



Universidad Católica
San Pablo

Funciones

Programación Básica con Python

Graciela Meza Lovon, Yessenia Yari Ramos, Alvaro Mamani Aliaga

Contenido

- ❖ ¿Qué es una función?
- ❖ Funciones en Python
 - Funciones de la Librería Estándar
 - Funciones Integradas
 - Funciones definidas en módulos
 - Funciones definidas por el usuario

¿Qué es una función?

- ❖ Una **función** es un bloque de código organizado y reutilizable que se utiliza para realizar una acción determinada.

Funciones en Python

- ❖ Las funciones en Python pueden ser:
 - Funciones de la Librería Estándar.
 - Definidas por el usuario.

Funciones en Python

- ❖ Las funciones en Python pueden ser:
 - Funciones de la Librería estándar
 - Funciones integradas (*built-in functions*)
 - Funciones definidas en módulos

Funciones Integradas

- ❖ Las funciones en Python pueden ser:
 - Funciones de la Librería estándar
 - Funciones integradas (*built-in functions*): un conjunto de funciones que el intérprete de Python siempre tiene disponibles.
 - Funciones definidas en módulos

Funciones Integradas

❖ Ejemplos ya usados:

➤ type

```
type("Ciudad")
```

```
# devuelve el tipo de "Ciudad", es decir,
```

```
# <type 'string'>
```

➤ print

```
print("Hola Mundo")
```

```
# Imprime en pantalla Hola Mundo
```

➤ ... y otras

■ <https://docs.python.org/3/library/functions.html>

Llamadas a Funciones

- ❖ La expresión `type ("Ciudad")` es una invocación a una función o llamada a una función.
- ❖ Si escribimos:

```
print("Hola Mundo")
```

estamos llamando o invocando a la función `print`

Funciones: Nombre, Argumento y Valor de Retorno

❖ Tres partes importantes de una función:

➤ Nombre de la función

➤ Argumento

➤ Valor de retorno

```
type("Ciudad")
```

```
<type 'string'>
```

Funciones: Nombre, Argumento y Valor de Retorno

❖ Tres partes importantes de una función:

➤ Nombre de la función

➤ Argumento

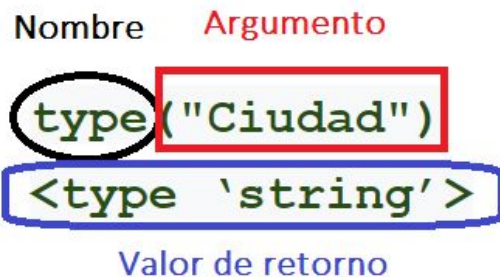
➤ Valor de retorno

Nombre Argumento

```
type("Ciudad")
```

<type 'string'>

Valor de retorno



Funciones Integradas: Conversión de Tipos

❖ Funciones que convierten valores de un tipo a otro.

➤ int

➤ float

➤ str

Funciones Integradas: Conversión de Tipos

❖ Funciones que convierten valores de un tipo a otro.

➤ `int`

```
int(3.99999) # devuelve 3
```

```
int(-2.3)    # devuelve -2
```

```
int(3)       # devuelve 3, el argumento ya es un entero
```

```
int("3")     # devuelve 3, el argumento es una cadena
```

Funciones Integradas: Conversión de Tipos

❖ Funciones que convierten valores de un tipo a otro.

➤ float

```
float(32)          # devuelve 32.0
```

```
float("3.14159") # devuelve 3.14159
```

```
float("hola")      # el intérprete da el siguiente error
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: could not convert string to float: 'hola'
```

Funciones Integradas: Conversión de Tipos

❖ Funciones que convierten valores de un tipo a otro.

➤ `str`

```
str(32)           # devuelve '32'
```

```
str(3.14149)     # devuelve '3.14149'
```

Funciones en Python

- ❖ Las funciones en Python pueden ser:
 - Funciones de la Librería estándar
 - Funciones integradas (*built-in functions*)
 - Funciones definidas en módulos

Funciones definidas en Módulos

❖ Funciones definidas en módulos

- Además las funciones integradas, existen funciones predefinidas disponibles en bibliotecas.
- Estas funciones se definen en módulos.
 - Un módulo es un archivo que contiene definiciones de funciones, clases, variables, constantes u otros objetos de Python.
 - Para usar las funciones de un módulo, debemos primero importar el módulo.

Funciones definidas en Módulos: Funciones Matemáticas

- ❖ Por ejemplo para llamar a una función de la biblioteca matemática debemos importar el módulo **math**, dicha importación debe ser realizada antes de la invocación:

```
import math
```

Funciones definidas en Módulos: Funciones Matemáticas

- ❖ Por ejemplo para llamar a una función de la biblioteca matemática debemos importar el módulo **math**, dicha importación debe ser realizada antes de la invocación:

palabra reservada para
la importación

`import` `math`

nombre del módulo

- ❖ Una lista de la funciones matematicas

➤ <https://docs.python.org/3/library/math.html>

Funciones definidas en Módulos: Funciones Matemáticas

- ❖ Las funciones se llaman usando el formato **notación punto**, es decir, nombre del módulo y el nombre de la función, separados por un punto.
- ❖ Ejemplo: Implemente un programa de convierta grados a radianes

Importa el módulo math

```
import math
```

```
grados = 45
```

```
angulo = grados * 2 * 3.1416 / 360.0
```

```
resultado = math.sin(angulo)
```

```
print(resultado)
```

Llama a la función sin
del módulo math,
usando la notación
punto

Funciones definidas en Módulos: Funciones Matemáticas

- ❖ Las funciones se llaman usando el formato **notación punto**, es decir, nombre del módulo y el nombre de la función, separados por un punto.
- ❖ Ejemplo: Implemente un programa de convierta grados a radianes

```
import math
```

```
grados = 45
```

```
angulo = grados * 2 * 3.1416 / 360.0
```

```
resultado = math.sin(angulo)
```

```
print(resultado)
```

Funciones definidas en Módulos: Funciones Matemáticas

- ❖ Algunas funciones no reciben argumentos, por ejemplo la función `random` del módulo `random`, genera números aleatorios en el rango `[0.0,1.0)`

```
import random
```

```
random.random() #devuelve un número en el rango [0.0,1.0)
```

```
random.random(20)# devuelve el siguiente error
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: random() takes no arguments (1 given)
```

Funciones definidas en Módulos: Funciones Matemáticas

❖ Las funciones pueden recibir más de un argumento

➤ Ejemplo: la función `pow(x,y)`

```
import math
```

```
math.pow(3, 4)          # devuelve 81.0
```

```
math.pow(3.2, 4.1)      # devuelve 117.79176005872732
```

```
math.pow(4, 0.5)        # devuelve 2.0
```

Contenido

- ❖ ¿Qué es una función?
- ❖ Funciones en Python
 - Funciones de la Librería Estándar
 - Funciones Integradas
 - Funciones definidas en módulos
 - Funciones definidas por el usuario

Contenido

❖ Agregando nuevas funciones

- Sintaxis
- Ejemplos
- Ventajas y desventajas
- Definiciones y uso

❖ Parámetros y argumentos

❖ Localidad

❖ Funciones Fructíferas

Funciones definidas por el usuario

- ❖ Las funciones que hemos usado hasta ahora ya han sido definidas para nosotros, pero también es posible **definir nuestras propias funciones**.

Funciones definidas por el usuario

- ❖ Las funciones que hemos usado hasta ahora ya han sido definidas para nosotros, pero también es posible **definir nuestras propias funciones**.
- ❖ La sintaxis para una definición de función es:
 - Nombre: puede escoger el nombre que desee para sus funciones, pero no use una palabra reservada.
 - Lista de parámetros: especifica que información, si es que la hay, se debe proporcionar a fin de usar la función creada.
 - Sentencias: Se puede incluir cualquier número de sentencias dentro de la función. Estas deben estar indentadas.

```
def NOMBRE (LISTA DE PARÁMETROS) :  
    SENTENCIAS
```

Funciones definidas por el usuario

❖ Ejemplo

```
def nuevaLinea():  
    print()  
  
print ("Primera Linea.")  
nuevaLinea()  
print ("Segunda Linea.")
```

Funciones definidas por el usuario

❖ Ejemplo

Definición de la función

```
def nuevaLinea():  
    print()
```

palabra reservada
para la definición de
funciones

nombre de la función

parámetro de la
función (si los hubiera)

Sentencias que
ejecuta la función en
caso de ser llamada

```
print ("Primera Linea.")
```

```
nuevaLinea()
```

```
print ("Segunda Linea.")
```



Llamada a la función nuevaLinea

Funciones definidas por el usuario

❖ Ejemplo

```
def nuevaLinea():  
    print()  
  
print ("Primera Línea.")  
nuevaLinea()  
print ("Segunda Línea.")
```

Salida:
Primera Línea.

Segunda Línea.

Funciones definidas por el usuario

❖ Llamadas repetidas de la misma función

```
print("Primera Línea.")  
nuevaLinea()  
nuevaLinea()  
nuevaLinea()  
print ("Segunda Línea.")
```

Salida:

Primera Línea.

Segunda Línea.

Funciones definidas por el usuario

❖ Llamadas de funciones dentro de funciones

```
def tresLineas():  
    nuevaLinea()  
    nuevaLinea()  
    nuevaLinea()
```

```
print ("Primera Línea.")  
tresLineas()  
print ("Segunda Línea.")
```

Salida:

Primera Línea.

Segunda Línea.

Funciones definidas por el usuario

- ❖ Las definiciones de Funciones se ejecutan como las otras sentencias, pero su efecto es crear nuevas funciones.
- ❖ Las sentencias dentro de la función no se ejecutan hasta que la función es llamada.
- ❖ La definición no genera salida.
- ❖ Se tiene que crear una función antes de ejecutarla, i.e., la definición de función tiene que ejecutarse antes de llamarla por primera vez.

```
def nuevaLinea() :  
    print()  
  
def tresLineas() :  
    nuevaLinea()  
    nuevaLinea()  
    nuevaLinea()  
print ("Primera Línea.")  
tresLineas()  
print ("Segunda Línea.")
```

Parámetros y argumentos

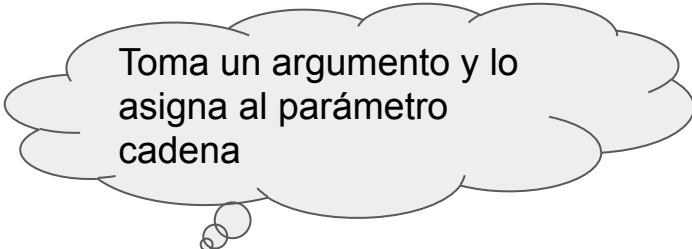
```
def imprimaDoble(cadena):  
    print (cadena, cadena)
```

```
imprimaDoble("Spam")  
imprimaDoble("Spam"*4)
```

Salida:

Spam Spam

SpamSpamSpamSpam SpamSpamSpamSpam



Toma un argumento y lo
asigna al parámetro
cadena

Parámetros y argumentos

- Observe algo muy importante, el nombre de la variable que pasamos como argumento (Spam) no tiene nada que ver con el nombre del parámetro (cadena). No importa como se nombraba el valor originalmente (en el lugar donde se hace el llamado); en la función `imprimaDoble`, la seguimos llamando de la misma manera `cadena`.

Las variables y los parámetros son locales

- ❖ Cuando usted crea una variable local en una función, solamente existe dentro de ella, y no se puede usar por fuera. Lo mismo sucede con los parámetros.
- ❖ Ejemplo

```
def concatenar_doble(partel1, parte2):  
    cat = partel1 + " " + parte2  
    print(cat + " " + cat)
```

```
cancion_1 = "Contigo Perú"  
cancion_2 = "Bello Durmiente"  
concatenar_doble(cancion_1, cancion_2)
```

Salida:

Contigo Perú Bello Durmiente Contigo Perú Bello Durmiente

Las variables y los parámetros son locales

- ❖ Cuando usted crea una variable local en una función, solamente existe dentro de ella, y no se puede usar por fuera. Lo mismo sucede con los parámetros.
- ❖ Ejemplo

```
def concatenar_doble(partel1, parte2):  
    cat = partel1 + " " + parte2  
    print(cat + " " + cat)
```

```
cancion_1 = "Contigo Perú"  
cancion_2 = "Bello Durmiente"  
concatenar_doble(cancion_1, cancion_2)  
print(cat)
```

Salida:

NameError: name 'cat' is not defined

Funciones Fructíferas

- Las funciones fructíferas son aquellas que retornan un valor.
- Dicho valor son retornado usando la sentencia **return**.
- Sintaxis:

```
def nombre_función(lista_de_parámetros):  
    sentencia_1  
    ...  
    sentencia_n  
    return valor_de_retorno
```

Funciones Fructíferas

- Ejemplo: escribir un programa de calcule el área de una circunferencia.

```
def nombre_función(lista_de_parámetros):  
    sentencia_1  
    ...  
    sentencia_n  
    return valor_de_retorno
```

```
import math  
def area(radio):  
    temp = 3.1416 * radio**2  
    return temp
```

Funciones Fructíferas

- Ejemplo: escribir un programa que calcule la distancia entre dos puntos.

```
def distancia(x1, y1, x2, y2):  
    dx = x2 - x1  
    dy = y2 - y1  
    suma_cuadrados = dx**2 + dy**2  
    resultado = math.sqrt(suma_cuadrados)  
    return resultado
```


Funciones Fructíferas

- Se puede tener múltiples sentencias return, ubicadas en ramas distintas de un condicional.
- Ejemplo: escribir un programa de calcule el valor absoluto.

```
def valorAbsoluto(x):  
    if x < 0:  
        return -x    # (-1)*x  
    else:  
        return x
```

Funciones Fructíferas

- Se debe garantizar que toda ruta posible de ejecución del programa llegue a una sentencia return.
- En el ejemplo la función puede terminar sin llegar a return.

```
def valorAbsoluto(x):  
    if x < 0:  
        return -x  
    elif x > 0:  
        return x
```

Funciones Fructíferas

- Se puede llamar a una función dentro de otra.
- Ejemplo, dado el punto central de una circunferencia y un punto perimetral, calcule el área de un círculo.

```
def area2(xc, yc, xp, yp):  
    radio = distancia(xc, yc, xp, yp)  
    resultado = area(radio)  
    return resultado
```

Funciones Fructíferas

- Funciones Booleanas: retornan un valor booleano y son convenientes para ocultar chequeos complicados dentro de funciones.
- Ejemplo, escribir un programa que indique si un número es divisible por otro número:

```
def esDivisible(x, y):  
    if x % y == 0:  
        return True # es cierto  
    else:  
        return False # es falso
```

Funciones Fructíferas

- Otra forma:
 - Ejemplo, escribir un programa que indique si un número es divisible por otro número:

```
def esDivisible(x, y):  
    return x % y == 0
```

¡Gracias!