INTRODUCCIÓN A LA PROGRAMACIÓN CON PYTHON



Práctica 1 - Introducción a Python

El lenguaje Python es un lenguaje:

- De propósito general: porque se puede hacer con él cualquier tipo de programa como aplicaciones web, aplicaciones científicas, aplicaciones de bajo nivel, aplicaciones de escritorio (Instagram, Word, Robótica, Inteligencia artificial, Big Data), etc.
- De alto nivel: ya tenemos una gran cantidad de funciones implementadas y podemos pasar directamente a realizar nuestra aplicación sin tener que definir por ejemplo cómo pintar píxeles, cómo llevar la gestión de la memoria, etc.
- Dinámico: mientras estamos escribiendo el código vamos viendo inmediatamente la respuesta que da el programa.
- Que tiene una sintaxis bastante sencilla y es muy útil para iniciarse en la programación.

Por qué utilizar Phyton?

- Es sencillo de aprender
- Tiene un lenguaje claro y legible
- Tiene muchos módulos o librerías ya predefinidos
- Es altamente interactivo, gratuito y hay una gran cantidad de desarrolladores detrás que pueden ayudar.

Para empezar a utilizar Phyton necesitamos dos programas:

- Un intérprete de Phyton. El ordenador necesita interpretar/entender el lenguaje. Se descarga desde su página web: https://www.python.org (SI NO ESTÁ INSTALADO EN EL ORDENADOR HACERLO AHORA)

- Un editor de código: este programa nos ayuda a escribir el código. Hay muchos aunque aquí en estas prácticas vamos a utilizar Visual Studio Code. Lo descargamos desde: https://code.visualstudio.com/ (SI NO ESTÁ INSTALADO EN EL ORDENADOR HACERLO AHORA)

Una de las cosas importantes de **Python** que debes recordar es que es *Case Sensitive* (sensible a las mayúsculas), es decir que para **Python** no es lo mismo escribir *Print* que *print*.
Conviene indicar también que no hace falta separar las instrucciones utilizando ;

El ordenador trabaja con 0 y 1. Si escribiéramos un programa en el lenguaje del ordenador nos volveríamos locos. Para eso están lenguajes como Python. Nos permiten escribir un programa de manera que sea fácil entenderlo por una persona (el **código fuente**). Luego es el intérprete el que se encarga de convertirlo al complicado idioma de un ordenador.

La principal diferencia entre un lenguaje compilado y uno interpretado es que el **lenguaje compilado** requiere un paso adicional antes de ser ejecutado, la compilación, que convierte el código que escribes a **lenguaje de máquina**. Un **lenguaje interpretado**, por otro lado, es convertido a **lenguaje de máquina** a medida que es ejecutado.

Ejemplos de **lenguajes compilados** incluyen C, C++, Java, Go y Rust, entre muchos otros. Ejemplos de **lenguajes interpretados** incluyen Ruby, Python y JavaScript, entre muchos otros. A todos estos lenguajes (compilados e interpretados) se les conoce como **lenguajes de alto nivel**.

En el caso de los lenguajes compilados, si tenemos el código fuente podemos modificar el programa tantas veces como queramos (sólo tenemos que volver a compilarlo), pero si tenemos el ejecutable final no podremos cambiar nada (realmente sí se puede pero es mucho más complicado y requiere más conocimientos).

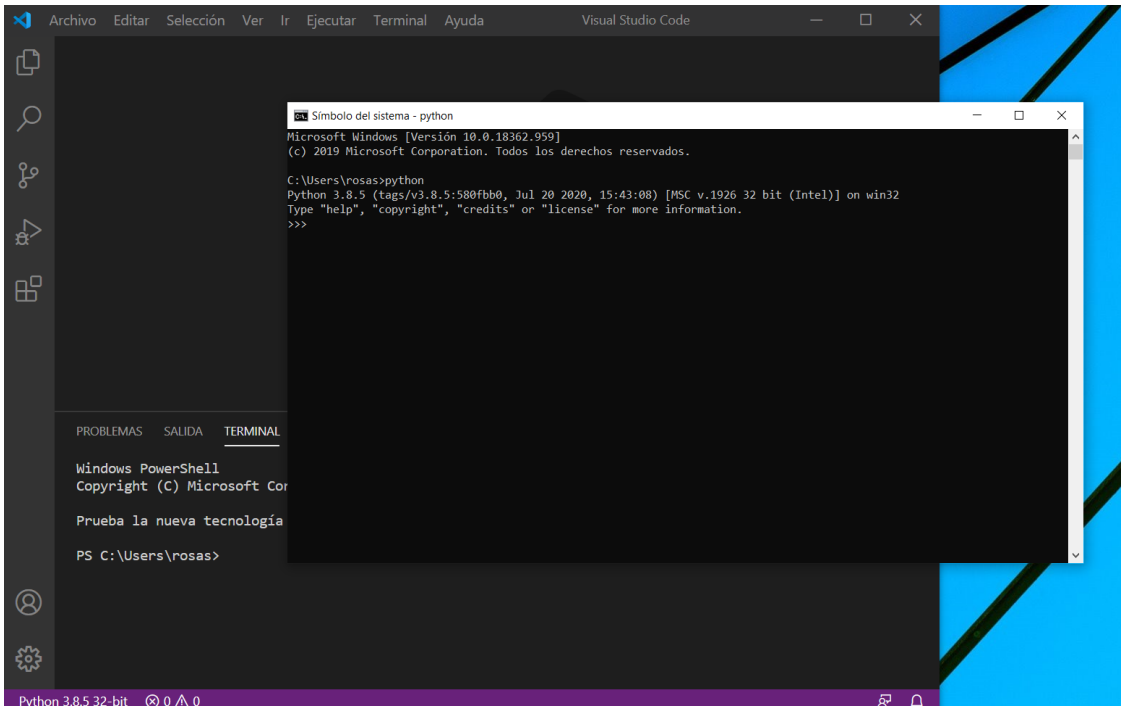
1) Antes de nada crearemos dentro de nuestra carpeta de alumno una carpeta llamada **'PROGSPYTHON'.** Dentro de ella iremos almacenando los diferentes programas que vayamos realizando en clase.

2) Instalar el intérprete de Python y el editor Visual Studio Code. Para ello habrá que descargarlo previamente e instalarlos en el ordenador (marcar la opción de agregar al PATH)





3) Abrir ambos programas. Para abrir el intérprete vamos a la consola de MS-DOS escribiendo "cmd" y ahí escribimos "python". Abrir también Visual Studio Code (se puede utilizar también la consola del editor si se prefiere).



El primer programa

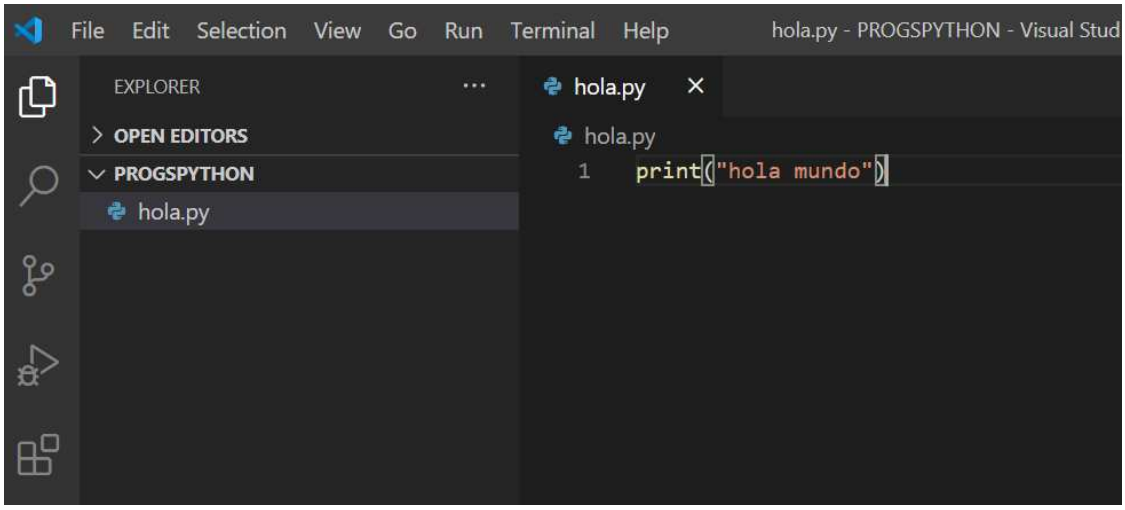
4) En un alarde de originalidad vamos a hacer nuestro primer programa: "hola mundo". Nadie puede llegar muy lejos en el mundo de la programación sin haber empezado su carrera con este original y funcional programa. Escribimos en el intérprete de Python el siguiente código:

```
"hola mundo"
```

Ahora vamos a hacer lo mismo pero desde el editor. Para ello seguimos los siguientes pasos:

- Arrastrar la carpeta PROGSPYTHON al editor.
- Pulsar sobre el icono de archivo nuevo.
- Llamamos al archivo "**hola.py**" (TODOS LOS PROGRAMAS EN PYTHON ACABARÁN CON LA EXTENSIÓN .PY)
- Escribimos el siguiente código:

```
print("hola mundo")
```



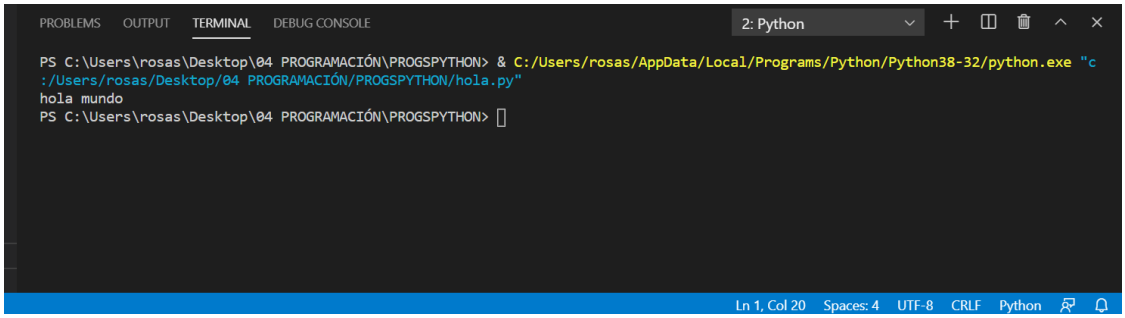
- Guardamos el programa y para ejecutarlo podemos ir a la terminal y ejecutar "python hola.py" o darle al botón derecho encima del archivo seleccionar "Ejecutar archivo Python en la terminal"

¿Qué fácil eh? Este programa lo único que hará será sacar por pantalla el mensaje:

```
Hola mundo
```

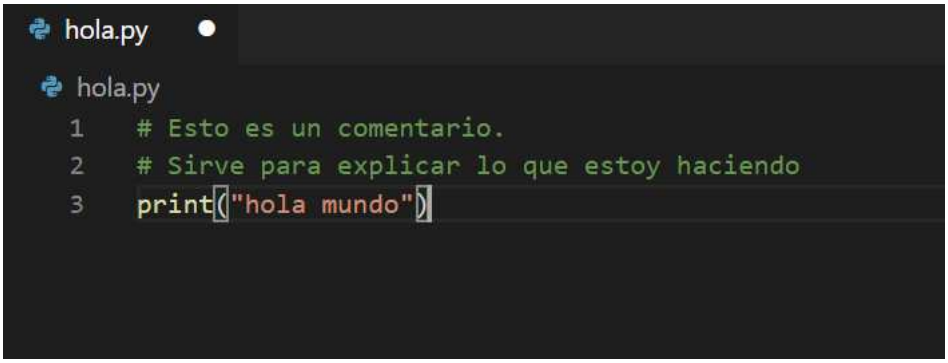
Si todo ha ido bien nos saldrá un mensaje como este donde nos informa de que no hay errores ni advertencias (warnings).

Si ha salido algún error mirad en qué línea os dice que está el error y revisar donde os habéis equivocado.



Si queremos poner algún comentario en el programa se utiliza el símbolo #

Para poner comentarios de más de una línea se pone " " " al principio y " " " al final
Los comentarios se utilizan para hacer aclaraciones para más adelante o para otros programadores. El programa se lo salta. Ejemplo:



→ 5) Realizad un programa de similares características que el que realizasteis en la práctica anterior pero cambiando el texto que muestra la función print. En concreto debe sacar la fecha y la hora actual. Añadir también varias líneas de comentarios.

→ 6) Guardarlo con el nombre **'prueba.py'** en la carpeta "**PROGSPYTHON**"

→ 7) Ejecutar el programa y comprobar que se ejecuta correctamente y hace lo que esperábamos.

→ 8) Realiza un programa que saque por pantalla los números del 1 al 5, uno en cada línea.

→ 9) Guardar el programa con el nombre **'numeros.py'** en la carpeta "**PROGSPYTHON**" y ejecutarlo para comprobar que se obtiene el resultado esperado.

Práctica 2 – Tipos de datos en Python

Dependiendo de lo que vayamos a sacar por pantalla los datos se pueden agrupar en diferentes tipos:

- String (cadenas de caracteres)
- Int (enteros)
- Float (Reales)
- Bool (Lógicos o booleanos)
- List (Listas de datos)
- Tuples (Tuplas)
- Dictionary (Diccionarios de datos)

1) Crear un nuevo programa llamado "**tipodedatos.py**" y probarlo mostrando por pantalla diferentes tipos de datos:

tipodedatos.py

```python
# String
print("Hola")
print("Hoy puede ser un buen día")

# Int
print(40)
print(100-20)
print(8*5)

# Float
print(40.5)

# Boolean
print(True)
print(False)

# List - pueden cambiar
print([10, 20, 30])
["hola", "adios", "uno"]
[10, "hola", False]

# Tuple - no se pueden cambiar
(10,20,30)

# Dictionary - como las listas pero asignadas a un objeto
{
    "nombre": "Paco",
    "apellido": "Martínez",
    "apodo": "Ryan"
}
```

2) Crear un nuevo programa llamado "**saludo.py**" que muestre el siguiente mensaje:
Hola (tu nombre) tienes (tu edad) años.
También debe mostrar por pantalla la multiplicación de 30*4

3) Añadir al programa anterior "**tipodedatos.py**" el siguiente código:

```
32  print(type("Hola que tal"))
33  print(type(40))
34  print(type (30.5))
35  print(type (False))
36  print(type(["hola", "adios", "uno"]))
37  print(type((10,20,30)))
38  print(type({
39      "nombre": "Paco",
40      "apellido": "Martínez",
41      "apodo": "Ryan"
42  }))
```

4) Guardarlo con el nuevo nombre "**tipodedatos2.py**" y ejecutarlo para comprobar que funciona correctamente y muestra cada uno de los tipos de datos.

Práctica 3 – Variables en Python

Cuando usamos un programa es muy importante manejar datos. En **Python** podemos almacenar los datos en variables. El contenido de las variables se puede ver o cambiar en cualquier momento. Estas variables pueden ser de distintos tipos dependiendo del tipo de dato que queramos meter. No es lo mismo guardar un nombre que un número.
EN PYTHON NO ES NECESARIO DECLARAR PREVIAMENTE LAS VARIABLES.

Notas sobre los nombres de las variables

A las variables no se les puede dar cualquier nombre. No se pueden poner más que letras de la 'a' a la 'z' (la ñ no vale), números y el símbolo '_'. No se pueden poner signos de admiración, ni de interrogación... El nombre de una variable puede contener números, pero su primer carácter no puede serlo.

Ejemplos de nombres válidos:

```
camiones
numero
buffer
a1
j10hola29
num_alumnos
```

Ejemplos de nombres no válidos:

```
1abc
nombre?
num/alumnos
```

Tampoco valen como nombres de variable las **palabras reservadas** que usa el compilador. Por ejemplo: *for, def, if, while.*

Por último es interesante señalar que el **Python** distingue entre mayúsculas y minúsculas (CASE SENSITIVE). Por lo tanto estas tres variables serían variables distintas:

```
Nombre
nombre
NOMBRE
```

El tipo String

Las variables de tipo String sirven para almacenar cadenas de caracteres.

1) Copiar el siguiente programa, probarlo y guardarlo con el nombre "**presenta.py**". Analizar qué se está haciendo en cada caso.

presenta.py > ...

```
1   print("Mi nombre es \n Luis")
2   print("Mi nombre es \t Luis")
3
4   nombre="Luis"
5
6   print("Mi nombre es", nombre)
7   print(f"Mi nombre es {nombre}")
8
9   edad=18
10
11  print("Mi nombre es", nombre, "y tengo", edad, "años")
12  print(f"Mi nombre es {nombre} y tengo {edad} años")
```

2) Crea un nuevo programa llamado "**misdatos.py**" que almacene en tres variables: tu nombre, tu curso y tu instituto. A continuación muéstralas por pantalla en una sola línea de dos formas diferentes.

Operaciones con String

Con estas cadenas podemos utilizar muchas funciones asociadas.

3) Copiar y probar el siguiente programa. Guardarlo con el nombre "**cadenas.py**"

cadenas.py > ...

```
1   miCadena="Hola mundo"
2   print(miCadena)
3
4   print(dir(miCadena))
5
6   print(miCadena.upper())
7   print(miCadena.lower())
8   print(miCadena.swapcase())
9   print(miCadena.capitalize())
10  print(miCadena.replace ("Hola", "Adios"))
11  print(miCadena.startswith("H"))
12  print(miCadena.startswith("J"))
13  print(miCadena.endswith("mundo"))
14  print(len(miCadena))
15
```

4) Añadir ahora comentarios a la derecha indicando qué hace cada una de las funciones anteriores. Guardar el programa anterior con el nombre "**cadenas2.py**"

Los tipos numéricos

A la hora de almacenar datos numéricos el programa se va a adaptar al dato que introduzcamos, sea entero o real. Con estos datos podremos hacer las operaciones básicas: suma, resta, multiplicación, etc.

5) Copia el siguiente programa, pruébalo y guárdalo con el nombre "**numeros.py**"

numeros.py > ...

```
1
2   num = 10
3   print("El valor inicial de num es", num)
4
5   num = 50
6   print("Ahora el valor de num es", num)
7
```

6) Poner comentarios al programa que expliquen qué se está haciendo en cada línea. Guardar el archivo con el nombre **'numeros2.py'**. La salida por pantalla deberá ser:

```
El valor inicial de x es 10
Ahora el valor es 50
```

7) Crear un programa de similares características al anterior que nos muestre por pantalla el valor de 3 variables: a, b y c , a las que previamente habremos dado estos valores, a=1, b=2, c=3.

La salida por pantalla deberá ser:

```
El valor de la variable a es 1
El valor de la variable b es 2
El valor de la variable c es 3
```

8) Guardarlo con el nombre **'numeros3.py'** y comprobar que funciona correctamente.

9) Crear un programa donde se muestren las letras del alfabeto desde la a hasta la h, cada una de ellas en una línea diferente. Para ello crearemos una variable llamada letra la que le iremos cambiando el valor y mostrando por pantalla. Guardar el programa con el nombre **'letras.py'** y comprobar que funciona.

10) Realizar el siguiente programa y corregir los errores para que el programa funcione correctamente (hay 3). Guardarlo con el nombre **'errores.py'** y comprueba que funciona correctamente.

errores.py > ...

```
1   número1 = 4
2   numero2 = 9;
3   numero3 = numero1*numero2
4
5   print("El valor final es", Numero3)
6
```

11) Fijándote en el programa anterior realiza uno nuevo que realice la multiplicación de 3 números: a=5, b=3, c=8. Guárdalo con el nombre **'multiplicacion.py'** y comprueba que el resultado es el correcto.

12) Realiza un programa similar al anterior pero en el que se realicen dos divisiones. Una debe dar resultado entero y la otra con decimales. Guardar el programa con el nombre "**decimales.py**"

El tipo Lista

Las variables de tipo List sirven para almacenar elementos en una lista. Se introducen los datos entre corchetes [ ]

13) Copiar el siguiente programa, probarlo y guardarlo con el nombre "**listas1.py**". Analizar qué se está haciendo en cada caso.

```
listaPrueba=[1,2,3,4]
colores=['amarillo', 'naranja', 'rojo']

print(type(colores))
print(colores)
print(listaPrueba)
print(len(listaPrueba))
print(colores[2])
```

14) Crea un nuevo programa llamado "**miLista.py**" que almacene en una lista: tu nombre, tu primer apellido y tu segundo apellido. A continuación muéstrala por pantalla y obtén la longitud de la lista.

Con estas listas podemos utilizar muchas funciones asociadas.

15) Añade al programa "**listas1.py**" el siguiente código.

```
print(dir(colores))

colores.append('violeta')
colores.extend(['blanco','azul'])
colores.insert(-1,'verde')
print(colores)
print(colores.count('amarillo'))
```

16) Añadir ahora comentarios a la derecha indicando qué hace cada una de las funciones anteriores. Guardar el programa anterior con el nombre "**listas2.py**"

El tipo Tupla

Las variables de tipo Tuple sirven para almacenar elementos. Es un tipo de datos muy parecido a una lista, pero en este caso los datos no se pueden modificar. Dependiendo de para qué sea nuestro programa utilizaremos una tupla o una lista. Aquí los valores se ponen entre paréntesis ( ).

17) Copiar el siguiente programa, probarlo y guardarlo con el nombre "**tuplas1.py**". Analizar qué se está haciendo en cada caso.

```
tuplaPrueba=(1,2,3,4)
meses=('enero', 'febrero', 'marzo')

print(type(meses))
print(meses)
print(tuplaPrueba)
print(len(tuplaPrueba))
print(meses[2])

print(dir(meses))
print(meses.count('enero'))
```

18) Añadir ahora comentarios a la derecha indicando qué hace cada una de las funciones anteriores. Guardar el programa anterior con el nombre "**tuplas2.py**"

El tipo Dictionary

Las variables de tipo Dictionary sirven para almacenar elementos pero de forma un poco más compleja. Aquí los ponemos utilizando claves y valores. Los valores se ponen entre llaves {}

19) Copiar el siguiente programa, probarlo y guardarlo con el nombre "**dictionary1.py**". Analizar qué se está haciendo en cada caso.

```
producto = {
    "nombre": "libro",
    "cantidad": 8,
    "precio":4.99
}

persona = {
    "nombre": "Pepe",
    "apellido": "Brotons"
}

print(type(persona))
print(producto)
```

```
print(persona.keys())
print(persona.items())

persona.clear()
print(persona)
```

20) Añadir ahora comentarios a la derecha indicando qué hace cada una de las funciones anteriores. Guardar el programa anterior con el nombre "**dictionary2.py**"

21) Crea un programa similar al anterior pero en este caso que almacene y muestre los datos de un deporte (nombre, precio, dificultad). Guardar el programa anterior con el nombre "**dictionary3.py**"

Práctica 4 – Operadores aritméticos

Un operador sirve para manipular datos. Los hay de varios tipos: de asignación, de relación, lógicos, aritméticos y de manipulación de bits.

Operador de asignación (=)

Este es un operador que ya hemos visto anteriormente. Sirve para dar un valor a una variable. Este valor puede ser un número que tecleamos directamente u otra variable:

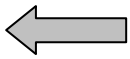
```
a = 3;          # Metemos un valor directamente
o
a = b;          # Le damos el valor de una variable
```

Podemos dar valores a varias variables a la vez:

```
a = b = c = 10          # Damos a las variables a,b,c el valor 10
```

También podemos asignar a varias variables el valor de otra de un sólo golpe:

```
a = b = c = d;          # a,b,c toman el valor de d
```

Con este operador, el objeto de la izquierda pasa a tener el valor del objeto de la derecha 

Operador Suma (+)

Este operador permite sumar variables.

```
a = 2
b = 3

c = a + b
print( "Resultado = ", c )
```

El resultado dará, obviamente 5.

Podemos utilizar este operador para incrementar el valor de una variable:

```
x = x + 5     o también se puede poner x+=5
```

En este ejemplo, si x valía 13, ahora pasa a valer 18. Veamos ahora un programa de ejemplo:

```
x = 3
y = 5

x = x+2
print( "x = ", x )
x = x + y
print("x = ", x )
```

Resultado:

```
x = 5
x = 10
```

→ 1) Realizar un programa que calcule la suma de tres números cualesquiera (los asignáis vosotros) y que tras mostrar el resultado por pantalla, al final incremente cada una de las variables en 5 unidades y las vuelva a mostrar. Guardar el programa con el nombre **'Suma.py'** y comprobar que funciona correctamente.

Operador Resta / Negativo (-)

Este operador tiene dos usos, uno es la resta, que funciona como el operador suma, y el otro es cambiar de signo.

```
x = x - 5
```

En este ejemplo, si x valía 13, ahora pasa a valer 8. Para la operación resta se aplica todo lo dicho para la suma.

Por otra parte, también tiene el uso de cambiar de signo. Poniéndolo delante de una variable o constante equivale a multiplicarla por -1. Veamos un programa de ejemplo:

```
a = 1
b = -a
print( "a = ", a, "b = ", b)
```

Resultado:

```
a = 1, b = -1
```

→ 2) Realizar un programa que calcule la resta de dos números (los asignáis vosotros) y que tras mostrar el resultado por pantalla, al final decremente cada una de las variables en 1 y las vuelva a mostrar. Guardar el programa con el nombre **'Resta.py'** y comprobar que funciona correctamente.

*Operador Multiplicación (*)*

Este operador sirve para multiplicar y funciona de manera parecida a los anteriores.

Operador División (/) y División Entera (//)

Este funciona también como los anteriores pero hay que tener cuidado. Si dividimos dos número en coma flotante (tipo float) tenemos las división con sus correspondientes decimales.

Pero si dividimos dos enteros obtenemos un número entero. Es decir que si queremos dividir 4/3 como enteros tenemos que poner 4//3 y el resultado será 1. El redondeo se hace por truncamiento, simplemente se eliminan los decimales y se deja el entero. Por eso debemos que recordar que **int** es solamente para trabajar con números enteros. Para ello se utiliza //

Operador Resto (%)

Si con el anterior operador obteníamos el cociente de una división entera, con éste podemos obtener el resto. Por tanto, cuando dividimos dos enteros, si queremos saber cual es el resto (o módulo) usamos el operador **%**

No funciona más que con enteros, no vale para números float.

→ 3) Para ver cómo se usa copia este ejemplo y guárdalo con el nombre "**misOperaciones.py**"

misOperaciones.py > ...

```
1   a=19
2   b=5
3
4   print("a vale ",a, "b vale ",b)
5   print("Suma de a + b =", a+b)
6   print("Resta de a - b =", a-b)
7   print("Multiplicación de a * b =", a*b)
8   print("División real de a / b =", a/b)
9   print("División entera de a // b =", a//b)
10  print("Resto de a // b =", a % b)
11
```

→ 4) Añade comentarios al programa anterior indicando qué estamos haciendo en cada línea y guardarlo con el nombre "**misOperaciones2.py**"

*Operador Elevado a **)*

Si lo que queremos calcular es la potencia de un determinado número utilizamos el operador ** para hacer dicho cálculo.

5) Para ver cómo se usa copia este ejemplo y guárdalo con el nombre "**misOperaciones2.py**"

```
num1=input("Introduce un numero: ")
num2=input("Introduce otro numero: ")

num3=int(num1)**int(num2)

print(num1, "elevado a ", num2, "es igual a ", num3)
```

6) Para practicar todos los anteriores operadores vamos a realizar un programa que tenga tres variables: x, y, z. Tiene que realizar las siguientes operaciones (los valores los asignáis vosotros):

- calcular la multiplicación de x por y , almacenando el valor en z (mostrar el resultado)
- calcular la suma de x e y , almacenando el valor en z (mostrar el resultado)
- calcular la división de x entre y , almacenando el valor en z (mostrar el resultado)
- calcular el valor de y elevado al cubo y el de x elevado al cuadrado.
- calcular el resto de la división entre x e y , almacenando el valor en z (mostrar el resultado)
- incrementar el valor de z en 10 y mostrar el resultado final.

7) Guardar el programa con el nombre **'Operaciones.py'** y comprobar que funciona correctamente.

Práctica 5 - Lectura de datos desde el teclado

Algo muy usual en un programa es esperar que el usuario introduzca datos por el teclado. Para ello vamos a utilizar la función input()

Input( )

El uso de input es muy similar al de print con una diferencia, nos da la posibilidad de que el usuario introduzca datos en vez de mostrarlos.

Veamos la sintaxis de esta función por medio de un ejemplo:

```
prueba=input("Introduce un numero... " )
print("Has introducido el número: ", prueba)

print("Vamos a restarle 5")
print(int(prueba)-5)
```

Veamos cómo funciona input( ). Lo primero es fijarnos en que hay que almacenar en una variable sin tipo el dato que introduzcamos. En principio ese dato será una cadena.
Podemos mostrarlo directamente con print, pero si es un número y vamos a hacer operaciones con él tenemos que forzar su conversión a un entero o real int() o float()

→ 1) Copiar el ejemplo anterior y guardarlo con el nombre "**input1.py**". Comprobar que funciona correctamente.

→ 2) Realizar el mismo programa pero ahora hay que modificarlo para que lea por teclado un **número real con decimales (float)** que almacenaremos en una variable llamada **"real"**. Luego, mostrar ese número por pantalla y le restaremos 5. Guardar el programa con el nombre **'input2.py'**.

→ 3) Realizar un programa de similares características al ejemplo anterior pero que lea por teclado un **carácter** que almacenaremos en una variable llamada **"var"**. Por último, mostrar esa letra por pantalla. Guardar el programa con el nombre **'input3.py'**.

Podemos forzar que se introduzca un valor con un tipo de datos determinado, por ejemplo:

```
numero1 = int(input("Dígame un número: "))
numero2 = int(input(f"Dígame un número mayor que {numero1}: "))
print(f"La diferencia entre ellos es {numero2 - numero1}.")
```

→ 4) Realizar un programa que lea los siguientes datos por teclado:

- un carácter y luego lo muestre por pantalla
- un numero entero, lo multiplique por 5 y luego lo muestre por pantalla
- un float, lo decremente en 3 y luego lo muestre por pantalla

→ 5) Guardarlo con el nombre **'input4.py'** y comprobar que funciona correctamente.

6) Copiar el siguiente ejemplo, guardarlo con el nombre teclado "**teclado.py**", probarlo y añadir comentarios indicando qué se está haciendo en cada caso.

```
cadena = input("Introduce una cadena de texto: ")
numero_entero=int(input("Dime un número entero: "))
numero_real=float(input("Dime un numero real: "))

print("La cadena de texto es:", cadena)
print("El número entero es el ", numero_entero)
print("El número real es el ", numero_real)

num1=numero_entero**3
num2=numero_real*5
num3=numero_entero // 5

print(num1, num2, num3)
```

Operadores de comparación y lógicos

Los operadores de condición y lógicos se utilizan para comprobar las condiciones de las sentencias de control de flujo (las estudiaremos más adelante). Los operadores de comparación son:

= =	igual que	se cumple si son iguales
!=	distinto que	se cumple 1 si son diferentes
>	mayor que	se cumple si el primero es mayor que el segundo
<	menor que	se cumple si el primero es menor que el segundo
>=	mayor o igual que	se cumple si el primero es mayor o igual que el segundo
<=	menor o igual que	se cumple si el primero es menor o igual que el segundo

Los operadores lógicos son:

Operador	Descripción	Ejemplo
and	Devuelve True si ambos valores son True	x>5 and x>15
or	Devuelve True si uno de los valores es True	x>5 or x<0
not	Negación de la sentencia	not(x>5 or x<0)

Un ejemplo de todo esto se puede ver a continuación:

```
print( "10 > 5 da como resultado", 10>5)
print( "10 < 5 da como resultado", 10<5)
print( "5 == 5 da como resultado", 5==5)
print( "10 == 5 da como resultado", 10==5)
```

De lo obtenido podemos ver que 1 = cierto y 0 = falso.
No sólo se pueden comparar constantes, también se pueden comparar variables.

7) Crear un programa de similares características al ejemplo anterior pero que compare letras las letras 'c' y 'h'. Guardarlo con el nombre **'comparacion.py'**.

Práctica 6 – Condicionales

Hasta aquí hemos visto cómo podemos introducir un conjunto de instrucciones las veces que deseemos. Pero ahora vamos a ver cómo podemos controlar totalmente el flujo de un programa. Dependiendo de los valores de alguna variable se tomarán unas acciones u otras. Empecemos con la sentencia *if*.

Condicional – If –

La palabra *if* significa *si* (condicional). Su formato es el siguiente:

if *condición* **:**
    instrucciones a ejecutar

Cuando se cumple la condición se ejecuta el bloque inmediatamente siguiente al if (bloque *instrucciones a ejecutar*). **Si no se cumple la condición entonces no se ejecuta lo que hay tabulado (SE TIENE QUE PONER TABULACIÓN).**

En el siguiente ejemplo tenemos un programa que nos pide un número, si ese número es 10 se muestra un mensaje. Si no es 10 no se muestra ningún mensaje:

```
num=int(input("Introduce un numero:"))

if num==10:
    print("El numero es correcto")
```

→ 1) Crear un programa parecido al ejemplo que lea un número por teclado. Si ese número es igual a 5 que imprima por pantalla el texto “Has pulsado el número 5”. **HAY QUE DESTACAR QUE NO ES LO MISMO = QUE = =**

→ 2) Añade comentarios al programa que acabas de crear que expliquen qué se está haciendo en cada línea del programa.

→ 3) Guarda el programa con el nombre “**if.py**” y comprueba que realiza lo esperado.

Condicional If - Else

Hay veces que queremos que si no se cumple la condición, se ejecuten otras sentencias. Para esto utilizamos las funciones if – else. El formato de las mismas es el siguiente:

if condición **:**
    bloque que se ejecuta si se cumple la condición

else :
    bloque que se ejecuta si no se cumple la condición

En el if si no se cumplía la condición no se ejecutaba el bloque siguiente y el programa seguía su curso normal. Con el if else tenemos un bloque adicional que **sólo se ejecuta si no se cumple la condición del if.** Veamos un ejemplo:

```
num=int(input("Introduce un numero:"))

if num==8:
    print("El numero introducido era un ocho")

else:
    print("Va a ser que no has escrito un ocho!!!" )
```

Al ejecutar el programa si introducimos un 8 se ejecuta el bloque situado dentro de las llaves del if y se muestra el mensaje:

El número introducido era un ocho.

Si escribimos cualquier otro número o letra que no sea un ocho, se ejecuta el bloque situado dentro del else, mostrándose el mensaje:

Va a ser que no has escrito un ocho!!!

4) Crear un programa parecido al ejemplo que lea un número por teclado, si ese número es igual a 6 que imprima por pantalla el texto "Has pulsado el número 6" y si no has pulsado ese número que muestre el mensaje "No has pulsado el número 6".

Guarda el programa con el nombre "**if_else.py**" y comprueba que realiza lo esperado.

5) Crea un programa donde se utilice if-else que lea un número por teclado y luego realice lo siguiente:

- Si el número es menor que 10, lo debe multiplicar por 5 y mostrar el resultado.
- Si no se cumple esta condición que le reste 10 y lo muestre por pantalla.

Guarda el programa con el nombre "**if_else2.py**" y comprueba que realiza lo esperado.

Condicional If – elif - else

Podemos utilizar if else anidados si lo deseamos, así podemos establecer muchas condiciones a la vez. Observa el siguiente ejemplo:

```
a=int(input("Introduce un numero:"))

if a<10:
    print("El número introducido era menor de 10")

elif a>10 and a<100:
    print("El número está entre 10 y 100")

elif a>100:
    print("El número es mayor que 100")
```

```
else:
    print("El número era el 10")
```

Para comprender este programa es necesario puedes seguir estos pasos:

- El símbolo **and** de la condición del segundo if debe cumplirse también. De esta forma la condición queda: *Si a es mayor que 10* ***Y*** *a es menor que 100*.
- Si la condición del primer if es verdadera se muestra el mensaje *El número introducido era menor de 10* y se saltan todos los if-else siguientes.
- Si la condición es falsa se ejecuta el siguiente else-if y se comprueba si a está entre 10 y 100.
- Si es cierto se muestra *El número está entre 10 y 100*.
- Si no es cierto se evalúa el siguiente else-if.
- Si no se ejecuta ninguno de los apartados anteriores se ejecuta lo del else

→ 6) Copiar el programa del ejemplo anterior. Compilar y comprobar que funciona correctamente, a continuación poner comentarios que aclaren qué se ha hecho en cada línea. Guardar el programa con el nombre **"condicional.py"** en tu carpeta PROGSPYTHON

→ 7) Escribir un programa que compruebe si un número introducido por el teclado es par o impar.
Pista: Utilizaremos la función módulo (%), sabiendo que cuando se divide un número entre 2 sus posibles restos son 0 (número par) o 1 (número impar).
Guardarlo con el **'par.py'** y comprobar que realiza la función deseada.

→ 8) Realizar un programa que nos resuelva una ecuación de primer grado ax+b=0 , donde a y b son los datos y las posibles soluciones son:

a!=0		entonces	Mostrar por pantalla x = -b/a
a==0	b!=0	entonces	Mostrar por pantalla "solución imposible"
a==0	b==0	entonces	Mostrar por pantalla "solución indeterminada"

Guardarlo con el nombre **'ecuacion.py'** y comprobar que realiza la función deseada.

If anidados

Podemos incluir unos condicionales dentro de otros, como podemos ver en este ejemplo:

```
nombre=input("Introduce tu nombre:")
apellido=input("Introduce tu apellido:")

if nombre=="Luis":
    if apellido=="Escandell":
        print("Eres Luis Escandell")

else:
    print("No eres Luis")
```

9) Copiar el programa anterior y guardarlo con el nombre "**condicionalAnidado.py**"

10) Modificar el programa anterior para que sólo tenga un if con una condición and. Guardarlo con el nombre "**condicionalAnd.py**"

11) Realizar un programa que lea dos números por teclado ("a" y "b") y luego realice las siguientes funciones:

- Si a > b que reste a menos b (a-b)
- Si a < b que sume a y b (a+b)
- Si a == b que multiplique a por b (a*b)

Guardarlo con el nombre **'numeroif.py'** y comprobar que realiza la función deseada.

12) Realizar un programa que pida una hora determinada (con el formato horas, minutos y segundos). Si la hora es >0 tiene que pasarlo todo a segundos, por ejemplo: 1 h 25 min 34 seg = 5134 seg.

Guardar con el nombre **'hora.py'** y comprobar que funciona con cualquier hora que no sea 0.

13) Realizar un programa que pida que adivinemos un numero del 1 al 100 y que trate de acertarlo. En caso de acertar que muestre por pantalla "Enhorabuena, lo has acertado" y si no lo adivina que ponga:

- Si es mayor que él numero a acertar: "Un poco menor, vuelve a intentarlo"
- Si es menor que él numero a acertar: "Un poco mayor, vuelve a intentarlo"

Guardar con el nombre **'adivina.py**' y comprobar que funciona correctamente.

14. Copiar el siguiente programa y añadir comentarios que indiquen qué se está haciendo en cada caso. Guardarlo con el nombre **"masComparaciones.py**":

```
x=int(input("Introduce un numero x:"))
y=int(input("Introduce un numero y:"))

if x>2 and x <=10:
    print("x es mayor que dos y menor o igual que diez ")
if x>2 or x <=20:
    print("x es mayor que dos o menor o igual que veinte")
if (not(x==y)):
    print("x no es igual que y")
```

Práctica 7 - Bucles For y While

Hasta ahora los programas que hemos visto eran lineales. Comenzaban por la primera instrucción y acababan por la última, ejecutándose todas una sola vez. Lógico ¿no?. Pero resulta que muchas veces no es esto lo que queremos que ocurra. Lo que nos suele interesar es que dependiendo de los valores de los datos se ejecuten unas instrucciones y no otras (condicionales). O también puede que queramos repetir unas instrucciones un número determinado de veces (bucles). Para esto están las sentencias de control de flujo.

Los bucles nos ofrecen la solución cuando queremos repetir una tarea un número determinado de veces. Supongamos que queremos escribir 1000 veces la palabra *hola*. Con lo que sabemos hasta ahora haríamos:

```
print("hola")
print("hola")
print("hola")
print("hola")
print("hola")
print("hola")
# ......

# y así hasta 1000 veces
```

¡Menuda locura! Y si queremos repetirlo más veces nos quedaría un programa sin fin. Para ocasiones como esta utilizamos los bucles, que son repeticiones de una parte del programa, donde especificamos cuando queremos que empiece y cuando acabe el mismo.

El bucle For

En general, un bucle es una estructura de control que repite un bloque de instrucciones. Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

La sintaxis de un bucle for es la siguiente:

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

No es necesario definir la variable de control antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa.

El programa anterior, usando un bucle *for* el programa quedaría:

```
for i in range(1000):
    print("hola")
```

1) Probad a ejecutar este programa para ver el efecto creado. Guardarlo con el nombre **“mibucle.py”**

2) A continuación se acompañan algunos ejemplos de utilización del bucle For. Copiarlos y probarlos en un nuevo programa llamado **"ejemplosFor.py"**

```
# Ejemplo de bucle 1
print("Comienzo bucle 1")
for i in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")

# Ejemplo de bucle 2
print("Comienzo bucle 2")
for i in []:
    print("Hola ", end="")
print()
print("Final")

# Ejemplo de bucle 3
print("Comienzo bucle 3")
for i in [1, 1, 1]:
    print("Hola ", end="")
print()
print("Final")

# Ejemplo de bucle 4
print("Comienzo bucle 4")
for _ in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")

# Ejemplo de bucle 5
print("Comienzo bucle 5")
for i in [3, 4, 5]:
    print(f"Hola. Ahora i vale {i} y su cuadrado {i ** 2}")
print("Final")

# Ejemplo de bucle 6
print("Comienzo bucle 6")
for i in ["Alba", "Benito", 27]:
    print(f"Hola. Ahora i vale {i}")
print("Final")

#Ejemplo de bucle 7
print("Comienzo bucle 7")
for numero in  [0, 1, 2, 3]:
    print(f"{numero} * {numero} = {numero ** 2}")
print("Final")
```

3) Crear un programa que nos muestre por pantalla los números del 1 al 1000 uno en cada línea. Guardar el programa con el nombre **'bucleFor.py'** y comprobar que funciona correctamente

Pista: La variable que se incrementa es la que se debe mostrar por pantalla.

Nº 1
Nº 2
Nº 3
……
Nº 1000

El bucle While

Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True).
La sintaxis del bucle while es la siguiente:

while condicion:
    cuerpo del bucle

La ejecución de esta estructura de control while es la siguiente:

Python evalúa la condición:

- si el resultado es True se ejecuta el cuerpo del bucle. Una vez ejecutado el cuerpo del bucle, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del bucle) una y otra vez mientras la condición sea cierta.
- si el resultado es False, el cuerpo del bucle no se ejecuta y continúa la ejecución del resto del programa.

La variable o las variables que aparezcan en la condición se suelen llamar variables de control. Las variables de control deben definirse antes del bucle while y modificarse en el bucle while.

Veamos esto con un ejemplo:

```
contador=0

while(contador<100):
    contador = contador + 1
    print("Ya voy por el ", contador, "pararé enseguida")
```

Este programa imprime en pantalla los valores del 1 al 100. Cuando contador=100 ya no se cumple la condición

4) Copiar el programa anterior y añadir comentarios a cada una de las líneas del mismo indicando qué se está haciendo en cada momento. Guardarlo con el nombre **'miWhile.py'** dentro de la carpeta PROGSPYTHON

5) A continuación se acompañan algunos ejemplos de utilización del bucle For. Copiarlos y probarlos en un nuevo programa llamado **"ejemplosWhile.py"**

```
print("Ejemplo de bucle while 1")
i = 1
while i <= 3:
    print(i)
    i += 1
print("Programa terminado")

print("Ejemplo de bucle while 2")
i = 1
while i <= 50:
    print(i)
    i = 3 * i + 1
print("Programa terminado")

print("Ejemplo de bucle while 3")
numero = int(input("Escriba un número positivo: "))
while numero < 0:
    print("¡Ha escrito un número negativo! Inténtelo de nuevo")
    numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")
```

6) Abre de nuevo el programa "**adivina.py**" que hiciste en la práctica anterior y agrega un bucle **while** para que este programa sólo acabe cuando el usuario acierte, mientras tanto debe seguir pidiendo números hasta acertar.

En caso de acertar que muestre por pantalla "Enhorabuena, lo has acertado" y si no lo adivina que ponga:

- Si es mayor que él numero a acertar: "Un poco menor, vuelve a intentarlo"
- Si es menor que él numero a acertar: "Un poco mayor, vuelve a intentarlo"

Guardar con el nombre **'adivina2.py**' y comprobar que funciona correctamente.

Práctica 8 – Ejercicios de repaso (1)

En esta práctica vamos a repasar todo lo visto hasta ahora por medio de ejercicios. En ellos ya no se os dan pistas, debéis ser vosotros los que descubráis la manera de resolverlos. Para ello deberéis repasar la sintaxis vista en prácticas pasadas.

→ 1) Crear un programa que imprima por pantalla el mensaje "Primer programa de repaso".

Guardarlo con el nombre **'primerRepaso.py'** y comprobar que funciona correctamente.

→ 2) Escribe y compila un programa que lea un carácter y lo muestre por pantalla. Después debe hacer lo mismo con un entero y un real.

Guardarlo con el nombre **'porPantalla.py'** y comprobar que funciona correctamente.

→ 3) Escribe un programa que imprima un mensaje de presentación, te pregunte tu año de nacimiento y te lo muestre por pantalla.

Guardarlo con el nombre **'añoNacimiento.py'** y comprobar que funciona correctamente.

→ 4) Crear un programa que realice lo siguiente:

- pida al usuario su edad
- la muestre por pantalla
- le muestre les años que faltan para llegar a los 100 años.

Guardarlo con el nombre **'edad.py'** y comprobar que funciona correctamente.

→ 5) Crear un programa nos permita introducir una cantidad en euros y nos la devuelva en pesetas.

Guardarlo con el nombre '**conversor.py'** y comprobar que funciona correctamente.

→ 6) Escribe un programa que calcule el área de un triángulo rectángulo, dada la altura y la base. Pasos a seguir:

- Pedir al usuario que introduzca la base y la altura.
- Calcular el área mediante la fórmula Área = (Base * Altura) / 2

Guardarlo con el nombre '**triangulo.py'** y comprobar que funciona correctamente.

7) Crear un programa que pida al usuario un carácter y un número de repeticiones. Luego tiene que imprimir dicho carácter el número de repeticiones especificado.

Guardarlo con el nombre **'repeticiones.py'** y comprobar que funciona correctamente.

---

8) Realizar un programa que muestre las tablas de multiplicar del 1 hasta el 10, realizando una parada entre la visualización de cada tabla. Para ello deberás realizar dos bucles anidados (uno dentro de otro).

Guárdalo con el nombre **'tablas.py'** y comprueba que funciona correctamente.

---

9) Escribe un programa que imprima las cuatro primeras potencias de los números del 1 al 10. Ejemplo:

2 → 2 4 8 16
3 → 3 9 27 81

.....

Guardarlo con el nombre **'potencias.py'** y comprobar que funciona correctamente.

Práctica 9 – Funciones

Introducción a las funciones

Vamos a dar un paso más en la complejidad de nuestros programas. A partir de ahora empezaremos a usar funciones creadas por nosotros. Las funciones son de una gran utilidad en los programas. Nos ayudan a que sean más legibles y más cortos. Con ellas estructuraremos mejor los programas.

Las funciones nos sirven para evitar tener que escribir el mismo código varias veces. Ya hemos visto en las prácticas algunas funciones como print( ), input( ) y type( ). Estas funciones ya están definidas para nosotros en Python.

Sin embargo nosotros también podemos definir nuestras propias funciones. Pocas veces se ve un programa un poco complejo que no use funciones. Hasta ahora todos los programas que hemos hecho tenían una función principal.

Definición de una función

Las funciones se pueden crear en cualquier punto de un programa, escribiendo su definición. La primera línea de la definición de una función contiene:

- la palabra reservada **def**
- el nombre de la función (la guía de estilo de Python recomienda escribir todos los caracteres en minúsculas separando las palabras por guiones bajos)
- paréntesis (que pueden incluir los argumentos de la función, como se explica más adelante)
- Al final ponemos :

Las instrucciones que forman la función se escriben con sangría con respecto a la primera línea.

Por comodidad, se puede indicar el final de la función con la palabra reservada return (más adelante se explica el uso de esta palabra reservada), aunque no es obligatorio.

Para poder utilizar una función en un programa se tiene que haber definido antes. Por ello, normalmente las definiciones de las funciones se suelen escribir al principio de los programas

→ 1) Vamos a crear un ejemplo de función donde tenemos un programa que contiene una función y el resultado de la ejecución de ese programa. Esta función no tiene argumentos así que no devuelve nada. Guardarlo con el nombre **'funcion.py'** y comprobar que realiza lo esperado.

Este programa llama a la función **licencia( )** que muestra el mensaje de la licencia. Por supuesto es de nula utilidad pero nos sirve para empezar.

```
def licencia():
    print("Copyright 2020 IES Poeta Paco Mollá")
    print("Licencia CC-BY-SA 4.0")
    return

print("Este programa no hace nada interesante.")
licencia()
print("Programa terminado.")
```

2) Crear un programa similar al anterior pero que tenga dos funciones llamadas **escribe_numeros( ) y escribe_letras( ).** La primera función debe escribir los números del 1 al 100 (usando un bucle). La segunda función debe escribir las letras de la a hasta la h. Desde el programa principal se debe hacer la llamada a estas dos funciones.

Guardarlo con el nombre **'funciones.py'** y comprobar que realiza lo esperado.

Vida de una variable

Cuando definimos una variable dentro de una función, esa variable sólo es válida dentro de la función. Si por el contrario la definimos como variable global, antes de las funciones, se puede usar en todas las funciones.

3) Copiar el siguiente programa y comprobar la diferencia entre variable local y global. Guardar el programa con el nombre "**variableLocal.py**"

```
# Ejemplo de variable local 1

def subrutina():
    a = 2
    print(a)
    return

a = 5
subrutina()
print(a)
```

4) Copiar el siguiente programa y comprobar cómo se pasan y devuelven valores a una función. Guardar el programa con el nombre "**pasoVariables.py**"

```
def mifuncion(numero):
    print(numero)
    doble=2*numero
    return doble

valor=int(input("Introduce un número:"))
print("Ahora vamos a sacar su doble:")
resul=mifuncion(valor)
print(resul)
```

5) Copiar el siguiente programa y comprobar cómo se pasan y devuelven valores a una función. Guardar el programa con el nombre "**pasoVariables2.py**"

```
def escribe_media(x, y):
    media = (x + y) / 2
    print(f"La media de {x} y {y} es: {media}")
    return

a = 3
b = 5
escribe_media(a, b)
print("Programa terminado")
```

6) Copiar el siguiente programa y comprobar cómo se pasan y devuelven valores a una función. Guardar el programa con el nombre "**pasoVariables3.py**". Compara su funcionamiento con el programa anterior.

```
def calcula_media(x, y):
    resultado = (x + y) / 2
    return resultado

a = 3
b = 5
media = calcula_media(a, b)
print(f"La media de {a} y {b} es: {media}")
print("Programa terminado")
```

7) Abre el programa "**triangulo.py**" y modifícalo para que le pasemos la base y la altura del triángulo y devuelva el área. Guardarlo con el nombre "**triangulo2.py**"

8) En el siguiente programa se calcula el mayor de dos números que se le pasan a una función. Guárdalo con el nombre "**compara.py**" y comprueba que hace lo esperado.

```
def mayor(a,b ):        # Metemos los parámetros a y b a la función
    if a>b:
         return a
    elif a<b:
        return b
    else:
        print("Son iguales")

num1=int(input( "Introduce el primer número: " ))
num2=int(input( "Introduce el primer número: " ))
resul=mayor(num1,num2)
print("El mayor es:",resul)
```

9) Añade comentarios al programa anterior explicando qué se está haciendo en cada línea del programa. Guardar el programa resultante con el nombre "**compara2.py**"

10) Crear un programa similar al anterior pero que tenga una función llamada **muestra_suma( ).** A esta función se le pasarán dos valores en su llamada desde el programa principal devolverá la suma final, que procederemos a mostrar al final. Guardarlo con el nombre **'muestra.py'** y comprobar que realiza lo esperado.

11) Realiza un programa que nos sirva de calculadora. Este programa debe realizar lo siguiente:

- Debe tener cuatro funciones: **suma( ), resta( ), multiplica( )** y **divide( ).**
- En el programa principal se debe pedir al usuario que introduzca 2 números.
- A continuación debe pedir al usuario que introduzca la operación a realizar (ej. **Pulsa '1' para sumar, '2' para restar, '3' para multiplicar....)**
- Dependiendo de lo que introduzca el usuario, se debe decidir **qué función utilizar: suma( ), resta( ), etc** dependiendo de la opción elegida y devolver el resultado obtenido.
- Tener en cuenta que la **división puede devolver valores reales**.

12) Guardar el programa con el nombre **'calculadora.py'** y comprobar que funciona correctamente.

Práctica 10 – Módulos o librerías

¿Qué es un módulo?

Un módulo es un fichero que contiene código `PYTHON`. Su extensión es `.py`. Almacena declaración de variables o implementación de funciones.

Tenemos la posibilidad de hacer referencia a otros módulos (mediante la instrucción `import`). Su función principal es reaprovechar y organizar bien el código. Ejemplos de módulos:

```
"""Modulo para el calculo de areas de formas basicas"""
pi = 3.1416


def cuadrado(lado):
    """Calcula el area del cuadrado a partir de su lado"""
    return lado ** 2


def circulo(radio):
    """Calcula el area del circulo dado el radio"""
    return pi * radio ** 2

print 'Area cuadrado =', cuadrado(2)
print 'Area circulo =', circulo(1)
```

Los módulos pueden ser de tres tipos:

- Módulos integrados ya en Python
- Nuestros propios módulos
- Módulos creados por terceros

Pueden ser "llamados" por otros módulos o desde la consola de Python mediante la instrucción **import archivo.py (SIN PONER LA EXTENSIÓN)**

Módulos integrados ya en Python

1) Copiar el siguiente programa y probar lo que hace. Guardarlo con el nombre "**dateTime.py**".

```
import datetime

print(datetime.date.today())
print(datetime.date.weekday(datetime.date.today()))
print(datetime.timedelta(minutes=70))
```

2) Añade comentarios al programa anterior explicando qué se está haciendo en cada línea del programa. Guardar el programa resultante con el nombre "**dateTime2.py**"

3) Copia ahora el siguiente programa y observa las diferencias con respecto al programa anterior. Guardar el programa resultante con el nombre "**dateTime3.py**"

```
from datetime import timedelta, date

print(date.today())
print(date.weekday(date.today()))
print(timedelta(minutes=70))
```

4) Visitar la siguiente página y probar a crear un nuevo programa llamado "**usoModulos.py**" donde importéis algún módulo integrado en Python de los que podéis encontrar en esta página: https://docs.python.org/3/py-modindex.html

Crear nuestros propios módulos

De igual forma podemos proceder a crear nuestros propios módulos que podrán ser reaprovechados más adelante en otros programas que realicemos.

5) Copiar el siguiente programa y probar lo que hace. Guardarlo con el nombre "**funcionMat.py**".

```
def sumar(n1, n2):
    print(n1+n2)

def restar(n1,n2):
    print(n1-n2)
```

6) Abrir el programa "**dateTime.py**" y a continuación, copiar el siguiente programa donde vamos a reutilizar el código creado. Guardarlo con el nombre "**reutilizar.py**".

```
import datetime
import funcionMat

print(datetime.date.today())
print(datetime.date.weekday(datetime.date.today()))
print(datetime.timedelta(minutes=70))

funcionMat.sumar(2,5)
funcionMat.restar(3,8)
```

7) También podemos importar directamente las funciones de nuestro módulo tal y como hicimos en el ejercicio 3). Abrir el programa "**DateTime3.py**" y añadir el siguiente código:

```
from datetime import timedelta, date
from funcionMat import sumar, restar

print(date.today())
print(date.weekday(date.today()))
print(timedelta(minutes=70))

sumar(3,4)
restar(4,9)
```

Guardarlo todo con el nombre "**reutilizar2.py**"

Módulos creados por terceros

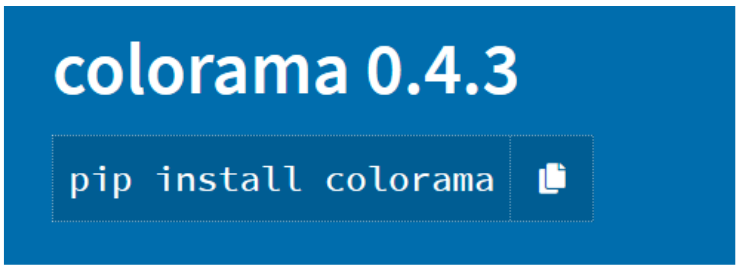
Python es uno de los lenguajes de programación actuales que cuenta con más soporte y desarrollo por parte de las comunidades de programadores.

8) Visitarla siguiente página y probar a crear un nuevo programa donde vamos a importar un módulo en Python de los que ya han sido desarrollados desde esta página: https://pypi.org/

Buscar el proyecto colorama:



A continuación instalarlo desde la consola escribiendo **pip install colorama**:



A partir de ese momento ya podemos reutilizar todo el código de colorama.

Copiar por ejemplo el siguiente código que sirve para poner el texto en colores:

```
from colorama import Fore, Back, Style
print(Fore.RED + 'texto rojo')
print(Back.GREEN + 'con un fondo verde')
print(Style.DIM + 'y con un texto claro')
print(Style.RESET_ALL)
print('volemos a dejarlo todo normal')
```

Guardarlo con el nombre "**modulosTerceros.py**" y probarlo para ver cómo funciona.

9) Probar a buscar en la página https://pypi.org/ los siguientes proyectos creados por terceros que podemos reutilizar en nuestros programas:



Framework para crear aplicaciones web



Framework avanzado para la creación de aplicaciones web

10) Probar a buscar en la página https://pypi.org/ otros proyectos creados por terceros que podamos reutilizar en nuestros programas. Instalarlos y ver cómo funcionan.

Guardarlo con el nombre "**modulosTerceros2.py**" y probarlo para ver cómo funciona.

Práctica 11 – Ejercicios de repaso (2)

En esta segunda práctica de ejercicios de repaso vamos a realizar más ejercicios sobre todo lo visto hasta ahora. Para resolverlos deberéis repasar la sintaxis vista en prácticas anteriores.

---

1) Recupera el programa **'calculadora.py'** y establece las siguientes condiciones:

- No se puede dividir entre 0 (en caso que se introduzca un 0 en el divisor nos salga un error por pantalla).
- La resta no puede dar números negativos, es decir, el primer operador debe ser siempre mayor que el segundo y sino que salga un error por pantalla.
- En caso que se introduzca una opción no válida a la hora de elegir la operación (ej 1 para suma, 2 para resta y se introduce un 8) que salga un error por pantalla diciendo que lo rectifiquemos.

Guardar el programa con el nombre "**repaso1.py**"

---

2) Abre el programa "**conversor.py"** que nos permitía una cantidad en euros y nos la devolvía en pesetas.

Ahora debes modificarlo para que pueda hacer también el paso contrario, es decir que pueda pasar de ptas. a euros y viceversa. Para ello el usuario debe elegir la opción para convertir (ej. Pulsa un '1' para pasar de ptas. a euros y un '2' para pasar de euros a pesetas).

Guardar el programa con el nombre "**repaso2.py**"

---

3) Crear un programa que halle el menor de tres números pedidos al usuario y lo muestre por pantalla. Se debe utilizar para ello una función y seguir estos pasos:

- Pedir al usuario que introduzca tres números.
- Averiguar cuál es el mayor de los tres (utilizando el condicional if)
- Mostrar por pantalla el menor de los 3.

Guardar el programa con el nombre "**repaso3.py**"

4) Crear un programa que sume los 100 primeros números naturales. Para ello deberéis el utilizar un bucle for y emplear únicamente dos variables (una hará de contador y la otra de acumulador).

Guardar el programa con el nombre "**repaso4.py**"

---

5) Escribe un programa que calcule el cuadrado y el cubo de cualquier número que le introduzcamos, pero utilizando para ello funciones. Pasos a seguir:

- Definir una función que tomará como argumento el valor a calcular, ejemplo: def cuadrado( ) y def cubo( )
- Dichas funciones calcularán el resultado y lo devolverán al programa principal
- El programa principal mostrará por pantalla el resultado

Guardar el programa con el nombre "**repaso5.py**"

---

6) Escribe un programa que calcule la media aritmética de 5 números introducidos por el usuario y muestre el resultado por pantalla. Debe hacerse utilizando una función y seis variables.

Guardar el programa con el nombre "**repaso6.py**"

---

7) Escribir un programa donde el usuario deberá introducir un número y que realice las siguientes operaciones:

- Si el número es cero, visualizar en pantalla su valor escrito con letras
- Si es distinto de cero:
  - Si es positivo obtener y mostrar ese número elevado al cuadrado
  - Si es negativo obtener y mostrar ese número elevado al cubo

Guardar el programa con el nombre "**repaso7.py**"

---

8) Crear un programa que permita realiza la conversión de grados Centígrados y grados Fahrenhet, sabiendo que:

- El valor de los grados Fahrenheit estará comprendido entre 0º y 300º F
- La conversión debe seguir la regla $^{o}C = \frac{5*(^{o}F - 32)}{9}$ siendo ºC la temperatura en grados Centígrados y Fº en grados Fahrenheit.
- El usuario introducirá grados Fahrenheit y obtendrá el resultado en grados Centígrados.

Guardar el programa con el nombre "**repaso8.py**"

9) Realizar un programa que lea un número del teclado y escriba en pantalla, en forma de cuadrado, la lista de números naturales desde el 1 hasta el número dado, rellenando los espacios sobrantes con 0.

Ejemplo, para el número 4:

```
1 2 3 4
2 3 4 0
3 4 0 0
4 0 0 0
```

Guardar el programa con el nombre "**repaso9.py**"

---

10) Realizar un programa que pida una hora determinada (con el formato horas, minutos y segundos) y la pase en segundos, por ejemplo: 1 h 25 min 34 seg = 5134 seg.

Guardar el programa con el nombre "**repaso10.py**"

---

11) Realizar un programa que reciba una cantidad en horas, minutos y segundos y la devuelva incrementada en un segundo. Ej. si recibe 54h 55m 59s devolverá 54h 56m 00s

Guardar el programa con el nombre "**repaso11.py**"

---

12) Realizar un programa que lea números entre 1 y 5 (los que no estén dentro de estos límites se considerarán erróneos) e indicar cuál es el número introducido más veces y si alguno de ellos no se ha introducido ninguna vez.
Cada vez que se pida un número se interrogará al usuario sobre si desea (S/N) seguir introduciendo números. Cuando se conteste N se terminará el programa.

Guardar el programa con el nombre "**repaso12.py**"

---

13) Realizar un programa que lea pares de números hasta que se de un 0, el primer número de cada par corresponde a la cantidad de unidades vendidas de un producto y el segundo al precio por unidad de ese producto, y diga cuál ha sido la mayor venta (precio x unidades).

Guardar el programa con el nombre "**repaso13.py**"

---

14) Diseñar un programa que lea los coeficientes de una ecuación de segundo grado $ax^2+bx+c=0$ y calcule sus dos soluciones. Se supone que la ecuación tiene soluciones reales.

Guardar el programa con el nombre "**repaso14.py**"

---

15) Escribir un programa en Python que nos calcule el cambio que debe dar la caja de un supermercado: Dado un precio y una cantidad de dinero, el programa nos dirá cuántas monedas deben darse como cambio de tal forma que el número total de monedas sea mínimo.

Guardar el programa con el nombre "**repaso15.py**"

---

16) El precio final de un producto para un comprador es la suma total del costo del producto, un porcentaje de beneficios que obtiene el vendedor y el I.V.A. Diseñar un algoritmo para obtener el precio final de un producto sabiendo su costo, el porcentaje de beneficios y el I.V.A. aplicable.

Guardar el programa con el nombre "**repaso16.py**"

---

17) Diseñar un programa que permita calcular si un año es bisiesto o no. Un año es bisiesto si es divisible por 4 pero no por 100, o bien es divisible por 400

bisiesto=((year%4 ==0) and (year % 100!=0)) or (year % 400 == 0)

Guardar el programa con el nombre "**repaso17.py**"

---

18) Realiza un programa principal que lea tres números enteros por teclado, los almacene en tres variables (***x***, ***y***, ***z***) y llame a una función tal y como ***maximo()***, de tres argumentos, que devuelva el máximo de estos tres valores.

Guardar el programa con el nombre "**repaso18.py**"

---

19) Realiza un programa que calcule las potencias de la 2 a la 10 del número PI y la raíz cuadrada de dicha potencia. Para ello construye una función que calcule la potencia n-ésima de un número cualquiera (busca algún módulo en Internet que la incluya).

Guardar el programa con el nombre "**repaso19.py**"

---

20) Realizar una función que reciba una cantidad de horas, minutos y segundos y la devuelva en segundos.

Guardar el programa con el nombre "**repaso20.py**"

---

21) Realizar una función que reciba una variable de tipo caracter y si es una letra minúscula la devuelva convertida en mayúsculas y viceversa.

Guardar el programa con el nombre "**repaso21.py**"

22) Realizar el cálculo aproximado del valor del número e con una determinada precisión, siendo dicha precisión un dato de entrada Este número se obtiene mediante la siguiente serie:
e = 1 + 1 / 1! + 1/ 2! + 1/3! +1/4! + …

Guardar el programa con el nombre "**repaso22.py**"

---

23) Realizar un programa que calcule el número de días que llevamos el año a partir de un día y un mes

Guardar el programa con el nombre "**repaso23.py**"

---

24) Generar los N primero términos de la serie de Fibonacci. El valor de N (entero y positivo) deberá ser leído por teclado. En esta serie los dos primeros números son 1 y el resto se obtiene sumando los 2 anteriores, ejemplo: 1,1,2,3,5,8,13,21...

Guardar el programa con el nombre "**repaso24.py**"

---

25) Realizar un programa que calcule las combinaciones que se pueden hacer con m elementos tomados de n en n. Donde n y m son números leídos por teclado. La formula para calcularlo es m!/(n!*(m-n)!)

Guardar el programa con el nombre "**repaso25.py**"

---

26) Realizar un programa que ofrezca 3 oportunidades para acertar una contraseña. (La contraseña está incluida en el código fuente del programa).

Guardar el programa con el nombre "**repaso26.py**"

---

27) Realizar un programa que lea 3 nombres y los ordene alfabéticamente

Guardar el programa con el nombre "**repaso27.py**"

---

28) Realizar un programa que lea un número y diga si es o no capicúa.

Guardar el programa con el nombre "**repaso28.py**"

---

29) Determinar la hipotenusa de un triángulo rectángulo conocidas las longitudes de sus dos catetos.

PISTA: utilizar la función sqrt( ) del módulo math

Guardar el programa con el nombre "**repaso29.py**"