



Módulo 1 – Fundamentos de Programación Python

Funciones y Módulos

Ciencia de Datos

Objetivos



- Aprender Instrucciones básicas de Python.
- Conocer sobre tipos y estructuras de datos, operadores y expresiones.
- Conocer sobre flujos de control.
- Codificar un programa creando funciones.

Contenido

1. Funciones preconstruidas.
2. Funciones personalizadas.
3. Módulos.



Funciones



KIBERNUM

Funciones en Python

¿Prefieres escribir una sola pieza de código 10 veces o sólo una vez y usarla 10 veces?

Reuse:

```
#collect input from user
celsius = float(input("Enter Celsius value:
"))
#calculate value in Fahrenheit
Fahrenheit = (celsius*1.8) + 32
print("Fahrenheit value is ",fahrenheit)
```



Fahrenheit = $(9/5)\text{Celsius} + 32$



```
#collect input from user
celsius = float(input("Enter Celsius
value: "))
#calculate value in Fahrenheit
Fahrenheit = (celsius*1.8) + 32
print("Fahrenheit value is ",fahrenheit)
```



No querrás repetir las mismas
líneas de código cada vez que
necesitas hacer una conversión

Funciones en Python

- ➊ Una **función** es un bloque de código reutilizable que forma la base para realizar acciones en un programa.

Razón para usar la función = REUTILIZACIÓN.

Una vez definida, una función se puede utilizar un número ilimitado de veces en cualquier parte del código, sin la necesidad de escribir nuevamente las instrucciones que la componen.

- ➋ **Tipos de Funciones en Python:**

- Funciones preconstruidas.
- Funciones creadas por usuario.

Funciones Preconstruidas

El intérprete de Python tiene una serie de funciones y tipos incluidos en él que están siempre disponibles. Están listados aquí en orden alfabético:

Built-in Functions

A

`abs()`
`aiter()`
`all()`
`any()`
`anext()`
`ascii()`

B

`bin()`
`bool()`
`breakpoint()`
`bytearray()`
`bytes()`

C

`callable()`
`chr()`
`classmethod()`
`compile()`
`complex()`

D

`delattr()`
`dict()`
`dir()`
`divmod()`

E

`enumerate()`
`eval()`
`exec()`

F

`filter()`
`float()`
`format()`
`frozenset()`

G

`getattr()`
`globals()`

H

`hasattr()`
`hash()`
`help()`
`hex()`

I

`id()`
`input()`
`int()`
`isinstance()`
`issubclass()`
`iter()`

L

`len()`
`list()`
`locals()`

M

`map()`
`max()`
`memoryview()`
`min()`

N

`next()`

O

`object()`
`oct()`
`open()`
`ord()`

P

`pow()`
`print()`
`property()`

R

`range()`
`repr()`
`reversed()`
`round()`

S

`set()`
`setattr()`
`slice()`
`sorted()`

`staticmethod()`
`str()`
`sum()`
`super()`

T

`tuple()`
`type()`

V

`vars()`

Z

`zip()`

`_import_()`

Funciones Preconstruidas

Algunos ejemplos adicionales de funciones preconstruidas:

```
s = 'Buenos dias'  
len(s)
```

11

```
n = -1.567  
round(n)
```

-2

```
round(n,1)
```

-1.6

```
abs(n)
```

1.567

```
abs(round(n))
```

2

Funciones Pre-Construidas

max()

El método max() devuelve el elemento más grande en un elemento iterable o el más grande entre dos (o más) parámetros.

Sintaxis: max(iterable). max(arg1, arg2, *args)

min()

El método min() devuelve el elemento más pequeño en un elemento iterable o el más pequeño entre dos (o más) parámetros.

Sintaxis: min(iterable). min(arg1, arg2, *args)

```
[1] # usando max(arg1, arg2, *args)
print('Máximo es:', max(1,3,2,5,4))
```

```
# usando max(iterable)
num = [1,3,2,8,5,10,6]
print('Máximo es:', max(num))
```

```
→ Máximo es: 5
Máximo es: 10
```

```
[2] # usando min(arg1, arg2, *args)
print('Mínimo es:', min(1,3,2,5,4))
```

```
# usando min(iterable)
num = [1,3,2,8,5,10,6]
print('Mínimo es:', min(num))
```

```
→ Mínimo es: 5
Mínimo es: 10
```

Funciones Definidas por el Usuario

- Funciones que definimos para realizar una determinada tarea específica.
- Ayudan a descomponer un programa grande en segmentos pequeños, lo que hace que el programa sea fácil de entender, mantener y depurar.
- Los programadores que trabajan en un proyecto grande pueden dividir la carga de trabajo haciendo diferentes funciones.



Sintaxis de Funciones

Esta es la estructura para definir una función:

```
def nombre_funcion(argumento1, argumento2, ...):
    # instrucción 1
    # instrucción 2
    # ...
    return
```

Utiliza sangría para indicar bloques y hacer que los códigos sean más legibles (indentación).

Reglas para nombrar una función:

Son las mismas que para nombrar variables.

Inicio:

Letras desde A-Z, a-z o un guión bajo(_)

Resto:

Letras desde A-Z, a-z , guión bajo(_) o dígitos(0-9)



KIBERNUM

Sintaxis de Funciones

Algunos ejemplos:

```
# definimos una función con parámetros de entrada
def saludo(nombre):
    print("Hola " + nombre )
```

```
# utilizamos la función
saludo('María José')
```

```
Hola María José
```

Las funciones que retornan un valor deben usar la palabra reservada **return**.

```
# Función con parámetros y retorno
def cubo(n):
    salida = n**3
    return salida
```

```
cubo(2)
```

Sintaxis de Funciones

Los parámetros de una función pueden ser definidos con valores por defecto. En este ejemplo, si no se especifica el parámetro `debug`, por defecto se utilizará el valor `False`.

Cuando explícitamente se indica el nombre del parámetro durante la invocación, entonces no es necesario utilizar el mismo orden que se utiliza en la definición de la función.

```
def myfunc(x, p=2, debug=False):
    if debug:
        print("Evaluando myfunc para x="+str(x)+" usando p="+str(p))
    return x**p
```

```
myfunc(5)
```

```
25
```

```
myfunc(5, debug=True)
```

```
Evaluando myfunc para x=5 usando p=2
```

```
25
```

```
myfunc(p=3, debug=True, x=7)
```

```
Evaluando myfunc para x=7 usando p=3
```

```
343
```

Sintaxis de Funciones

- El cuerpo de la función debe estar **indentado**, una buena práctica es utilizar 4 espacios de indentación. El siguiente ejemplo ilustra un error lanzado por la consola de un error de indentación.
- De forma opcional, pero altamente recomendado, se puede documentar la descripción acerca de la función. A esto se le conoce como “**docstring**”.

```
# Función mal indentada
def cubo(n):
    salida = n**3
    return salida

File "<tokenize>", line 4
    return salida
^
IndentationError: unindent does not match any outer indentation level

def func1(s):
    """
        Imprime un string 's' e indica cuántos caracteres tiene
    """
    print(s + " tiene " + str(len(s)) + " caracteres")

help(func1)
Help on function func1 in module __main__:

func1(s)
    Imprime un string 's' e indica cuántos caracteres tiene
```

Sintaxis de Funciones

Una función puede retornar múltiples valores, usando para eso, estructuras de tipo tupla.

```
def potencias(x):
    """
    Retorna algunas potencias de x
    """
    return x ** 2, x ** 3, x ** 4

potencias(3)
(9, 27, 81)
```

Una función que retorna tuplas, pueden ser “desempaquetadas” como se aprecia en la siguiente línea de código.

```
x2, x3, x4 = potencias(3)

print(x3)
27
```

Módulos



KIBERNUM

Módulos



Uno de los conceptos más importantes dentro de la programación, es la **reutilización de código** evitando redundancia. La idea es escribir funciones y clases con un propósito y ámbito bien definido, privilegiando la reutilización en vez de repetir código en diferentes partes de un programa (programación modular). El resultado es mejor legibilidad y mantenibilidad del programa, lo cual facilita la detección de bugs y son más fáciles de extender.

Los módulos de Python son definidos en un archivo Python (con extensión `.py`) y puede ser accesible a otros módulos y programas utilizando la sentencia `import`.

Módulos

- El siguiente código crea el módulo **mymodule.py**, el cual puede contener variables, funciones, clases.
- Es recomendable documentar adecuadamente cada módulo, para facilitar su posterior utilización.

```
%%file mymodule.py
"""
Ejemplo de un modulo Python. Contiene una variable llamada my_variable,
una funcion llamada my_function, y una clase llamada MyClass.
"""

my_variable = 0

def my_function():
    """
    Funcion ejemplo
    """
    return my_variable

class MyClass:
    """
    Clase ejemplo
    """

    def __init__(self):
        self.variable = my_variable

    def set_variable(self, new_value):
        """
        setea self.variable a un nuevo valor
        """
        self.variable = new_value

    def get_variable(self):
        return self.variable
```

Writing mymodule.py

Módulos

La Librería Estándar de Python es una colección de módulos accesibles a un programa en Python para **simplificar el proceso de programación** y mover la necesidad de reescribir código comúnmente utilizado. Éstos pueden ser usados realizando la importación del módulo requerido.

Los módulos más importantes son los siguientes:

- time
- sys
- os
- math
- random
- urllib
- re
- statistic

Más información en la documentación oficial:
<https://docs.python.org/3.9/contents.html>

El Módulo Math

En la web oficial de la librería estándar de Python se puede encontrar una referencia del Módulo Math.

The screenshot shows a detailed view of the Python documentation for the `math` module. At the top, there's a navigation bar with links to Python, English, version 3.9.7, the documentation index, the standard library, and numeric/mathematical modules. On the left, a sidebar contains a "Table of Contents" with a hierarchical tree of mathematical functions, including Number-theoretic and representation functions, Power and logarithmic functions, Trigonometric functions, Angular conversion, Hyperbolic functions, Special functions, and Constants. Below this are links to "Previous topic" (`numbers`) and "Next topic" (`cmath`), along with "This Page" options for reporting bugs or viewing source code. The main content area is titled "math — Mathematical functions". It starts with a brief description: "This module provides access to the mathematical functions defined by the C standard." It then explains that these functions are for real numbers only and refer to the `cmath` module for complex numbers. It also notes that most functions return float values. A section titled "Number-theoretic and representation functions" follows, listing the `ceil` and `comb` functions with their descriptions. The URL at the bottom is <https://docs.python.org/3/library/math.html>.

Módulo Statistic

El Módulo Statistics está disponible en la librería estándar de Python y es utilizado para realizar cálculos matemáticos estadísticos sobre datos numéricos.

```
: import statistics  
  
: print(statistics.mean([-11, 5.5, -3.4, 7.1, -9, 22]))  
1.8666666666666667  
  
: print(statistics.stdev([1, 30, 50, 100]))  
41.67633221226008  
  
: print(statistics.median([-11, 5.5, -3.4, 7.1, -9, 22]))  
1.05  
  
: print(statistics.mode(['red', 'green', 'blue', 'red']))  
red
```



Dudas y consultas



KIBERNUM



Fin presentación



KIBERNUM