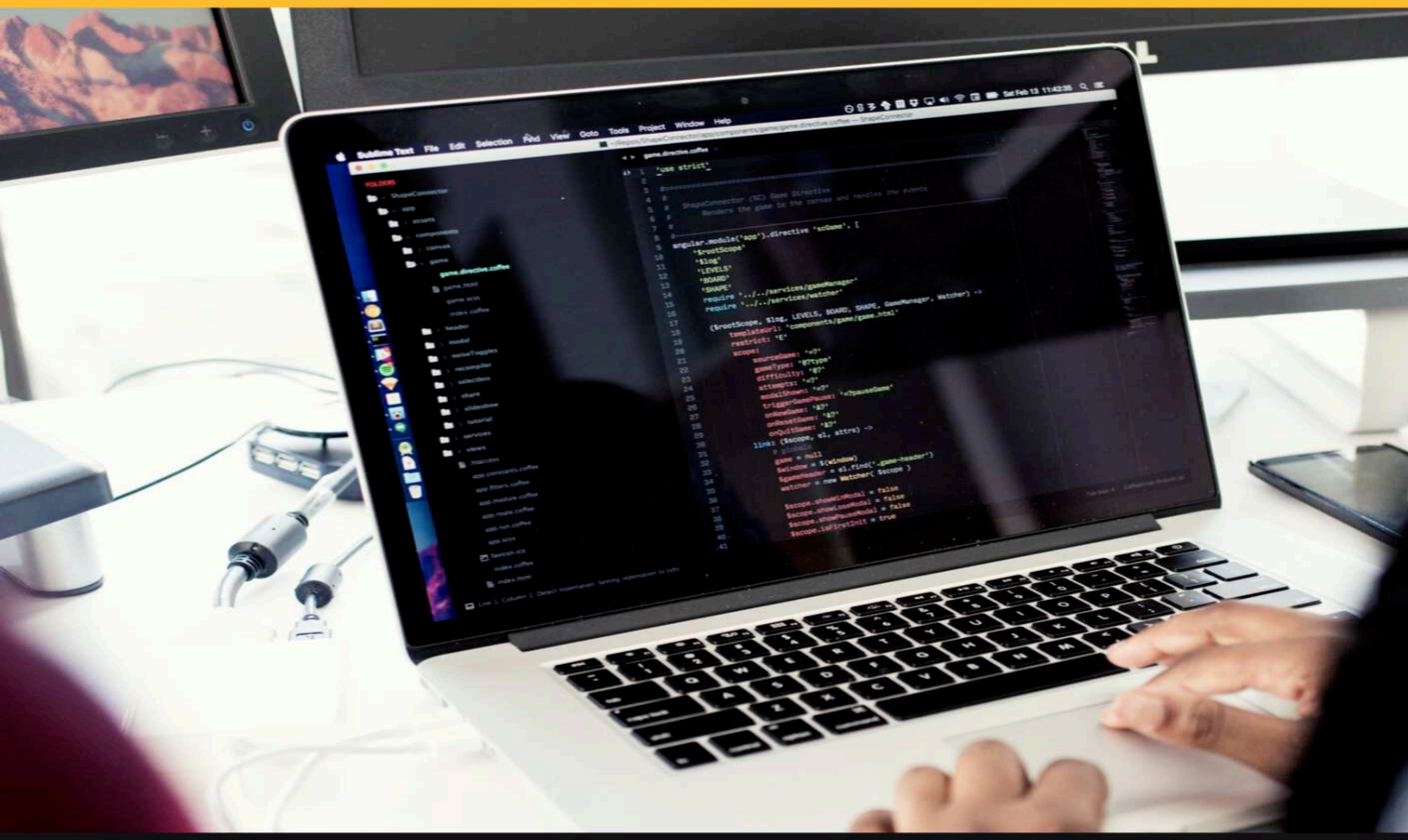
APRENDE PYTHON
DE CERO A EXPERTO

JON VADILLO



Aprende de forma práctica y sencilla uno de los lenguajes más populares de la actualidad.

"Lo único que necesitas saber es que tú puedes aprender cualquier cosa"
- Sal Kahn -

Aprende Python desde cero a expero

Jon Vadillo Romero

Este libro está a la venta en http://leanpub.com/aprende-python

Esta versión se publicó en 2019-10-07

1. Agradecimientos

Cada palabra de este libro no hubiese sido posible sin todas aquellas personas que me han acompañado durante este viaje y me han ayudado a tomar decisiones, rumbos y caminos que me han llevado a decidir escribir este libro y ser capaz de hacerlo. Compañeros, amigos, familia: MUCHAS GRACIAS A TODOS.

Este libro es una forma de devolver una pequeña parte de todo lo que he recibido y contribuir a que otras personas puedan beneficiarse y acceder de forma libre a una parte del conocimiento que he adquirido a lo largo de los años.

Índice general

2. Introducción

Python es sin duda alguna uno de los lenguajes de programación más populares que existen hoy en día. En comparación con otros lenguajes de programación, Python presume de tener una **curva de aprendizaje pequeña y un gran potencial**, ya que hablamos de un lenguaje con el que es posible realizar tareas de todo tipo, como por ejemplo:

- Desarrollo de aplicaciones web
- Inteligencia artificial
- Automatización de tareas
- Análisis avanzado de datos

En este libro aprenderás los fundamentos básicos del lenguaje de forma que al terminar su lectura cuentes con una sólida base sobre la que seguir tu carrera como programador.

Características principales

Se trata de un lenguaje **open source** en el que destaca su legibilidad. El código es limpio y ordenado, lo cual convierte a Python en un lenguaje cómodo de leer y escribir. En definitiva: **un lenguaje de programación fácil de entender y aprender**.

Al contrario que otros lenguajes de programación como C o Java, **Python es un lenguaje interpretado**, lo que significa que no es necesario compilar el código de Python antes de ejecutarlo. El intérprete irá analizando y ejecutando el código línea por línea.

Otra de las principales características de Python es que es que **es un lenguaje de programación dinámicamente tipado**, es decir, el programador no tiene que declarar el tipo de las variables si no que este se deduce automáticamente en el tiempo de ejecución.

```
1  # La variable "edad" guarda el valor como integer
2  edad = 25
3  print("La variable 'edad' es de tipo: " + str(type(edad)))
4  # Ahora "edad" guarda un string
5  edad = "Tengo 25 años"
6  print("La variable 'edad' es de tipo: " + str(type(edad)))
```

Por último, comentar que una de las mayores ventajas de este **lenguaje de programación orientado a objetos** es la extensa variedad de **liberías y frameworks** disponibles para cualquier propósito, lo cual hace que Python sea la opción perfecta para el desarrollo de aplicaciones de cualquier tipo.

Instalación de Python

Python es un **lenguaje multiplataforma**, lo que significa que podemos trabajar con Python tanto en Windows, Mac o Linux. A pesar de que todavía encontrarás código escrito en Python 2, en la actualidad **la versión recomendada es Python 3**. A continuación podrás ver cómo instalar Python en cada uno de los entornos.

Instalar Python en Windows

Sigue los siguientes pasos para instalar Python en Windows:

1. Descarga la última versión de Python para Windows desde la página web oficial: [https://www.python.org/downlo
2. En función de la versión de tu sistema operativo (32 btis o 64 bits), tendrás que escoger entre una de las dos versiones: **Windows x86 executable installer** o **Windows x86-64 executable installer**.
3. Una vez descargado, ejecuta el instalador marcando las casillas "Add Python 3.6 to PATH" y "Add Python to your environment variables".

Instalar Python en Ubuntu

En primer lugar comprueba que Python no esté instalado en el sistema mediante el siguiente comando:

```
$ python3 --version
Python 3.6.1
```

En caso de no estar instalado, basta con ejecutar el siguiente comando en la consola:

```
$ sudo apt install python3
```

Instalar Python en Mac

La instalación para Mac es sencilla y directa. Simplemente descarga la última versión de Python desde la página web oficial[1] y ejecuta el archivo .pkg descargado.

Entorno de desarrollo

Editores de código

Para programar en Python **es suficiente con tener un editor de texto cualquiera**, aunque hoy en día es recomendable utilizar un editor avanzado o un IDE que permita programar de forma ágil.

Estas son algunas de las opciones más recomendadas:

[1] https://www.python.org/downloads/mac-osx/

Editores de texto

- Atom: https://atom.io/[2]
- Sublime Text: https://www.sublimetext.com/3[3]
- Visual Studio Code: https://code.visualstudio.com/[4]

IDEs

- PyCharm: https://www.jetbrains.com/pycharm/[5]
- Eclipse (con el plug-in Pydev): https://www.eclipse.org/[6];[https://www.pydev.org/] (https://www.pydev.org/)

Entorno de desarrollo avanzado

Más adelante, una vez conozcas los fundamentos del lenguaje y tengas fluidez a la hora de programar, te recomendamos profundizar en los conocidos como "entornos Virtuales". Los entornos virtuales permiten configurar tu entorno de desarrollo de forma más avanzada, creando ambientes (entornos) aislados que permitan trabajar con distintas versiones de frameworks y librerías. Mientras tanto, te recomiendo que continúes paso a paso y en orden por el resto de lecciones.

Tu primer programa: ¡Hola Mundo!

Ya tenemos Python instalado en nuestro sistema operativo, por lo que estamos preparados para comenzar a escribir y ejecutar código Python. Veremos cómo ejecutar código Python **desde la consola directamente o desde un archivo** de extensión `.py`

Consola de Python

La consola de Python nos **permite ejecutar código escrito en Python directamente**, sin tener que escribir el código previamente en ningún fichero. Para entrar en la consola de Python, abre una consola (también conocidda como terminal o shell) del sistema operativo en el que te encuentres y escribe `py` o `python`. Esto nos mostrará la versión de Python instalada y dará comienzo a la consola de Python:

[2] https://atom.io/
[3] https://www.sublimetext.com/3
[4] https://code.visualstudio.com/
[5] https://www.jetbrains.com/pycharm/
[6] https://www.eclipse.org/

```
    python
    Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Inte\
l)] on win32
    Type "help", "copyright", "credits" or "license" for more information.
    >>>
```

Verás cómo el cursor cambia y ahora aparece el símbolo `>>>`. A partir de ahora ya podemos ir ejecutando las instrucciones que queramos. En este caso escribiremos nuestro primer programa, el ya por todos conocido ¡Hola Mundo!:

```
    >>> print("¡Hola Mundo!")
    ¡Hola Mundo!
```

También puedes ejecutar otras instrucciones. Prueba a ejecutar la siguiente operación:

```
    >>> 3*3
    9
```

Para salir de la consola de Python, escribe `exit()` o pulsa `CTRL + Z` y `ENTER`.

Ejecutar fichero

También podemos escribir nuestro código en un **fichero con extensión `.py`** y ejecutarlo. Abre un editor de texto y crea un fichero llamado `holamundo.py` con el siguiente contenido:

```
    print("¡Hola Mundo!")
```

Para ejecutar el archivo, abre una consola de comandos en la misma ubicación donde has guardado el archivo y escribe `python`seguido del nombre del fichero:

python holamundo.py

¡Enhorabuena! Ya has escrito y ejecutado tu primer programa en Python.

Sintaxis

Indentación

Un aspecto muy importante a mencionar antes de comenzar a programar en Python es el hecho de que **Python utiliza la indentación** (también conocida como sangría, tabulación o espaciado) para **delimitar los bloques de código**. La indentación estándar de Python requiere una tabulación de 4 espacios:

```
x = 5
if x == 5:
    # tabulación de 4 espacios
    print("El valor de x es 5.")
```

En caso contrario, al ejecutar nuestro código recibiremos un error como el siguiente

```
    IndentationError: unindent does not match any outer indentation level
```

Comentarios

A la hora de programar es posible que queramos introducir en el código comentarios que añadan información extra sin afectar a la ejecución del programa. En Python los comentarios se insertan mediante el carácter hash (#):

```
    # Mi primer comentario
    x = 5 # Un comentario junto con el código
```

Funcionamiento de Python

Python ejecuta nuestro código línea por línea. Por cada línea de código estas son las acciones que se realizan:

1. **Analizar (*parse* en inglés) el código** comprobando que formato y la sintaxis es correcta, es decir, que cumplen las normas establecidas para el lenguaje de progamación.
2. **Traducir el código a bytecode** (instrucciones que nuestra máquina puede ejecutar).
3. **Enviar el código para su ejecución** a la Python Virtual Machine(PVM), donde el código es ejecutado.

3. Variables y Tipos de datos

Las variables permiten **almacenar datos del programa**. Estas serán de un tipo u otro en función de la información que se guarde en ellas.

```
1 nombre = 'Amaia' # cadena de texto
2 edad = 30 # número entero
```

El nombre de una variable se conoce como **identificador**, y deberá cumplir las siguientes reglas:

- Comenzar con una letra o un guión bajo.
- El resto del nombre estará formado por letras, números o guiones bajos.
- Los nombres de las variables son *case sensitive*, es decir, no es lo mismo que una variable se llame `resultado` que `RESULTADO`.
- Existen una serie de palabras reservadas que no se pueden utilizar (def, global, return, if, for, ...).

Algunas de las recomendaciones respecto a los nombres de las variables están recogidas en la Guía oficial de Estilos PEP8 de Python[1]. Entre las más habituales encontramos las siguientes:

- Utilizar nombres descriptivos, en minúsculas y separados por guiones bajos si fuese necesario: `resultado`, `mi_variable`, `valor_anterior`,...
- Escribir las constantes en mayúsculas: `MI_CONSTANTE`, `NUMERO_PI`, ...
- Antes y después del signo `=`, debe haber uno (y solo un) espacio en blanco.

Nota: No olvides que lo que la guía plantea son recomendaciones y no obligaciones. Por ejemplo, mientras PEP8 recomienda tabular el código con 4 espacioes en blanco, la guía particular de los desarrolladores de Google habla de 2 espacios en lugar de 4.

Resumen de tipos de variables

[1] https://www.python.org/dev/peps/pep-0008/

```
    edad = 24 # número entero (integer)
    precio = 112.9 # número de punto flotante (float)
    titulo = 'Aprende Python desde cero' # cadena de texto (string)
    test = True # booleano
```

Lectura de datos en Python

La función `input()` permite introducir datos al usuario:

```
>>> nombre = input()
Leire
>>> print(nombre)
Leire
```

Como se puede ver en el siguiente ejemplo, es posible también mostrar un mensaje al usuario, tal y como muestra el siguiente ejemplo.

```
>>> nombre = input("Escribe tu nombre: ")
Escribe tu nombre: Leire
>>> print(nombre)
Leire
```

Números

Python soporta dos tipos de números: enteros (integer) y de punto flotante (float).

```
# integer
x = 5
print(x)

# float
y = 5.0
print(y)

# Otra forma de declarar un float
z = float(5)
print(z)
```

Si tenemos dudas del valor de una variable, podemos mostrar su tipo utilizando la función `type()`:

```
>>> x = 5.5
>>> type(x)
<class 'float'>
```

Cadenas de texto (string)

Las cadenas de texto o strings se definen mediante comilla simple (' ') o doble comilla (" "):

```
mi_nombre = 'Ane'
print(mi_nombre)
mi_nombre= "Ane"
print(mi_nombre)
```

La diferencia principal se encuentra en que las comillas dobles aportan mayor facilidad en textos que incluyan apóstrofes:

```
mi_nombre = 'I\'m John'
print(mi_nombre)
mi_nombre= "I'm John"
print(mi_nombre)
```

Más información sobre strings y carácteres especiales en: https://docs.python.org/3/tutorial/introduction.html#string

Para definir strings multi-línea se utiliza la triples comillas (""""):

```
frase = """ esto es una
        frase muy larga de más de
        una línea ..."""
```

Concatenación de strings

Es posible unir dos strings con el operador +:

[2] https://docs.python.org/3/tutorial/introduction.html#strings

```
>>> primera_palabra = 'Hola'
>>> frase_completa = primera_palabra + ', mundo'
>>> print(frase_completa)
'Hola, mundo'

>>> segunda_palabra = 'mundo'
>>> frase_completa = primera_palabra + ', ' + segunda_palabra
>>> print(frase_completa)
'Hola, mundo'
```

Método alternativo 1: str.join():
El método `join()` recibe como argumento el listado (de tipo List, Tuple, String, Dictionary y Set) de strings que se desean concatenar. Se invoca sobre el separador que se utilizará para unir las cadenas (el cual a su vez es un string también):

```
>>> strings = ['do', 're', 'mi']
>>> separador = ' - '
>>> separador.join(strings)
'do - re - mi'
```

Para iterar un elemento detrás del otro se introducirá string vacío como separador:

```
>>> strings = ['do', 're', 'mi']
>>> ''.join(strings)
'doremi'
```

Método alternativo 2: `str.format()`:
Python 3 introdujo una nueva forma para formatear strings, la cual sustituye a la anterior en la que se hace uso del operador `%`. Para ello se invoca el método `format()` de un string:

```
# Ordenado por defecto:
frase = "Meses: {}, {} y {}".format('Enero','Febrero','Marzo')
print(frase)

# Especificar el orden indicando la posición:
frase = "Meses: {1}, {0} y {2}".format('Enero','Febrero','Marzo')
print(frase)

# Especificar el orden mediante parejas clave-valor:
orden_palabra = "Meses: {ene}, {feb} y {mar}".format(ene='Enero', feb='Febrero',m='M\
arzo')
print(frase)
```

Conversión de tipos

A la hora de concatenar un string con otras variables como `integer` o `float` puede haber problemas:

```
>>> edad = 25
>>> nota_media = 7.3
>>> print("Tengo " + edad + " años y mi nota media es " + nota_media + ".")

Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Mediante la función `str()` podemos convertir un valor a string y evitar así cualquier tipo de problema:

```
>>> edad = 25
>>> nota_media = 7.3
>>> print("Tengo " + str(edad) + " años y mi nota media es " + str(nota_media) + ".")
Tengo 25 años y mi nota media es 7.3.
```

De igual manera es posible convertir a otros tipos con las funciones `int()`, `float()` and `bool()`.

Métodos en cadenas de texto (string)

Es posible **obtener un carácter concreto** de un string utilizando los corchetes `[]` y el índice del carácter al que queremos acceder:

```
frase = 'Aprendiendo a programar en Python'
course[0] # devuelve el primer caracter
course[1] # devuelve el segundo caracter
course[-1] # devuelve el primer caracter empezando por el final
course[-2] # # devuelve el segundo caracter empezando por el final
```

Si queremos **obtener un substring**, utilizaremos la siguiente notación:

```
frase = 'Aprendiendo a programar en Python'
mi_substring = frase[1:5] # devuelve los caracteres desde la posición 1 hasta la 5 (\
no incluye el 5)
```

En caso de dejar la primera variable vacía, se considera la primera posición del string. Dejando la segunda variable vacía se considera la última posición del string:

```
>>> frase = 'Aprendiendo a programar en Python'
>>> mi_substring = frase[:5]
>>> mi_substring
'Apren'
>>> mi_substring = frase[4:]
>>> mi_substring
'ndiendo a programar en Python'
```

Otros métodos útiles de string:

```
str.upper() # convierte a mayúsculas
str.lower() # convierte a minúsculas
str.title() # convierte a mayúsculas la primera letra de cada palabra
str.count(substring [, inicio, fin]) # devuelve el número de veces que aparece el su\
bstring en el string. Opcionalmente se puede indicar el inicio y fin.
str.find('d') # devuelve el índice de la primera aparición de 'd' (o -1 si no lo enc\
uentra)
substr in str # devuelve True si el string contiene el substring
str.replace(old, new [, count])   # reemplaza 'old' por 'new'un máximo de 'count' ve\
ces (opcional).
str.isnumeric()   # devuelve True si str contiene solamente números
```

Coding time!

Ejercicio 1

Escribe un programa que contenga las siguientes variables:

- nombre: tipo string y valor “Kobe Bryant”
- edad: tipo integer y valor 45
- media_puntos: tipo float y valor 28.5
- activo: False

El programa deberá mostrar en pantalla todos los valores.

Ejercicio 2

Escribe un programa que solicite el nombre, DNI y edad, lo almacene en 3 variables distintas y muestre por pantalla los valores introducidos.

Ejercicio 3

Escribe un programa que genere un string compuesto por los primeros 3 caracteres y los último 3 caracteres de un string introducido por el usuario.

- Ejemplo 1: ‘aprendiendo’
- Resultado 1: ‘aprndo’
- Ejemplo 2: ‘escribiendo código’
- Resultado 2: ‘escigo’

Ejercicio 4

Escribe un programa que solicite al usuario dos números y una frase. El primer número introducido se corresponderá a la posición de inicio del substring que deberá mostrar el programa por pantalla. El segundo número indicará la longitud de dicho substring.

- Ejemplo 1: 4, 8, ‘Desarrollar es mi nueva afición’
- Resultado 1: ‘‘rollar es”
- Ejemplo 2: ‘11, 8, Bienvenido a la clase de programación’
- Resultado 2: ‘a la cla’

Ejercicio 5

Escribe un programa que solicite al usuario una frase. A continuación le solicitará la letra que quiere reemplazar y por qué letra deberá reemplazarse. Por último el programa mostrará el número de veces que la letra está presente en la frase y el resultado final tras reemplazarla.

- Ejemplo: ‘Desarrollar es mi nuevo pasatiempos’, ‘a’,’e’
- Resultado: 4 apariciones. ‘Deserroller es mi nueve pesetiempos’

4. Operadores y expresiones

Los operadores son símbolos especiales que permiten realizar operaciones aritméticas o lógicas.

Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas (suma, resta, multiplicación,...). La tabla siguiente contiene todos los operadores aritméticos permitidos por Python:

Operador	Ejemplo	Significado
+	a + b	Suma
-	a - b	Resta
-	-a	Negación (asignar valor negativo)
*	a * b	Multiplicación
/	a / b	División
%	a % b	Módulo (resto de la división)
//	a // b	División entera
**	a ** b	Exponente

Ejemplos:

```
x = 5
y = 2
print(x + y) # 7
print(x - y) # 3
print(x * y) # 10
print(x / y) # 2.5
```

Operadores relacionales o de comparación

Los operadores relacionales se utilizan para comparar valores y devuelven como resultado un booleano: `True` o `False`.

Operador	Ejemplo	Significado
>	a > b	Mayor que: `True` si a es mayor que b
<	a > b	Menor que: `True` si a es menor que b
==	a == b	Igual: `True` si a y b son iguales
!=	a != b	Distinto: `True` si a y b son distintos
>=	a >= b	Mayor o igual: `True` si a es igual o mayor que b
<=	a >= b	Menor o igual: `True` si a es igual o menor que b

Operadores lógicos

Los operadores lógicos `and` `or`, y `not` evalúan valores devolviendo también `True`o `False` como resultado:

Operador	Ejemplo	Significado
and	a and b	True si a y b son True
or	a or b	True si a o b son true
not	not b	True si b es falso

Coding time!

Ejercicio 1

Crea un programa que solicite al usuario un número y muestre por pantalla la tabla de multiplicación del 1 al 10.

Ejercicio 2

Crea un programa que solicite al usuario dos números y muestre por pantalla el resultado de las siguientes operaciones: suma, resta, multiplicación y división.

Ejercicio 3

Crea un programa que solicite al usuario el radio de un círculo y calcule el área.

5. Estructuras de control

Condicionales

Las estructuras de control se utilizan para **ejecutar bloques de código en función de condiciones**.

Sentencia IF - ELSE

Se evalúa la condición especificada en la sentencia `if` y en caso de cumplirse se ejecutará el bloque de código indentado. En caso de que el resultado de la condición sea `False`, no el bloque especificado no se ejecutará:

```
numero = 5
if numero > 1:
    print("Es mayor que uno")
```

Es posible especificar un bloque de código que se ejecute en caso de que la condición no se cumpla:

```
numero = 2
if numero > 10:
    print("Es mayor que diez")
else:
    print("Es menor o igual que diez")
```

También podemos comprobar más condiciones mediante la expresión `elif`. En este caso, se seguirán comprobando todas las condiciones `elif` hasta que una de ellas se cumpla. En caso contrario, se ejecutará el bloque de código dentro de `else` (si lo hubiera).

```
numero = 5
if numero < 3:
    print("Es menor que 3")
elif numero < 6:
    print("El número está entre el 3 y el 5")
else:
    print("Es mayor o igual a 6")
```

Python no tiene limitaciones para anidar distintos bloques de `IF`s.

```
numero = 2
if numero >= 0:
    if numero == 0:
        print("El valor es 0")
    else:
        print("Es un número positivo")
else:
    print("Es un número negativo")
```

Bucles

Los bucles permiten ejecutar un bloque de código tantas veces como queramos.

Sentencia WHILE

La sentencia `while` permite ejecutar un bloque de código mientras la expresión que definamos se cumpla (es decir, devuelva `True`). Python interpretará como `True` cualquier valor distinto a `0` o `None`.

```
contador = 0
while(contador < 5):
    contador = contador+1
    print("Iteración número {}".format(contador))
print ("¡Fin!")
```

Para detener una ejecución se utiliza la sentencia `break`:

```
contador = 0
while(contador < 5):
    contador = contador+1
    print("Iteración número {}".format(contador))
    if contador == 3:
        break
print ("¡Fin!")
```

También es posible saltar únicamente la iteración actual mediante la sentencia `continue`:

```
contador = 0
while(contador < 5):
    contador = contador+1
    if contador == 3:
        continue
    print("Iteración número {}".format(contador))
print ("¡Fin!")
```

La salida del programa anterior será la siguiente:

Iteración número 1
Iteración número 2
Iteración número 4
Iteración número 5
Bucle while finalizado

Otros lenguajes de programación ofrecen otra estructura similar conocida como DO - WHILE. No es el caso de Python, por lo que habría que emular dicho comportamiento mediante nuestro código.

Bucle WHILE con ELSE

Para ejecutar un código al finalizar, es decir, cuando la condición es `False` y no se ha finalizado con `break`, se utiliza la expresión `else`:

```
count = 0
while(count < 5):
    count = count+1
    print("Iteración número {}".format(count))
else:
    print("Bucle while finalizado")
```

Sentencia FOR

A diferencia de otros lenguajes de programación, en Python la sentencia FOR itera únicamente por secuencias (listas, tuplas, cadenas de caracteres,...).

```
alumnos = ["Ane", "Mikel", "Unai", "Lorea"]
for alumno in alumnos:
    print(alumno)
```

También es posible utilizarlo para recorrer un string:

```
frase = "Aprendiendo Python"
for c in frase:
    print(c)
```

Para detener una ejecución se utiliza la sentencia `break`:

```
numeros = [4,8,2,7,1,9,3,5]
total = 0

for n in numeros:
    total += n
    if total > 10
        break
```

También es posible saltar únicamente la iteración actual mediante la sentencia `continue`:

```
numeros = [4,8,2,7,1,9,3,5]
total = 0

# solo sumar los números impares
for num in numeros:
    if num % 2 == 0:
        print("Numero par, no lo sumamos")
        continue
    total += n
```

Bucle FOR con ELSE

Python permite definir un bloque de código a ejecutar una vez la iteración por los elementos de una lista ha finalizado. Es importante mencionar que no se ejecutará si se ha finalizado mediante `break`.

```
alumnos = ["Ane", "Mikel", "Unai", "Lorea"]
for alumno in alumnos:
    print(alumno)
else:
    print("No quedan más alumnos.")
```

La función range()

La función `range([start,] stop [, step])` devuelve una secuencia de números. Es por ello que se utiliza de forma frecuente para iterar:

```python
for i in range(3):
    print(i)
# 0
# 1
# 2
```

También podemos indicar el inicio, fin y step:

```python
print("Números del 5 al 10")
for i in range(5,  10):
    print(i,  end=', ')
# 5,  6,  7,  8,  9,

print("Números impares del 1 al 10")
for i in range(1,  10,  2):
    print(i,  end=', ')
# 1,  3,  5,  7,  9,
```

Para iterar por una lista utilizando el índice, basta con combinarlo con la función `len()`:

```python
    alumnos = ["Ane", "Mikel", "Unai", "Lorea"]
    for i in range(len(alumnos)):
            print(alumnos[i])
```

Coding time!

Ejercicio 1

Crea un programa que solicite un número al usuario y devuelva el siguiente mensaje:

- Si es mayor que 0: "Es un número positivo."
- Si es igual a 0: "Es igual a cero.
- Si es menor que 0: "Es un número negativo."

Ejercicio 2

Crea un programa que solicite dos números al usuario y muestre por pantalla la suma de todos los números que hay entre los dos números (ambos incluidos).

- Ejemplo: 4, 8
- Resultado: 30

Ejercicio 3

Mejora el programa anterior para que muestre por separado la suma de los números pares y los impares.

- Ejemplo: 4, 8
- Resultado: Pares = 18, Impares = 12, Total = 30

Ejercicio 4

Escribe un programa que solicite al usuario un nombre de usuario y contraseña. El programa mostrará el mensaje "¡Bienvenido!" si el usuario introduce los siguientes datos:

- Nombre de usuario: root
- Contraseña: toor

Si los datos de acceso son incorrectos mostrará el mensaje "Acceso fallido" y el programa finalizará.

Ejercicio 5

Mejora el programa anterior para que permita 3 intentos. Cada vez vez que el usuario introduzca datos de acceso incorrectos el programa mostrará el mensaje: "Datos incorrectos. Le quedan X intentos.", siendo X el número de intentos restantes. Tras el tercer fallo el programa mostrará el mensaje "Acceso fallido" y finalizará.

Ejercicio 6

Crea un programa que reciba 5 números del usuario y muestre el mayor de todos por pantalla.

Ejercicio 7

Mejora el programa anterior, de forma que el usuario pueda introducir tantos números como quiera. El programa solicitará números al usuario hasta que introduzca la palabra "fin". Entonces mostrará el mayor de todos por pantalla.

6. Listas y tuplas

Las listas permiten **guardar más de un elemento** dentro de una variable en un orden concreto. Pueden contener un número **ilimitado** de variables de **cualquier tipo**:

```
alumnos = ["Ane", "Unai", "Itziar", "Mikel"]
print(alumnos [0]) # muestra "Ane"
print(alumnos [1]) # muestra "Unai"
print(alumnos [2]) # muestra "Itziar"
print(alumnos [-1]) # muestra "Mikel"
```

Los **métodos más utilizados** con las listas son los siguientes:

Método	Acción
`alumnos.append("Amaia")`	Inserta "Jon" al final de la lista
`alumnos.insert(0,"Amaia")`	Inserta "Amaia" en la posición 0
`alumnos.remove("Amaia")`	Elimina la primera aparición de "Amaia" de la lista
`alumnos.pop()`	Elimina el último elemento de la lista
`alumnos.clear()`	Elimina todos los elementos de la lista
`alumnos.index("Amaia")`	Devuelve el índice de la primera aparición de "Amaia"
`alumnos.sort()`	Ordena la lista (los elementos deben ser comparables)
`sorted(alumnos)`	Devuelve una copia de la lista 'alumnos' ordenada (no ordena la pasada como parámetro)
`alumnos.reverse()`	Ordena la lista en orden inverso
`alumnos.copy()`	Devuelve una copia de la lista
`alumnos.extend(otra_lista)`	Fusiona las dos listas

Recorrer una lista

La forma habitual de recorrer una lista es mediante la sentencia `for`, tal y como muestra el ejemplo a continuación:

```
for elemento in ['Python','JavaScript','JAVA']:
    print("Programo en ", elemento)
```

De igual manera se podría hacer mediante la sentencia `while`:

```
    lista =  ['Python','JavaScript','JAVA']
    i = 0
    sizeofList = len(lista)
    while i < sizeofList :
        print(lista[i])
        i += 1
```

Tuplas

Las tuplas son **listas inmutables**. Es decir, una vez declaradas, no se pueden realizar modificaciones sobre ellas (añadir/eliminar elementos o hacer modificaciones sobre ellos). Para definir una tupla se escriben los elementos entre paréntesis:

```
    valores = (1,2,3)
```

El acceso a sus elementos se hace de igual que con listas:

```
    valores = ("a", "b", "c","d","e","f")
    print(valores[1])
    print(valores[2:4])
```

Una acción típica de las tuplas es "desempaquetar" (unpack) sus valores:

```
    a, b, c = valores
```

Coding time!

Ejercicio 1

Dada la siguiente lista `[12, 23, 5, 29, 92, 64]` realiza las siguientes operaciones y vete mostrando la lista resultante por pantalla:

1. Elimina el último número y añádelo al principio.
2. Mueve el segundo elemento a la última posición.
3. Añade el número `14` al comienzo de la lista.
4. Suma todos los números de la lista y añade el resultado al final de la lista.
5. Comina la lista actual con la siguiente: `[4, 11, 32]`
6. Elimina todos los números impares de la lista.
7. Ordena los números de la lista de forma ascendente.
8. Vacía la lista.

Ejercicio 2

Crea un programa que solicite al usuario `5` números y los guarde en una lista. A continuación el programa pedirá otros `5` números al usuario almacenándolos en una segunda lista. El programa mostrará al usuario una lista que contenga los números que tienen en común las dos listas anteriores.

- Ejemplo: Lista `1 = [6,14,11,78,5]` y Lista `2 = [3,14,22,78,9]`
- Resultado: `[14, 78]`

7. Diccionarios

Un diccionario es un conjunto de parejas **clave- valor** (key-value). Es decir, se accede a cada elemento a partir de su clave. Se definen de la siguiente manera:

```
estudiante = {
        "nombre": "Iñaki Perurena",
        "edad": 30,
        "nota_media": 7.25,
        "repetidor" : False
}
```

Las **claves tienen que ser únicas** y estar formadas por un **string o un número**. Para acceder al valor de una clave exiten dos maneras distintas:

```
edad = estudiante["edad"] # devuelve el valor de 'edad'
nota_media = estudiante.get("nota_media") # devuelve el valor de 'nota_media'

estudiante["edad"] = 25 # actualiza el valor de 'edad'
estudiante["suspensos"] = 3 # inserta una nueva pareja clave - valor
estudiante.update({'aprobados':'8'}): inserta una nueva pareja clave - valor o actua\
liza su valor si ya existiera
```

Algunos de los métodos más utilizados son los siguientes:

Método	Acción
`diccionario.keys()`	Devuelve todas las claves del diccionario
`diccionario.values()`	Devuelve todos los valores del diccionario
`diccionario.pop(clave[,<default>])`	Elimina la clave del diccionario y devuelve su valor asociado. Si no la encuentra y se indica un valor por defecto, devuelve el valor por defecto indicado.
`diccionario.clear()`	Vacía el diccionario
`diccionario.clear()`	Vacía el diccionario
`clave in diccionario`	Devuelve True si el diccionario contiene la clave o False en caso contrario.
`valor in diccionario.values()`	Devuelve True si el diccionario contiene el valor o False en caso contrario.

Recorrer un diccionario

La forma más habitual de recorrer un diccionario es mediante la sentencia `for`. Al recorrer un diccionario, por defecto se iterará sobre sus claves:

```
diccionario = {'a':1, 'b':2, 'c':3}
for key in diccionario:
        print(key)

# Resultado: a b c
```

Es decir, el código anteiror será equivalente al siguiente:

```
diccionario = {'a':1, 'b':2, 'c':3}
for key in diccionario.keys():
        print key

# Resultado: a b c
```

Por lo tanto, para iterar accediendo a los valores, realizaremos lo siguiente:

```
diccionario = {'a':1, 'b':2, 'c':3}
for key in diccionario:
        print(diccionario[key])

# Resultado: 1 2 3
```

Otro manera alternativa sería empleando la función `items()`, la cual devuelve el diccionario como tuplas de tipo (key,value):

```
diccionario = {'a':1, 'b':2, 'c':3}
for key, value in diccionario.items():
        print("El valor de %s is %d" % (key, value))

# Resultado:
# El valor de a is 1
# El valor de b is 2
# El valor de c is 3
```

Borrar un elemento

Para borrar un elemento de un diccionario se utiliza la instrucción `del`.

```
edades = {
   "Ane" : 22,
   "Jokin" : 27,
   "Aitor" : 15
}
del edades ["Aitor"]
```

Otra alternativa también utilizada y mencionada anteriormente es la función `pop()`, el cual devuelve el valor del elemento eliminado:

```
edades = {
   "Ane" : 22,
   "Jokin" : 27,
   "Aitor" : 15
}
edades.pop("Aitor")
```

Coding time!

Ejercicio 1

Crea un programa que recorra una lista y cree un diccionario que contenga el número de veces que aparece cada número en la lista.

- Ejemplo: `[12, 23, 5, 12, 92, 5,12, 5, 29, 92, 64,23]`
- Resultado: `{12: 3, 23: 2, 5: 3, 92: 2, 29: 1, 64: 1}`

Ejercicio 2

Recorre un diccionario y crea una lista solo con los valores que contiene, sin añadir valores duplicados.

- Ejemplo: `{'Mikel': 3, 'Ane': 8, 'Amaia': 12, 'Unai': 5, 'Jon': 8, 'Ainhoa': 7, 'Maite': 5}`
- Resultado: `[3, 8, 12, 5, 7]`

Ejercicio 3

Crea una programa de Login que compruebe el usuario y contraseña en el diccionario a continuación:

```
usuarios = {
      "iperurena": {
          "nombre": "Iñaki",
                  "apellido": "Perurena",
                  "password": "123123"
          },
          "fmuguruza": {
               "nombre": "Fermín",
                  "apellido": "Muguruza",
                  "password": "654321"
          },
          "aolaizola": {
               "nombre": "Aimar",
                  "apellido": "Olaizola",
                  "password": "123456"
          }
    }
```

El usuario tendrá un máximo de 3 intentos, y al acceder correctamente se mostrará el nombre y apellido del usuario.

Ejercicio 4

Crea un programa que permita introducir a un profesor las notas de sus estudiantes (máximo 10 estudiantes). Los datos se deberán almacenar en un diccionario como el siguiente:

```
estudiantes = {
   "1": {
          "nombre": "Lorea",
          "nota": 8
  },
  "2": {
      "nombre": "Markel",
          "nota": "4.2"
  },
  "3": {
      "nombre": "Julen",
          "nota": 6.5
  }
}
```

Una vez introducidos todos los datos, el programa mostrará una lista con los nombres de los estudiantes que han suspendido y otra con los que han aprobado. También calculará y mostrará la nota media de la clase.

8. Funciones

Una función es un grupo de sentencias que realizan una tarea concreta. Esta forma de agrupar código es una forma de **ordenar** nuestra aplicación en pequeños bloques, **facilitando así su lectura** y permitiendo **reutilizar** el código que contienen sin esfuerzo.

Definir y llamar a una función

La sintaxis de una función en Python es la siguiente:

```
1  def saludo(nombre):
2  # codigo de la función
3      print("Hola, " + nombre+ ". ¡Bienvenido!")
```

Se escribe la palabra reservada `def` seguida del nombre de la función y sus parámetros entre paréntesis.

Para **llamar a una función** solo hay que escribir el nombre de la función seguida de los parámetros (si los hubiera) entre paréntesis.

```
1  >>> saludo('Maitane')
2  Hola, Maitane. ¡Bienvenida!
```

Es posible asignar al parámetro un **valor por defecto**.

```
1  def saludo(nombre = "Anónimo"):
2      print("Hola, " + nombre+ ". ¡Bienvenido!")
3
4  saludo("Leire") # Hola, Maitane. ¡Bienvenida!
5  saludo() # Hola, Anónimo. ¡Bienvenida!
```

Existen dos tipos de parámetros o argumentos:

- **Parámetros posicionales**: la posición en la que se pasan importa
- **Parámetros con palabra clave (keyword arguments)**: la posición no importa, se indica una clave para cada parámetro.

```
def suma(a, b):
        resultado = a + b
        print(resultado)
suma(45, 20) # parámentros posicionales
suma(a = 45, b =20) # parametros mediante clave
```

Las funciones pueden **devolver un valor** utilizando la palabra `return`. Una vez devuelto un valor, la función finaliza su ejecución.

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

print(suma(4,5)) # 9
```

Funciones con argumentos múltiples

Es posible recibir un **número desconocido** de parámetros añadiendo un * en la definición de la función.

```
def suma_todo(*args):
    resultado = 0
    for i in args:
        resultado += i
        return resultado
v, w, x, y, z = 5, 2, 12, 6, 9
total = suma_todo(v, w, x, y, z)
print("La suma total es:" + str(total))  # La suma total es: 34
```

Ámbito de las variables (scope)

El ámbito de una variable (*scope*) se refiere a la zona del programa dónde una variable "existe". Fuera del ámbito de una variable no podremos acceder a su valor ni manejarla.

Los parámetros y variables definidos en una función no estarán accesibles fuera de la función. A este ámbito se le conoce como **ámbito local**. Es importante mencionar que una vez ejecutada una función, el valor de las variables locales no se almacena, por lo que la próxima vez que se llame a la función, ésta no recordará ningún valor de llamadas anteriores.

```python
def calcula():
    a = 1
    print("Dentro de la función:", a)

a = 5
calcula()
print("Fuera de la función:", a)

### Output ###
# Dentro de la función:1
# Fuera de la función:5
```

Por el contrario, las variables definidas fuera de una función sí que están accesibles desde dentro de la función. Se considera que están en el **ámbito global**. No obstante, no se podrán modifican dentro de la función a no ser que estén definidas con la palabra clave `global`.

Coding time!

Ejercicio 1

Crea un programa que determine si un número es primo o no. Deberás crear la función `esPrimo()` que reciba como parámetro un número y devuelva `True` o `False` indicando si el número es primo o no.

Ejercicio 2

Crea un programa que genere un número aleatorio del 1 al 10. El usuario tendrá que adivinarlo, y el programa tras cada intento le indicará al usuario si el número es más alto, bajo o si ha acertado. La lógica para dar la respuesta al usuario deberá estar incluida en una función a la que se llamará tras cada intento.

Ejercicio 3

Crea un programa que reciba un número del 1 al 20 introducido por el usuario y compruebe si está dentro de la siguiente lista: `[6,14,11,3,2,1,15,19]`. Implementa una función que se asegure que el número introducido por el usuario está en el rango indicado y otra que compruebe si está dentro de la lista. Trata de crear las funciones de forma que puedan ser reutilizadas lo máximo posible en otros programas.

9. Excepciones

Las excepciones son **errores en la ejecución de un programa** que hacen que el programa termine de forma inesperada. Normalmente ocurren debido a un uso indebido de los datos (p.ej. una división entre cero). La manera de controlar las excepciones es agrupando el código en 2 bloques (más 1 opcional):

- `try`: agrupa el bloque de código en el que se pueda dar una excepción.
- `catch`: contiene el código a ejecutar en caso de que la excepción haya sido lanzada.
- `finally` (opcional): permite ejecutar un bloque de código siempre, se haya capturado o no una excepción.

```
try:
        numero = int(input('Introduce un número: '))
        dividendo = 150
        resultado = dividendo / numero
        print(resultado)
except ValueError:
        print('Número invalido')
except ZeroDivisionError:
        print('No se puede dividir entre 0')
finally:
        print("Ejecutando finally antes de salir")
```

También es posible lanzar excepciones de forma controlada mediante la sentencia raise.

```
    raise NameError('¡Soy una excepción!')
```

Excepciones comunes

Hay algunas excepciones que son bastante comunes a la hora de programar en Python y que deberíamos contemplar en nuestros programas:

TypeError es lanzado cuando se intenta realizar una operación o una función sobre un objeto de un tipo inapropiado.

```
>>> '1'+1
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
'2'+2
TypeError: must be str, not int
```

ValueError es lanzado cuando el argumento de una función es de un tipo inapropiado.

```
>>> int('hola')
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
int('xyz')
ValueError: invalid literal for int() with base 10: 'hola'
```

NameError es lanzado cuando no utiliza un objeto que no existe.

```
>>> persona
Traceback (most recent call last):
File "<pyshell#6>", line 1, in <module>
age
NameError: name 'persona' is not defined
```

IndexError es lanzado al intentar acceder a un índice que no existe en un array.

```
>>> lista = [1,2,3]
>>> lista[5]
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
lista[5]
IndexError: list index out of range
```

KeyError es lanzado cuando no se encuentra la clave (key),

```
>>> diccionario={'1':"esto", '2':"es", '3':"python"}
>>> diccionario['4']
Traceback (most recent call last):
File "<pyshell#15>", line 1, in <module>
diccionario['4']
KeyError: '4'
```

ModuleNotFoundError es lanzado cuando no se encuentra el módulo indicado.

```
>>> import mimodulo
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
import mimodulo
ModuleNotFoundError: No module named 'mimodulo'
```

Coding time!

Ejercicio 1

Crea un programa que acceda a la posición que el usuario indique de la siguiente lista: `[6,14,11,3,2,1,15,19]`. No olvides capturar las excepciones que puedan surgir en caso de que el usuario introduzca un índice incorrecto o que no exista en la lista.

Ejercicio 2

Crea una aplicación reciba la clave de un diccionario y acceda a uno de sus valores. Asegúrate de que capturas las excepciones que puedan saltar al intentar acceder a una clave del diccionario inexistente.

10. Clases y objetos

Python soporta la programación orientada a objetos. Esto quiere decir que podemos definir entidades agrupando (encapsulando) sus atributos y comportamiento (métodos) en clases.

La definición de una clase en Python se hace de la siguiente manera:

```
class Persona:
        # atributos
        nombre = "Josune"
        edad = 24

        # metodos
        def camina(self):
                print(self.nombre + " está caminando")
```

A partir de una clase se pueden crear tantas **objetos** como se desee. Los objetos de una clase se conocen como **instancias**. Cada objeto **contiene los atributos y métodos de la clase** y podrá asignar a esos atributos unos valores concretos. Esto se conoce como el **estado de un objeto**.

```
p1 = Persona() # la variable p1 contiene un objeto de la clase Persona
p1.camina()
print(p1.nombre)
print(p1.edad)
```

Una función dentro de una clase se conoce como **método**. Las clases contienen el método especial `__init__` conocido como **constructor** y que sirve para inicializar un objeto. Al crear un objeto siempre se llama al constructor. Una diferencia importante con otros lenguajes como Java es que solo se puede definir un único constructor.

```
class Persona:
        def __init__(self, nombre, apellidos, edad):
                self.nombre= nombre
                self.apellidos = apellidos
                self.edad = edad

        def camina(self):
                print(self.nombre + " está caminando")
```

En la creación del objeto es necesario indicar los argumentos del constructor:

```
p1 = Persona("Mike", "Mendiola", 25)
p1.camina()
print(p1.nombre)
print(p1.edad)
```

El parámetro `self` de los métodos es una referencia a la propia instancia y se utiliza para acceder a las variables que pertenecen a la clase. Si no se define un constructor, la clase hereda uno que únicamente recibe el argumento `self`.

Atributos de clase vs Atributos de instancia

Los atributos definidos dentro del constructor se conocen como **atributos de instancia**, por lo tanto, los atributos definidos dentro de la clase pero fuera del constructor se conocen como **atributes de clase**.

La principal diferencia es que un atributto de clase puede ser accedido aunque no existan instancias de la clase. Además, si se modifica su valor, se modificará el valor en todas las instancias existentes de dicha clase.

```
class Demo:
        atrib_estatico = 123 # compartido por todos los objetos
        def __init__(self,numero):
                self.atrib_instancia = numero # específico de cada objeto

c1 = Demo(456)
c2 = Demo(789)

# Valor inicial
print("C1: Estatico {0} - Instancia: {1}".format(c1.atrib_estatico, c1.atrib_instanc\
ia))
# output: C1: Estatico 123 - Instancia: 456
print("C2: Estatico {0} - Instancia: {1}".format(c2.atrib_estatico, c2.atrib_instanc\
ia))
# output: C2: Estatico 123 - Instancia: 789

Demo.atrib_estatico = -1

print("C2: Estatico {0} - Instancia: {1}".format(c2.atrib_estatico, c2.atrib_instanc\
ia))
# output: C2: Estatico -1 - Instancia: 456
print("C2: Estatico {0} - Instancia: {1}".format(c2.atrib_estatico, c2.atrib_instanc\
ia))
```

```
# output: C2: Estatico -1 - Instancia: 789

c1.atrib_instancia = 999

print("C1: Estatico {0} - Instancia: {1}".format(c1.atrib_estatico, c1.atrib_instanc\
ia))
# output: C1: Estatico -1 - Instancia: 999

print("C2: Estatico {0} - Instancia: {1}".format(c2.atrib_estatico, c2.atrib_instanc\
ia))
# output: C2: Estatico -1 - Instancia: 789
```

Private y protected

A diferencia de otros lenguajes de Programación Orientada a Objetos, todos los métodos y atributos en Python son públicos. Es decir, **no es posible definir una variable como** `private` o `protected`.

Existe una convención de añadir como prefijo un **guión bajo** (_) a los atributos que consideramos como **protected** y dos guiones bajos (__) a las variables que consideramos private.

```
class Persona:
    def __init__(self, nombre, edad):
        self._nombre = nombre  # atributo protected
        self.__edad = edad # atributo private
```

Coding time!

Ejercicio 1

Crea la clase Coche que contenga las siguientes propiedades:

- `matrícula` (string)
- `marca` (string)
- `kilometros_recorridos` (float)
- `gasolina` (float)

La clase tendrá un método llamado `avanzar()` que recibirá como argumento el número de kilómetros a conducir y sumará los kilómetros recorridos al valor de la propiedad `kilometros_recorridos`. El método también restará al valor de `gasolina` el resultado de los kilómetros multiplicado por 0'1.
La clase también contendrá otro método llamado `repostar()` que recibirá como argumento los litros

introducidos que deberán sumarse a la variable `gasolina`.
Por último, será necesario controlar que el método avanzar nunca obtendrá un número negativo en la gasolina. En dicho caso, deberá mostrar el siguiente mensaje: "Es necesario repostar para recorrer la cantidad indicada de kilómetros".

Ejemplo:

- `repostar(50)` # gasolina = 50
- `recorrer(100)` # kilometros_recorridos = 100, gasolina = 40
- `recorrer(40)` # kilometros_recorridos = 140, gasolina = 36
- `recorrer(180)` # kilometros_recorridos = 320, gasolina = 18

Ejercicio 2

Crea una clase Robot que simule los movimientos de un robot y calcule la posición en la que se encuentra cada momento. El robot se moverá por un tablero infinito de coordenadas X e Y, podrá realizar los siguientes movimientos:

- Avanzar hacia adelante (A)
- Retroceder (R)
- Avanzar hacia la izquierda (I) o hacia la derecha (D)

El robot tendrá un método llamado `mueve()` que recibirá la orden como parámetro y otro método, `posicion_actual()`, que indicará su posición en las coordenadas X e Y. Al crear el robot este se inicializará a las coordenadas (0,0).

Ejemplo:

- mueve("A") # (1,0)
- mueve("A") # (2,0)
- mueve("I") # (2,-1)
- mueve("A") # (3,-1)
- mueve("D") # (3,0)
- mueve("D") # (3,1)
- mueve("D") # (3,2)
- mueve("R") # (2,2)
- mueve("I") # (2,1)

Ejercicio 3

Mejora el ejercicio anterior de forma que el robot pueda recibir una secuencia de movimientos.
Ejemplo:

- mueve("AADDADIR") # (2,2)

También deberá tener otros dos métodos: uno que devuelva todas las órdenes recibidas y otro capaz de listar los movimientos necesarios para volver a la posición inicial (0,0).

Ejercicio 4

Crea la clase Triangulo que almacene la longitud de cada uno de sus lados. Deberá contener los siguientes métodos:

- `area()`: devuelve el área del triángulo
- `forma()`: indica si se trata de un triángulo equilátero, isósceles o irregular.
- `perímetro()`: devuelve el perímetro del triángulo.

11. Herencia

La herencia es una técnica de la Programación Orientada a Objetos en la que una clase (conocida como **clase hija** o **subclase**) hereda todos los métodos y propiedades de otra clase (conocida como **padre** o **clase base**).

La sintaxis para definir una clase que herede de otra es la siguiente:

```
class ClaseBase:
        # código de la clase base

class Subclase(BaseClass):
        # código de la subclase
```

La subclase puede añadir funcionalidades. Esta técnica permite **reutilizar código**.

```
class Dispositivo:
        def __init__(self,identificador,marca):
                self.identificador = identificador
                self.marca = marca

        def conectar(self):
                print("¡Conectado!)

# la clase base se indica entre paréntesis
class Teclado(Dispositivo):
        def __init__(self,identificador,marca,tipo):
                # llamada al constructor del padre
                Dispositivo.__init__(self,identificador,marca)
                self.marca = marca
        # metodo de la subclase
        def pulsar_tecla(self,tecla):
                print(tecla)

t1 = Teclado("0001", "Logitech", "AZERTY")
t1.conectar()
t1.pulsar_tecla("a")
```

Herencia múltiple

Python soporta la herencia múltiple, es decir, heredad métodos y atributos de más de un padre. En caso de heredar atributos o métodos con el mismo nombre, Python dará prioridad al posicionado más a la izquierda en la declaración.

```
# en caso de conflicto Dispositivo tendrá prioridad sobre Periférico
class Teclado(Dispositivo, Periférico):
        # cuerpo de la clase
```

Coding time!

Ejercicio 1

Continuando con el Ejercicio 1 del tema anterior, crea una clase `Vehículo` y otra llamada `Bicicleta`. La clase Vehículo será el padré de `Coche` y `Bicicleta` y contendrá las propiedades y métodos comunes de ambos. La bicicleta no tendrá gasolina ni repostará, pero cada 50 kilometros necesitará invocar al método `hinchar_ruedas()` o no podrá continuar.

Ejercicio 2

Continuando con el Ejercicio 4 del tema anterior, crea una clase `Poligono` y otra llamada `Cuadrado`. La clase `Poligono` será el padre de `Triangulo` y `Cuadrado`, y contendrá las propiedades y métodos comunes de ambos. Ambos tendrán también otra propiedad llamada `color`.

12. Módulos y Paquetes

Módulos

Un módulo es un archivo de Python que contiene variables, funciones y clases. Es una forma de ordenar y reutilizar código ya que todo el contenido de un módulo es accesible por los archivos que lo importen.

```
# mundo.py

def hola_mundo():
    print("¡Hola Mundo!")

def adios_mundo():
    print("¡Adios Mundo!")
```

Para acceder a las funciones desde otro archivo Python se utiliza la palabra reservada `import`:

```
# app.py

import mundo

# Llamada a la función
mundo.hola_mundo()
```

También existe la posibilidad de importar únicamente objetos concretos de un módulo mediante la sintaxis `from ... import`:

```
# app.py

from mundo import adios_mundo

# Llamada a la función
adios_mundo()
```

De esta forma no es necesario escribir el nombre del modulo antes de utilizar la función. De igual manera, se pueden importar varios objetos de un módulo separándolos por una coma:

```
# app.py

from mundo import adios_mundo, hola_mundo
```

Para importar todos los los objetos de un módulo basta con utilizar el asterisco:

```
# app.py

from mundo import *
```

Localización de los módulos

Al importar un módulo Python lo buscara en los siguientes directorios:

1. En el directorio actual.
2. En los directorios declarados en el `PYTHONPATH` (variable de entorno que contiene un listado de directorios)
3. En el directorio de instalación de Python por defecto (en UNIX normalmente '`/usr/local/lib/python/`')

Paquetes

Es posible agrupar los módulos que tienen relación en un mismo directorio. Estos directorios son conocidos en Python como paquetes y deben contener siempre un archivo llamado `__init__.py` para que Python lo reconozca como un paquete.

A medida que desarrollamos una aplicación es habitual agrupar los archivos en directorios (paquetes) para tener el código organizado.

Para cargar un módulo ubicado en un paquete lo haremos de la siguiente forma:

```
import mipaquete.mundo
```

o bien de la siguiente manera:

```
from mipaquete import mundo
```

También es posible importar elementos concretos de un módulo:

```
from mipaquete.mundo import adios_mundo, hola_mundo
```

13. Próximos pasos

Como ya te habíamos adelantado, **tu viaje en el mundo de Python solo acaba de empezar**. Python es uno de los lenguajes de programación con mayor potencial, y ahora que ya dispones de una sólida base de los fundamentos básicos, podrás comenzar a explorar otros mundos como:

- Desarrollo de aplicaciones web
- Internet of Things
- Análisis de datos (Data Science, Machine Learning, Big Data,...)
- Scripting
- Aplicaciones embebidas
- Desarrollo de aplicaciones de escritorio

Podrás profundizar en cada una de estas ramas a partir de los conocimientos de los que dispones. En función del camino que elijas, podrás aprender frameworks como Django, Flask, Numpy, PyTorch, PANDAS, TensorFlow, Libmraa, ...

A continuación te listo algunos de los recursos a los que sin duda alguna merecen la pena darles un vistazo:

- Tutoriales de Data Camp[1]
- Head First Python[2]
- Python Crash Course[3]
- Learn Python 3 the Hard Way[4]
- Python for Data Analysis[5]
- Google's Python Class[6]
- Curso "Introduction to Python", de Udacity[7]

Recuerda, lo único que necesitas saber es que **tú puedes aprender cualquier cosa**.

---

[1] https://www.learnpython.org/

[2] https://www.amazon.es/dp/1491919531/?tag=devdetailpa03-21

[3] https://www.amazon.es/Python-Crash-Course-Hands-Project-Based-ebook/dp/B018UXJ9RI/ref=sr_1_3?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=python&qid=1570180835&sr=8-3

[4] https://www.amazon.es/Learn-Python-Hard-Way-Introduction-ebook/dp/B07378P8W6/ref=sr_1_7?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=python&qid=1570180896&sr=8-7

[5] https://www.amazon.es/Python-Data-Analysis-Wrangling-IPython-ebook/dp/B075X4LT6K/ref=sr_1_13?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=python&qid=1570180896&sr=8-13

[6] https://developers.google.com/edu/python/

[7] https://www.udacity.com/course/introduction-to-python--ud1110