil de portar a otros ordenadores.

Los siguientes comandos deberás escribirlos en la consola del sistema.

pip3 install virtualenv

Defaulting to user installation because normal site-packages is not writeable  
Collecting virtualenv  
 Downloading virtualenv-20.0.31-py2.py3-none-any.whl (4.9 MB)  
 |████████████████████████████████| 4.9 MB 447 kB/s   
Collecting distlib<1,>=0.3.1  
...

Con virtualenv ya podemos crear un nuevo proyecto:

virtualenv proyecto

Lo cual creará una carpeta llamada proyecto. A continuación debemos activar el entorno virtual del proyecto ejecutando:

source proyecto/bin/activate

Ahora ya podemos añadir código fuente o instalar dependencias. Para ello conviene crear un fichero de texto llamado requirements.txt que debe tener el siguiente formato:

# Para instalar una versión concreta   
# nombre\_paquete==version  
  
# Para instalar una versión igual o superior  
# nombre\_paquete>=version  
  
# Para instalar la versión más reciente  
# nombre\_paquete

Por ejemplo, si queremos instalar pygame y un paquete de testing podríamos poner:

pygame==1.9.6  
unittest

Y luego podríamos instalar ese y otros paquetes que ahí indiquemos con el comando:

source bin/activate  
pip3 install -r requirements.txt

Otra opción es instalar los paquetes necesarios con pip3:

pip3 install pygame

Comprobamos que está instalado:

pip3 list

Package Version  
---------- -------  
pip 20.2.2  
pygame 1.9.6  
setuptools 49.6.0  
wheel 0.35.1

Y mediante freeze lo guardamos en el fichero requirements.txt:

pip3 freeze > requirements.txt

Ahora ya podemos crear un fichero que utilice pygame:

import pygame  
  
pygame.init()  
  
# Set up the drawing window  
screen = pygame.display.set\_mode([500, 500])  
  
# Run until the user asks to quit  
running = True  
while running:  
  
 # Did the user click the window close button?  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
  
 # Fill the background with white  
 screen.fill((255, 255, 255))  
  
 # Draw a solid blue circle in the center  
 pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75)  
  
 # Flip the display  
 pygame.display.flip()  
  
# Done! Time to quit.  
pygame.quit()

Ejecutaríamos el juego así:

python3 game.py

Y para terminar el entorno virtual de Python, bastaría con hacer:

deactivate

Epílogo

|  |  |
| --- | --- |
|  |  |

El libro termina aquí pero la Aventura de aprendera programar y de crear nuevos proyectos acaba de empezar. Hay infinidad de problemas para resolver y toda clase de herramientas para que puedas elegir la que más te guste.

¡Te animamos a que sigas descubriendo más!

