16-input

May 28, 2020

1 Función input

La función `input()` permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla `Enter (Intro)`

1.1 Ejemplo

[1]:
```
# preguntarle al usuario cómo se llama
print("¿Cómo te llamas?")
# guardar la información en la variable nombre
nombre = input()
# mostrar en pantalla el nombre ingresado por el usuario
print(f"Hola {nombre}, es un gusto saludarte")
```

```
¿Cómo te llamas?
 Alejandra
Hola Alejandra, es un gusto saludarte
```

En el ejemplo anterior, el usuario escribe su respuesta en una línea distinta a la pregunta. Python añade un salto de línea al final de cada `print()`.

Si se prefiere que el usuario escriba su respuesta a continuación de la pregunta, se podría utilizar el argumento opcional `end` en la función `print()`, que indica el carácter o caracteres a utilizar en vez del salto de línea.

[2]:
```
# preguntarle al usuario cómo se llama
print("¿Cómo te llamas? ", end="")
# guardar la información en la variable nombre
nombre = input()
# mostrar en pantalla el nombre ingresado por el usuario
print(f"Hola {nombre}, es un gusto saludarte")
```

```
¿Cómo te llamas?
 Felipe
Hola Felipe, es un gusto saludarte
```

Otra solución, más compacta, es aprovechar que a la función `input()` se le puede enviar un argumento que se escribe en la pantalla (sin añadir un salto de línea)

[3]:
```
# preguntarle al usuario cómo se llama y guardar la información en la variable␣
↪nombre
nombre = input("¿Cómo te llamas? ")
# mostrar en pantalla el nombre ingresado por el usuario
print(f"Hola {nombre}, es un gusto saludarte")
```

```
¿Cómo te llamas?  Ana María
```

```
Hola Ana María, es un gusto saludarte
```

1.2 Conversión de tipos

De forma predeterminada, la función `input()` convierte la entrada en una cadena, aunque escribamos un número. Para convertir el texto realizar *casting*

[4]:
```
# preguntarle al usuario cómo se llama y guardar la información en la variable␣
↪nombre
nombre = input("¿Cómo te llamas? ")
# preguntarle al usuairo su edad. Lo vamos a convertir a int
edad = int(input(f"{nombre}, ¿cuál es tu edad? "))
# mostrar el nombre y la edad del usuario
print(f"Hola {nombre}, tu edad es {edad} año(s)")
```

```
¿Cómo te llamas?  Alexander
Alexander, ¿cuál es tu edad?  26
```

```
Hola Alexander, tu edad es 26 año(s)
```

Validar el tipo de datos para la variables `nombre` y `edad`

[5]:
```
type(nombre)
```

[5]:
```
str
```

[6]:
```
type(edad)
```

[6]:
```
int
```

17-funciones

May 28, 2020

1 Funciones

Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamados cuando se necesite.

Para crear un función en python debemos usar la sentencia `def`. La sintaxis para en función en python sería:

```
def nombre_función(parámetros):
    """documentación de la función"""
    sentencia
    return [expresion]
```

nombre_función: es el nombre que debemos usar cada vez que vamos a usar la función. **parámetros:** es la lista de parámetros que puede recibir una función. **sentencia:** bloque de código que realiza cierta operación. **return:** es la sentencia *return* en código python. **expresión:** es la expresión o variable que devuelve la sentencia return.

1.1 Ejemplo

crear un función que reciba dos parámetros y realice la suma de estos dos datos.

[1]:
```
def sumar(a, b):
    """función creada para realizar la suma de dos (2) números"""
    return a + b
```

Ahora vamos a guardar el resultado de esta operación en una variable para posteriormente mostrar la su valor

[2]:
```
# guardar el resultado de la operación en una variable
operacion = sumar(33, 57)
# mostrar el resultado en la consola
print(f"33 + 57 = {operacion}")
```

```
33 + 57 = 90
```

1.1.1 Parámetros por posición

Cuando enviá argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición

[3]:
```
def suma(a, b):
    return a + b

suma(45, 60)
```

[3]: 105

En el ejemplo anterior el argumento `45` es la posición `0` por consiguiente es el parámetro de las función `a`, seguidamente el argumento `60` es la posición 1 por consiguiente es el parámetro de la función `b`.

1.1.2 Parámetros por nombre

Es posible evadir el orden de los parámetros si indica durante la llamada que valor tiene cada parámetro a partir de su nombre.

[4]:
```
def suma(a, b):
    return a + b

suma(b = 5, a = 45)
```

[4]: 50

Cuando llamamos una función y no le pasamos todos los argumentos provoca una exepción `TypeError`

[5]:
```
suma(4)
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-5-09bad68fb336> in <module>
----> 1 suma(4)


TypeError: suma() missing 1 required positional argument: 'b'
```

[6]:
```
suma()
```

```
␣
↪---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-6-1f6ae598e611> in <module>
----> 1 suma()


TypeError: suma() missing 2 required positional arguments: 'a' and 'b'
```

1.1.3 Parámetros por defecto

Para solucionar la excepción `TypeError` ejecutada al momento de la llamada a una función sin argumentos, podemos asignar unos valores por defecto *nulos* o un *estaticos* a los parámetros.

[7]:
```
def suma(a, b=5):
    return a + b
```

[8]:
```
suma(45, 89)
```

[8]:
```
134
```

Observamos que podemos enviar ambos parámetros y reemplazamos el valor por defecto del parámetro `b`

[9]:
```
suma(45)
```

[9]:
```
50
```

Aquí podemos ver que si no enviamos el parámetro `b` la función no provoca la excepción `TypeError`, ya que por defecto tiene la asignación de `5`

1.1.4 Argumentos indertiminados por posición

En ocasión no sabemos previamente cuantos elementos necesitas enviar a una función. En estos casos puede utilizar los parámetros indeterminados por posición y por nombre.

[10]:
```
def indeterminado_posicion(*args):
    for arg in args:
        print(arg)

indeterminado_posicion("hola", ["manzana", False, "pera", 48.9, "mandarina"],␣
↪(25, 89, 89.0, True, "nombre"), 78.9, False)
```

```
hola
['manzana', False, 'pera', 48.9, 'mandarina']
(25, 89, 89.0, True, 'nombre')
78.9
False
```

1.1.5 Argumentos indeterminados por nombre

Para recibir un número indeterminado de parámetros por nombre (clave-valor o en inglés keyword args), debemos crear un diccionario dinámico de argumentos definiendo el parámetro con dos asteriscos.

[11]:
```
def indeterminado_nombre(**kwargs):
    for kwarg in kwargs:
        print(f"{kwarg} => {kwargs[kwarg]}")

indeterminado_nombre(nombre="Edwin", edad=33, sexo="m", lista=[1, 2, 3],␣
↪tupla=(False, True))
```

```
nombre => Edwin
edad => 33
sexo => m
lista => [1, 2, 3]
tupla => (False, True)
```

18-funciones-anonimas

May 28, 2020

1 Funciones Anónimas o Funciones lambda

Las funciones lambda, a las que también se les conoce como funciones anónimas, es una forma de definir funciones en línea. No es una técnica propia de Python ya que se encuentran disponibles en muchos lenguajes de programación. Al definir las funciones en una línea estas se pueden aplicar a un conjunto de datos y unir posteriormente lo resultados. No siendo necesario asignar un identificador a la función, de ahí el nombre de funciones anónimas.

Sintaxis de función lambda o función anónima en python

`nombre_función = lambda parámetros : sentencia return [expresión]`

nombre_función: es el nombre que debemos usar cada vez que vamos a usar la función. **parámetros:** es la lista de parámetros que puede recibir una función. **sentencia:** bloque de código que realiza cierta operación. **return:** es la sentencia *return* en código python. **expresión:** es la expresión o variable que devuelve la sentencia return.

Se trata de crear funciones de manera rápida, *just in time*, sobre la marcha, para prototipos ligeros que requieren únicamente de una pequeña operación o comprobación. Por lo tanto, toda función lambda también puede expresarse como una convencional (pero no viceversa).

1.1 Ejemplo

[1]:
```
suma = lambda a, b : a + b
```

Ahora llamamos la función

[2]:
```
suma(5, 15)
```

[2]: 20

[3]:
```
mayor = lambda a, b : a if a > b else b
```

[4]:
```
mayor(10, 6)
```

[4]: 10

[5]:
```
mayor(-1, 5)
```

[5]: 5

20200528

May 28, 2020

1 Input

[1]:
```
# Capturar el nombre del usuario
print("Hola, ¿cómo te llamas?")
nombre = input()
print(f"Hola, tu nombre es {nombre}")
```

```
Hola, ¿cómo te llamas?
 Alexander
Hola, tu nombre es Alexander
```

[2]:
```
nombre = input("Hola, ¿cuál es tu nombre? ")
print(f"Hola, tu nombre es {nombre}")
```

```
Hola, ¿cuál es tu nombre?  Felipe
Hola, tu nombre es Felipe
```

[3]:
```
edad = input("Hola, ¿cuál es tu edad?")
type(edad)
nacimiento = edad - 2020
```

```
Hola, ¿cuál es tu edad? 89
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-3-0486c96ff752> in <module>
         1 edad = input("Hola, ¿cuál es tu edad?")
         2 type(edad)
   ----> 3 nacimiento = edad - 2020
```

```
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

[4]:
```
edad = input("Hola, ¿cuál es tu edad?")
type(edad)
```

```
Hola, ¿cuál es tu edad? 89
```

[4]:
```
str
```

[5]:
```
edad = int(edad)
type(edad)
```

[5]:
```
int
```

[6]:
```
nacimiento = 2020 - edad
nacimiento
```

[6]:
```
1931
```

[8]:
```
edad = int(input("Hola, ¿cuál es tu edad?"))
type(edad)
print(f"Tu edad es {edad}")
```

```
Hola, ¿cuál es tu edad? b
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-8-c34e893a9f14> in <module>
----> 1 edad = int(input("Hola, ¿cuál es tu edad?"))
      2 type(edad)
      3 print(f"Tu edad es {edad}")


ValueError: invalid literal for int() with base 10: 'b'
```

1.1 Funciones

[12]:
```
def sumar(a, b):
    """Función para realizar la suma de dos números"""
    return a + b
```

[14]:
```
resultado = sumar(58, 12)
resultado
```

[14]:
```
70
```

[15]:
```
sumar(89, 'hola')
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-15-d21b112c8aed> in <module>
----> 1 sumar(89, 'hola')


<ipython-input-12-d6d4d771a974> in sumar(a, b)
  1 def sumar(a, b):
  2     """Función para realizar la suma de dos números"""
----> 3     return a + b


TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

[16]:
```
sumar("hola", "mundo")
```

[16]:
```
'holamundo'
```

[17]:
```
def Saludar():
    print("Hola, bienvenido a la clase de Python")

Saludar()
```

```
Hola, bienvenido a la clase de Python
```

[18]:
```
def Saludar():
    return "Hola, bienvenido a la clase de Python"

print(f"{Saludar()}")
```

```
Hola, bienvenido a la clase de Python
```

[19]:
```
def sumar(a, b):
    return a + b
```

[20]:
```
sumar(b=89, a=25)
```

[20]:
```
114
```

[21]:
```
def restar(a, b):
    return a - b
```

[22]:
```
restar(12, 8)
```

[22]:
```
4
```

[23]:
```
restar(b=4, a=12)
```

[23]:
```
8
```

[24]:
```
restar(b=50, a=5)
```

[24]:
```
-45
```

[25]:
```
def saludar(nombre=None):
    if nombre is None:
        return "Hola, bienvenido a la clase de Python"
    else:
        return f"Hola {nombre}, bienvenido a la clase de Python"
```

[26]:
```
saludar()
```

[26]:
```
'Hola, bienvenido a la clase de Python'
```

[27]:
```
saludar("pepito")
```

[27]:
```
'Hola pepito, bienvenido a la clase de Python'
```

[28]:
```
saludar(58)
```

[28]:
```
'Hola 58, bienvenido a la clase de Python'
```

1.2 Funciones con argumentos indertiminados por posición

[29]:
```
def indeterminado(*args):
    for arg in args:
        print(arg)
```

[31]:
```
indeterminado("Hola", 8.9, 5, [9, 8, 7], ("Mundo", "saludo"), True)
```

```
Hola
8.9
```

```
5
[9, 8, 7]
('Mundo', 'saludo')
True
```

[32]:
```python
def diccionario(**kwargs):
    for kwarg in kwargs:
        print(f"{kwarg} => {kwargs[kwarg]}")
```

[33]:
```python
diccionario(validar=False, nombres=["Edwin", "Camila", "Eliana"], edades=(33,␣
↪22, 28))
```

```
validar => False
nombres => ['Edwin', 'Camila', 'Eliana']
edades => (33, 22, 28)
```

1.3 Funciones lambda

[34]:
```python
def sumar(a, b):
    return a + b

suma = lambda a, b : a + b
```

[35]:
```python
sumar(12, 5)
```

[35]: 17

[36]:
```python
suma(12, 5)
```

[36]: 17

[37]:
```python
restar = lambda a, b : a - b
```

[38]:
```python
restar(56, 12)
```

[38]: 44

[39]:
```python
restar(b=56, a=12)
```

[39]: -44

[40]:
```python
mayor = lambda a, b : a if a > b else b
```

[41]:
```python
mayor(-12, 12)
```

[41]: 12

[44]:
```
def mayor_def(a, b):
    if a > b:
        return a
    else:
        return b
```

[45]:
```
mayor_def(-12, 12)
```

[45]: 12

1.4 Referencia

[49]:
```
def doblar(num):
    num *= 2
    return num
```

[53]:
```
n = 10
print(n)
```

```
10
```

[51]:
```
doblar(n)
```

[51]: 20

[52]:
```
print(n)
```

```
10
```

[56]:
```
print(f"{n} => {doblar(n)} => {n}")
```

```
10 => 20 => 10
```

[57]:
```
n = 10
print(n)
doblar(n)
print(n)
```

```
10
10
```

[ ]:
```
def dividir(a, b):
    resultado
```

[58]:
```
a = 1
id(a)
```

[58]: 140705022058896