Material Imprimible

Curso de Python

Módulo 4

Argumentos de palabras clave arbitrarias, **kwargs

Si no sabe cuántos argumentos de palabra clave se pasarán a la función, agregue dos asteriscos: antes del nombre del parámetro en la definición de función.**

De esta manera, la función recibirá un *diccionario* de argumentos y podrá acceder a los elementos en consecuencia:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Valor de parámetro predeterminado

En el ejemplo siguiente se muestra cómo utilizar un valor de parámetro predeterminado.

Si llamamos a la función sin argumento, utiliza el valor predeterminado:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Pasar una lista como argumento

Puede enviar cualquier tipo de argumento de datos a una función (cadena, número, lista, diccionario, etc.) y se tratará como el mismo tipo de datos dentro de la función.

Por ejemplo, si envía una lista como argumento, seguirá siendo una lista cuando llegue a la función:

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

Valores devueltos

Para permitir que una función devuelva un valor, utilice la instrucción:return

```
def my_function(x):
  return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

La Declaración de pase

function las definiciones no pueden estar vacías, pero si por alguna razón tiene una definición sin contenido, coloque la instrucción para evitar recibir un error.functionpass

```
def myfunction():
  pass
# tener una definición de función vacía como esta, generaría un error sin la instrucción pass
```

Entrada por teclado: la función input()

En Informática, la "entrada" de un programa son los datos que llegan al programa desde el exterior. Actualmente, el origen más habitual es el teclado.

La 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 **Intro**, como muestra el siguiente ejemplo:

```
print("¿Cómo se llama? ")
nombre = input()
print("Me alegro de conocerle," ,nombre)
```

En el ejemplo anterior, el usuario escribe su respuesta en una línea distinta a la pregunta porque 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. Para separar la respuesta de la pregunta se ha añadido un espacio al final de la pregunta.

```
print("¿Cómo se llama? ", end="")
nombre = input()
print("Me alegro de conocerle," ,nombre)
```

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): como haría?

```
nombre = input("¿Cómo se llama? ")
print("Me alegro de conocerle", nombre)
```

Conversión de tipos

De forma predeterminada, la función input() convierte la entrada en una cadena, aunque escribamos un número. Si intentamos hacer operaciones, se producirá un error.

Round() es una función incorporada disponible con python. Le devolverá un número de flotador que se redondeará a los decimales que se dan como entrada. Si no se especifican los decimales a redondear, se considera como 0, y se redondeará al entero más cercano

Ejemplo:

Realizar una entrada de datos que indique la cantidad de pesos que tiene y que le dé la cantidad de dólares que puede comprar a un valor de 72.55 pesos cada dólar.

```
cantidad = int(input("Dígame una cantidad en pesos: "))
print(cantidad, "pesos son", round(cantidad / 72.55), "dolares")
```

Si se quiere que Python interprete la entrada como un número entero, se debe utilizar la función int() de la siguiente manera:

```
cantidad = int(input("Dígame una cantidad en pesos: "))
print(f"{cantidad} pesos son {round(cantidad / 72.55)} dolares")
```

De la misma manera, para que Python interprete la entrada como un número decimal, se debe utilizar la función float() de la siguiente manera:

Ejemplo:

Ingresar por pantalla la cantidad de dólares indicando con hasta dos decimales y realizar la conversión a pesos a un valor de 72.55

```
cantidad = float(input("Dígame una cantidad en dolares (hasta con 2 decimales): "))
print(f"{cantidad} dolares son {round(cantidad * 72.55)} pesos")
```

Variables como argumento de la función input()

La función input() sólo puede tener un argumento.

En versiones de Python anteriores a la versión 3.6 esto causaba problemas cuando se querían incorporar variables en el argumento de la función input(), pero las cadenas "f" permiten hacerlo fácilmente:

Ejemplo:

Realizar una entrada de datos que pregunte el nombre, luego preguntar por su apellido mostrando su nombre en pantalla y por último imprimir “me alegro de conocerte”, mostrando su nombre y apellido.

```
nombre = input("Dígame su nombre: ")
apellido = input(f"Dígame su apellido, {nombre}: ")
```

```
print(f"Me alegro de conocerle, {nombre} {apellido}.")
```

Ejemplo:

Realizar una entrada de un número, luego indicar que ingrese otro número mayor al anterior y por último mostrar la diferencia entre ellos.

```
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}.")
```

Recursividad

Python también acepta recursividad de función, lo que significa que una función definida puede llamarse a sí misma.

La recursividad es un concepto matemático y de programación común. Significa que una función se llama a sí misma. Esto tiene la ventaja de significado que puede recorrer los datos en bucle para alcanzar un resultado.

El desarrollador debe tener mucho cuidado con la recursividad, ya que puede ser bastante fácil deslizarse en la escritura de una función que nunca termina, o una que utiliza cantidades excesivas de memoria o potencia del procesador. Sin embargo, cuando se escribe correctamente la recursividad puede ser un enfoque muy eficiente y matemáticamente elegante para la programación.

```
def atras(segundos):
        segundos-=1
        if segundos >= 0:
        print(segundos)
        atras(segundos)
        else:
        print("Termino la cuenta regresiva")
atras(10)
```

Función recursiva sin retorno

Un ejemplo de una función recursiva sin retorno, es el ejemplo de cuenta regresiva hasta cero a partir de un número:

```
def cuenta_regresiva(numero):
        numero -= 1
        if numero > 0:
        print (numero)
    cuenta_regresiva(numero)
        else:
        print ( "Booooooooom!")
        print ("Fin de la función", numero)

cuenta_regresiva(5)
```

Función recursiva con retorno

Un ejemplo de una función recursiva con retorno, es el ejemplo del calculo del factorial de un número corresponde al producto de todos los números desde 1 hasta el propio número. Es el ejemplo con retorno más utilizado para mostrar la utilidad de este tipo de funciones:

```
def factorial(numero):
        print ("Valor inicial ->",numero)
        if numero > 1:
        numero = numero * factorial(numero -1)
        print ("valor final ->",numero)
        return numero
print (factorial(5))
```

En este ejemplo, tri_recursion() es una función que hemos definido para llamarse a sí misma ("recurse"). Usamos la variable k como los datos, que disminuyen (-1) cada vez que nos

recursivos. La recursividad termina cuando la condición no es mayor que 0 (es decir, cuando es 0).

Para un nuevo desarrollador puede tomar algún tiempo para averiguar cómo funciona exactamente esto, la mejor manera de averiguarlo es probando y modificándolo.

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result
print("\n\nRecursion Example Results")
tri_recursion(6)
```

Lambda de Python

Una función lambda es una pequeña función anónima.

Una función lambda puede tomar cualquier número de argumentos, pero solo puede tener una expresión.

Sintaxis

lambda *arguments* : *expression*

Se ejecuta la expresión y se devuelve el resultado:

Ejemplo

Una función lambda que agrega 10 al número pasado como argumento e imprime el resultado:

```
x = lambda a: a + 10
print(x(5))
```

Las funciones de Lambda pueden tomar cualquier número de argumentos:

Ejemplo

Función lambda que multiplica el argumento a con el argumento b e imprime el resultado

```
x = lambda a, b: a * b
print(x(5, 6))
```

Función lambda que suma el argumento a, b y c e imprime el resultado:

```
x = lambda a, b, c: a + b + c
print(x(5, 6, 2))
```

¿Por qué utilizar Lambda Functions?

El poder de lambda se muestra mejor cuando se utilizan como una función anónima dentro de otra función.

Supongamos que tiene una definición de función que toma un argumento y ese argumento se multiplicará por un número desconocido:

```
def myfunc(n):
  return lambda a : a * n
```

Utilice esa definición de función para crear una función que siempre duplique el número que envíe:

```
def myfunc(n):
  return lambda a : a * n
mydoubler = myfunc(2)
print(mydoubler(11))
```

O bien, utilice la misma definición de función para crear una función que siempre *triplica* el número que envía:

```
def myfunc(n):
  return lambda a : a * n
mytripler = myfunc(3)
print(mytripler(11))
```

O bien, utilice la misma definición de función para realizar ambas funciones, en el mismo programa:

```
def myfunc(n):
  return lambda a : a * n
mydoubler = myfunc(2)
mytripler = myfunc(3)
print(mydoubler(11))
print(mytripler(11))
```

Utilice funciones lambda cuando se requiera una función anónima durante un breve período de tiempo.

Clases y objetos de Python

Python es un lenguaje de programación orientado a objetos.

Casi todo en Python es un objeto, con sus propiedades y métodos.

Una clase es como un constructor de objetos o un "blueprint" para crear objetos.

Crear una clase

Para crear una clase, utilice la palabra clave :class

Ejemplo

Cree una clase denominada MyClass, con una propiedad denominada x:

```
class MyClass:
  x = 5
print(MyClass)
```

Crear objeto

Ahora podemos usar la clase denominada MyClass para crear objetos:

Ejemplo

Cree un objeto denominado p1 e imprima el valor de x:

```
class MyClass:
```

```
 x = 5
p1 = MyClass()
print(p1.x)
```

La función __init__()

Los ejemplos anteriores son clases y objetos en su forma más simple y no son realmente útiles en aplicaciones de la vida real.

Para entender el significado de las clases tenemos que entender la función __init__() integrada.

Todas las clases tienen una función llamada __init__(), que siempre se ejecuta cuando se inicia la clase.

Utilice la función __init__() para asignar valores a propiedades de objeto u otras operaciones que sean necesarias realizar cuando se crea el objeto:

Cree una clase denominada Person, utilice la función __init__() para asignar valores para name y age:

```
class Person:
 def __init__(self, name, age):
                    self.name = name
                    self.age = age
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Nota: La función __init__()se llama automáticamente cada vez que se utiliza la clase para crear un nuevo objeto

Métodos de objeto

Los objetos también pueden contener métodos. Los métodos de los objetos son funciones que pertenecen al objeto.

¿Cuál es la **diferencia** entre un **método** y una función? ... Un **método** puede operar con datos que están contenidos dentro de la clase (recordando que un **objeto** es una instancia de una clase - la clase es la definición, el **objeto** es una instancia de esos datos).
Una función es una pieza de código que se llama por su nombre. Se pueden pasar datos para operar (es decir, los parámetros) y, opcionalmente, se pueden devolver datos (el valor de retorno). Todos los datos que se pasan a una función se pasan explícitamente.
Un método es una pieza de código que se llama con un nombre que está asociado con un objeto. En la mayoría de los aspectos, es idéntico a una función, excepto por dos diferencias clave:
A un método se le pasa implícitamente el objeto sobre el que fue llamado.
Un método puede operar con datos que están contenidos dentro de la clase (recordando que un objeto es una instancia de una clase - la clase es la definición, el objeto es una instancia de esos datos).

Vamos a crear un método en la clase Person:
Inserte una función que imprima un saludo y ejecútela en el objeto p1:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Fuentes: https://qastack.mx/programming/1769403/what-is-the-purpose-and-use-of-kwargs
https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/funciones_recursivas.html#:~:text=Las%20funciones%

20recursivas%20son%20funciones,tendr%C3%A1%20una%20funci%C3%B3n%20recursiva%20infinita.

https://www.digitallearning.es/intro-programacion-js/objetos-propiedades-metodos.html