

LABORATORIO FUNDAMENTOS DE LA PROGRAMACIÓN

PRIMEROS PROGRAMAS



Primeros Programas

En todos los programas que realicemos nos acostumbraremos a rellenar el siguiente encabezado:

```
# Archivo: XXXX_XX.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Enunciado del programa
# Llamadas a librerías
# Constantes
# Definición de funciones
# Programa principal
```

1. Escribir por pantalla el siguiente texto “Este es mi primer programa”, guardarlo con el nombre de ejercicio0_1.py.

```
# Archivo: ejercicio0_1.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Escribe por pantalla un texto dado.

# Programa principal
print ('Este es mi primer programa')
```

2. Escribir un número entero para leer el valor de una variable desde el teclado. Guardarlo con el nombre ejercicio0_2.py.

```
# Archivo: ejercicio0_2.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Escribe un número entero para leer el
# valor de una variable desde el teclado.

# Programa principal
dato = int(input('Escribe un numero entero:'))
print ('El numero escrito es:', dato)
```

3. **Modificar el programa anterior incluyendo al final una nueva sentencia que muestre el valor de la variable dato multiplicada por 2.**

```
# Archivo: ejercicio0_3.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar un programa que imprima por
# pantalla el doble de un número introducido por
# teclado.

# Programa principal
dato = int(input('Escribe un numero entero:'))
print ('El doble del numero escrito es:', 2*dato)
```

4. **Modificar el programa ejercicio0_2.py para que se lean 2 números enteros.**

```
# Archivo: ejercicio0_4.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar un programa que lea dos
# números enteros.

# Programa principal
dato1 = int(input('Escribe un número entero:'))
dato2 = int(input('Escribe otro número entero:'))
print ('Los números escritos son:', dato1, dato2)
```

5. **Realizar un programa que lea dos números reales. Probar el programa con dos datos**

NOTA: La sintaxis de lectura de un float parte_entera.parte_decimal.

```
# Archivo: ejercicio0_5.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar un programa que lea dos
# números reales.

# Programa principal
dato1 = float(input('Escribe un numero:'))
dato2 = float(input('Escribe otro numero:'))
print ('Los numeros escritos son:', dato1, dato2)
```

a. Probar el programa con un valor entero y otro valor tipo carácter.

```
# Archivo: ejercicio0_6.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar un programa que lea dos datos
# uno de tipo carácter y otro de tipo entero.

# Programa principal
dato1 = str(input('Escribe un carácter:'))
dato2 = int(input('Escribe un número entero:'))
print ('El carácter escrito es:', dato1, 'y el entero
escrito es:', dato2)
```

b. Cambiar el formato de salida para que se muestren 2 decimales para la variable dato1 y 3 decimales para la variable dato2.

```
# Archivo: ejercicio0_7.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir por pantalla dos números
# reales con 2 y 3 decimales respectivamente.

# Programa principal
dato1 = float(input('Escribe un numero:'))
dato2 = float(input('Escribe otro numero:'))
print ('Los numeros escritos son:', round(dato1,2),
round(dato2,3))
```

c. Cambiar el formato de salida para que el espacio ocupado por la variable dato1 sea de 6 caracteres y para la variable dato2 sea de 10 caracteres, y ambos números deben tener sólo un decimal.

```
# Archivo: ejercicio0_8.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir dos datos por pantalla que
# ocupen 6 y 10 espacios respectivamente.

# Programa principal
dato1 = float(input('Escribe un numero:'))
dato2 = float(input('Escribe otro numero:'))
print ('El primer número ocupa 6 caracteres:\n%6.1f\ny
el segundo ocupa 10 caracteres:\n%10.1f'%(dato1,
dato2))
```

- d. **Cambiar el programa para que lea 2 datos de tipo carácter dato1, dato2, y se obtenga la siguiente salida:**

Ejecución 1: ab

Ejecución 2: a b

Ejecución 3: a ↵ b

```
# Archivo: ejercicio0_9.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Leer dos caracteres introducidos por
# teclado e imprimirlos por pantalla juntos, separados
# por un espacio, y separados por un salto de línea.

# Programa principal
d1 = str(input('Escribe un caracter:'))
d2 = str(input('Escribe otro caracter:'))
print ('Los caracteres escritos son:', d1, d2, sep='')
print ('Los caracteres escritos son:', d1, d2)
print ('Los caracteres escritos son:', d1, d2,
sep='\n')
```

UNIVERSIDAD DE ALCALÁ

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN



LABORATORIO

FUNDAMENTOS DE LA PROGRAMACIÓN

NOTA: ESTA PARTE ES SIMPLEMENTE UN REPASO CONCEPTUAL DE LO APRENDIDO EN TEORIA

1. CONTENIDOS SINTÁCTICOS

1.1 INTRODUCCIÓN.

A diferencia de otros lenguajes de programación, Python ofrece una serie de ventajas que hacen mucho más fácil el aprendizaje de la programación.

Las siguientes características hacen que Python sea ideal para iniciarse en la programación:

- Los programas en Python son mucho más compactos que sus equivalentes en lenguajes como C. Está considerado como un lenguaje de muy alto nivel.
- Es un lenguaje orientado a objetos, que combina datos y funcionalidades. Es muy versátil, con lo cual se pueden explicar varios paradigmas.
- Posee una sintaxis muy legible y visual. Para separar bloques en Python hay que tabular hacia adentro, esto ayuda a que todos los que programen en este lenguaje adopten unas notaciones similares y cualquier programa escrito en Python tenga el mismo aspecto.
- Python contiene una gran cantidad de librerías, funciones y tipos de datos incorporados en el propio lenguaje, lo que le hace fácilmente ampliable.
- Python es case sensitive, es decir, distingue entre mayúsculas y minúsculas.

El intérprete de Python es gratuito. Nosotros trabajaremos con el Python 3.4. Hay algunas diferencias en la sintaxis utilizada en esta versión con respecto a versiones anteriores, todas ellas fácilmente solventables.

En Python no es necesario declarar ningún tipo de dato. Lo único que hay que declarar son las llamadas a librerías, nosotros usaremos la librería math que incluye raíces cuadradas, exponenciales, entre otras muchas más cosas. La sintaxis utilizada para invocar a toda una librería es la siguiente:

from nombre librería import *

Un caso concreto → Para invocar a toda la librería math: **from math import ***

El asterisco (*) indica que se llaman a todas las funciones que contiene la librería math, si queremos llamar a una en particular debemos cambiar el asterisco (*) por la función deseada, por ejemplo:

from math import atan

Esta línea llama única y exclusivamente a la función arcotangente (atan). Este es uno de los casos en que la sintaxis ha cambiado de una versión a otra, ya que en la versión más antigua la función se llamaba **arctan** en vez de **atan**.

Después trabajaremos con variables, pero previamente habremos tenido que asignarle un valor, pero como hemos dicho anteriormente no hace falta declararlas, **SÓLO asignar un valor**.

1.2 ESTRUCTURA DE UN PROGRAMA EN PYTHON.

La estructura de un programa en Python, a diferencia de otros lenguajes como Pascal o C/C++, es más simple. No hace falta comenzar con ninguna palabra reservada ni función en particular, tampoco hace falta poner punto y coma (;) al final de cada línea. Simplemente con escribir un par de líneas ya tenemos un programa en Python. Pero, no debemos olvidar que este curso no pretende enseñar un lenguaje concreto (Python) sino programación estructurada, empleando únicamente la sintaxis de Python. Por ello, previamente se recomienda leer las transparencias del tema 1. Este archivo es únicamente un breve resumen.

```
# CABECERA (Simplemente comentarios aclaratorias de
nuestro programa)
#LLAMADA A LIBRERIAS (Si las hubiera)
#CONSTANTES (No existen en Python, las simularemos con
un identificador que comience por mayúscula)
#FUNCIONES Y PROCEDIMIENTOS(Si los hubiera)
# Cuerpo principal del programa (Aquí se definirán todas
las variables globales)
```

Por ejemplo, para escribir el típico programa que te inicia en la programación, sacar por pantalla la frase “Hola mundo” sería de la siguiente forma:

```
# Archivo: ejemplo1.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir por pantalla la frase 'Hola
# mundo'

# Programa principal
print ('Hola mundo')
```

Nos acostumbraremos a poner de encabezado con comentarios el nombre del archivo, el autor, la fecha y la descripción del mismo, con el fin de seguir un mismo patrón en todos los ejercicios.

Trabajaremos directamente con el intérprete de Python (IDLE Python 3.4 GUI – 32 bits).

En Python no es necesario declarar las variables ni las funciones como en otros lenguajes, simplemente hay que asignarle un valor pero **SIN DECLARAR NADA**. El lenguaje interpreta el tipo de valor asignado.

Los programas son un conjunto de instrucciones escritas que realizan una tarea en un lenguaje de programación concreto (es decir son algoritmos, traducidos a un lenguaje de programación concreto)

Estas instrucciones están formadas por dos tipos de palabras:

- **Identificadores** → Son palabras que nosotros elegimos. Con ellas designaremos nuestros programas, funciones, variables, constantes, objetos...
- **Palabras reservadas** → Son las palabras a las que el lenguaje les tiene asignada alguna misión.

A continuación explicaremos los tipos de palabras que existen en Python y las normas por las que se rigen.

1.2.1 IDENTIFICADORES.

Los identificadores son los nombres que se les asignan a las variables. Debe estar formado por letras mayúsculas o minúsculas, dígitos y/o el carácter de subrayado (_), hay normas fundamentales que deben seguir:

- El primer carácter no puede ser un dígito.
- No puede coincidir con una palabra reservada o palabra clave propia del lenguaje.
- No pueden contener más de una palabra, si fuera necesario habría que escribir un guión bajo (_).

Como se estudió en clase teórica, la última versión de Python acepta algunos caracteres especiales como la ñ, la cual no sería admitida en casi ningún lenguaje.

1.2.2 PALABRAS RESERVADAS.

Python como cualquier otro lenguaje de programación posee su propio código con palabras de uso exclusivo para ciertas funciones, a estas palabras se les denomina palabras reservadas o palabras clave.

and	assert	break	class	continue	def
del	elif	else	except	exec	finally
for	from	global	if	import	in
is	lambda	not	or	pass	print
raise	return	try	while	yield	

1.2.3 CUERPO DEL PROGRAMA.

Cabe destacar que para diferenciar los distintos bloques de un programa se utiliza la **indentación**, es decir, la cantidad de espacios o tabulaciones que hay entre el comienzo de la línea y el primer carácter escrito.

1.2.4 COMENTARIOS DE UN PROGRAMA.

Los comentarios son trozos de código muy importante, en Python y en cualquier lenguaje de programación son ignorados por el compilador, y únicamente sirven para mejorar la comprensión del código, pudiendo explicar que se hace en el programa para en un futuro pueda entenderse bien el código.

Hay tres tipos de comentarios:

- Comentario de una línea → Hay que poner al principio de la línea el símbolo de almohadilla (#).

```
# Esto es un comentario de una línea en Python.
```

- Para comentar varias líneas a la vez en Python → basta con seleccionar todo el texto escrito y pulsar **Alt + 3** y se seleccionaran las líneas pero en vez de con un solo símbolo de almohadilla, con dos (##).

```
##Esto es un comentario  
##de varias línea en Python.
```

- Comentario de varias líneas (Docstring) Solo en funciones → Hay que poner al inicio y al final del comentario tres comillas simples (' ' ') . Se utiliza como ayuda para explicar lo que hacen las funciones (esto se estudiará en el siguiente capítulo). Poniendo **help (nombre_funcion)** en el intérprete aparece el comentario (docstring) que explica su función.

```

# Archivo: ejemplo2.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Uso de docstring, comentarios de más de
una línea que sirven de ayuda en funciones.

# Definición de función
def nombre_funcion():

    ''' Esto es un comentario de más de una línea y un
docstring que sirve de ayuda en funciones '''

# INTÉRPRETE
>>> help (nombre_función)
Help on function nombre_funcion in module __main__:

nombre_funcion(x, y)
    Esto es un comentario de más de una línea y un
docstring que sirve de ayuda en funciones

```

1.3 PRIMERAS SENTENCIAS.

ESCRITURA Y LECTURA DE DATOS.

1.3.1 PRINT.

En versiones anteriores **print** era considerada una simple instrucción, pero en las nuevas versiones está considerada como una función (trozo de código que realiza una pequeña tarea) por lo tanto lo que queramos imprimir por pantalla tiene que ponerse entre paréntesis () y comillas simples o dobles (no hay distinción entre las comillas) todo carácter que no sea un número o una variable. Para separar lo que queramos imprimir debemos de poner una coma (,) entre los datos. En pantalla estos datos saldrán separados por un espacio.

```

# Archivo: ejemplo3.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir por pantalla 'Este es mi
primer programa'

# Programa principal
print ('Este es mi primer programa')

# Salida por pantalla:
Este es mi primer programa

```

```
# Archivo: ejemplo4.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir del 1 al 5 por pantalla

# Programa principal
print (1, 2, 3, 4, 5)

# Salida por pantalla:
1 2 3 4 5
```

Con esta función se pueden hacer infinidad de usos:

- **Redondeo de un número real:** Utilizamos la función **round (x, y)** donde ‘x’ corresponde al número que queremos redondear, e ‘y’ corresponde al número de decimales que queremos que tenga.

```
# Archivo: ejemplo5.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir por pantalla dos números
# redondeados, el primero a 2 decimales y el segundo
# a 3.

# Constantes
Dato1 = 1.1234567
Dato2 = 1.1234567

# Programa principal
print (round(Dato1,2), round(Dato2,3))

# Salida por pantalla:
1.12 1.123
```

También hay otra forma para redondear números, ponemos entre los paréntesis ('%.yf'%(x)) donde la ‘x’ corresponde al número que queremos redondear, e ‘y’ corresponde al número de decimales que queremos que tenga. Esto sólo afecta a la salida por pantalla.

```
# Archivo: ejemplo6.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir un número redondeado a 1
# decimal.

# Constante
Dato1 = 1.1234567

# Programa principal
```

```
print ('%.1f'%(Dato1))
```

```
#Salida por pantalla:  
1.1
```

- **Tabulado de caracteres y números:** Número de caracteres que queremos que ocupe una variable. Pondremos un ejemplo con un carácter o cadena e caracteres, y con números de tipos enteros y reales. En este ejemplo vamos a tener 3 tipos de datos, una cadena (s), un entero (d) y un decimal (f). Para que el intérprete sepa qué tipo de dato se va a imprimir, debemos decírselo en la función print, poniendo s, d o f, respectivamente.

```
# Archivo: ejemplo7.py  
# Autor: XXXXX XXXXXXX (nombre y apellidos)  
# Fecha: DD/MM/AAAA  
# Descripción: Imprimir 3 datos por pantalla que  
# ocupen 6, 9 y 8 espacios respectivamente.  
  
# Constantes  
Dato1 = 'hola'                #cadena de datos, string (s)  
Dato2 = 5                     #número entero, int (d)  
Dato3 = 1.123456789          #número decimal, float (f)
```

```
# Programa principal  
print ('%6s%9d%8.1f'%(Dato1, Dato2, Dato3))
```

```
# Salida por pantalla:  
hola          5          1.1
```

```
'''Dato1 ocupa 6 espacios y es de tipo str, Dato2 ocupa  
9 espacios y es de tipo entero, y Dato3 ocupa 8  
espacios, tiene solamente 1 decimal y es de tipo  
float'''
```

- **Separación entre los datos que se imprimen:** Se utiliza la sentencia `sep=''` entre las comillas se pone el carácter con el que queremos separar los datos. Si queremos que se muestren por pantalla los datos juntos debemos escribir la sentencia sin espacios entre las comillas y si queremos que se separen por un intro (↵) tenemos que poner `\n`. Por defecto si no podemos nada se separan los datos por un espacio.

```
# Archivo: ejemplo7.py  
# Autor: XXXXX XXXXXXX (nombre y apellidos)  
# Fecha: DD/MM/AAAA  
# Descripción: Imprimir dos constantes sin espacios,  
# separadas con un espacio, y separadas con un salto  
# de línea.  
  
# Constantes
```

```
dato1 = 'o'
dato2 = 'k'

# Programa principal
print (dato1, dato2, sep='')
print (dato1, dato2)
print (dato1, dato2, sep='\n')
```

```
# Salida por pantalla:
ok
o k
o
k
```

- **Escribir números en notación exponencial.** Se utiliza la siguiente sentencia:

```
# Archivo: ejemplo8.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir por pantalla un número en
# notación exponencial.

# Constante
x = 0.000005

# Programa principal
print("El número en notación exponencial es el
siguiente: %e" % x)

# Salida por pantalla:
El número en notación exponencial es el siguiente:
5.000000e-06
```

1.3.2 INPUT (Entrada de datos).

La función que se encarga de recoger datos es **input()** y dentro de los paréntesis podemos escribir como si fuera un **print**. Esta función se utiliza para asignarle un valor a una variable por teclado.

```
# Archivo: ejemplo9.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir por pantalla el valor de un dato
# introducido por teclado.

# Programa principal
x = input('Introduce un dato: ')
```

```
print('El valor del dato es el siguiente:', x)
```

```
# Salida por pantalla:  
Introduce un dato: 5  
El valor del dato es el siguiente: 5'''
```

En el ejemplo anterior no se especifica el tipo de dato que va a introducir, pero podemos especificarlo de la siguiente forma:

variable = tipo dato(input())

A continuación un ejemplo de los diferentes tipos de datos de entrada:

```
# Archivo: ejemplo10.py  
# Autor: XXXXX XXXXXX (nombre y apellidos)  
# Fecha: DD/MM/AAAA  
# Descripción: Imprimir varios tipos de datos por pantalla  
  
# Programa principal  
x = int(input('Introduce un entero: '))  
y = float(input('Introduce un número decimal: '))  
z = str(input('Introduce una cadena: '))  
print('Los datos introducidos son los siguientes:', x, y, z,  
sep='\n')  
  
# Salida por pantalla:  
Introduce un entero: 5  
Introduce un número decimal: 5.6  
Introduce una cadena: hola mundo  
El dato introducido es el siguiente:  
5  
5.6  
hola mundo
```

1.3.3. INSTRUCCIÓN DE ASIGNACIÓN.

La instrucción de asignación se emplea para asignarle valores a variables o constantes.

➤ Sintaxis :

variable = expresión

- variable → El nombre que se le da a una variable se llama **identificador**.
- expresión → Pueden ser de 3 tipos: la función input() (un dato que se pide por teclado), un número, o un carácter o cadena de caracteres que debe ir entre comillas.

➤ Ejemplo:

```
# Archivo: ejemplo11.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Imprimir varios tipos de datos por pantalla
# separados con un salto de línea (\n).

# Constantes
y = 5.5
z = 'hola'

# Programa principal
x = int(input('Introduce un dato entero: '))
print('Los datos son los siguientes:', x, y, z, sep='\n')
# separamos con intro (↵)

# Salida por pantalla:
Introduce un dato entero: 5
Los datos son los siguientes:
5
5.5
hola
```

ENUNCIADOS EJERCICIOS



TEMA1: CONSTANTES, VARIABLES E INSTRUCCIONES SIMPLES

1. Escribir un programa que imprima un árbol hecho con estrellas, centrándolo respecto a la pantalla:

```
  *
 ***
*****
*****
  *
```

2. Escribir un programa que imprima vuestro nombre, dirección y fecha de nacimiento, primero todo en la misma línea y después cada dato en una línea distinta.
3. Hacer un programa que calcule e imprima el valor de π como: $\pi = \arctan(1)*4$
4. Escribir un programa que imprima los valores del logaritmo neperiano de cinco valores de x.
5. Escribir un programa que sume, reste, multiplique y divida dos números.
6. El área de un cuadrado es igual a la longitud de uno de sus lados al cuadrado.
- Escribir un programa que declare variables para almacenar la longitud del lado y el área del cuadrado. Asignar un valor a la longitud del lado y calcular el área. Visualizar el valor de las dos variables.
 - Repetir el apartado anterior pero asignando un valor al área y calculando la longitud del lado a partir de ésta.
 - Repetir el apartado a) pero considerando que se debe calcular el área lateral de un cubo y su volumen a partir de su arista. Hacer un programa que asigne las variables, haga los cálculos, y visualice de forma adecuada los resultados.
7. Escribir un programa que calcule el área de un triángulo.
8. Escribir un programa que calcule la longitud y el área de una circunferencia.
9. Escribir un programa que:
- Lea tres caracteres y los imprima primero por separado y después formando una palabra.
 - Lea un número entero e imprima su cuadrado.
 - Lea tres números reales e imprima el primero en notación exponencial, el segundo en notación exponencial limitada y el tercero en notación decimal.
10. Escribir un programa que pida una hora (hora, minutos, segundos) y calcule e imprima el total de segundos transcurridos desde la última medianoche.
11. Escribir un programa que pida una temperatura en grados Fahrenheit y calcule su equivalente en grados Centígrados, imprimiendo ambos datos. [$C = (F-32)/1.8$]

12. Escribir un programa que calcule el impuesto que debe pagar un contribuyente a partir de sus ingresos anuales y del número de hijos. El impuesto a pagar es un tercio del ingreso imponible, siendo este último igual a los ingresos totales menos una deducción personal de 6000 EUROS. y una deducción de 60 EUROS. por cada hijo.

13. Escribir un programa que calcule la velocidad de un misil que recorre 25 Km en 4 minutos. Expresar el resultado en metros/segundo.

$$\text{Velocidad} = \text{espacio/tiempo}$$

14. Escribir un programa que calcule el valor de la siguiente expresión:

$$(a+9*c)/(b+4-a)+4*b \quad a = 3, b = 6, c = 4$$

15. Elaborar un programa que calcule el coste total y los kilómetros por litro necesarios para el manejo de un vehículo basándose en los kilómetros recorridos, la gasolina consumida, el coste de la gasolina por litro y otros costes de mantenimiento del camión por kilómetro.

Funciones predefinidas necesarias para afrontar los ejercicios 1ª Parte Tema1

Para realizar los siguientes ejercicios necesitaremos llamar a la librería math ya que necesitamos ciertas funciones matemáticas.

Para llamar a la **librería math entera** la sintaxis es la siguiente:

```
From math import *
```

Si queremos llamar a alguna **función en concreto** se hace de la siguiente forma:

```
From math import nombre_funcion
```