Funcion LAMBDA

Las funciones Lambda son funciones anónimas que solo pueden contener una expresión.

Podrías pensar que las funciones lambda son un tema intermedio o avanzado, pero aquí aprenderás de manera sencilla como puedes comenzar a utilizarlas en tu código.

Así creas una función en Python:

```
def mi_func(argumentos):
  # cuerpo de la función
```

Se declara la función con la palabra clave def, les das un nombre y entre paréntesis los argumentos que recibirá la función. Puede haber tantas líneas de código como quieras con todas las expresiones y declaraciones que necesites.

Pero algunas veces solo necesitarás una expresión dentro de tu función, por ejemplo, una función que duplique el valor de un argumento:

```
def doble(x):
  return x * 2
```

Esta función puede ser usada dentro de la función map, de la siguiente forma:

```
def doble(x):
  return x * 2

mi_lista = [1, 2, 3, 4, 5, 6]
lista_nueva = list(map(doble, mi_lista))
print(lista_nueva) # [2, 4, 6, 8, 10, 12]
```

Este es un buen lugar para usar una función lambda, ya que puede ser creada en el mismo lugar donde se necesite, esto significa menos líneas de código y así también evitarás nombrar una función que solo utilizaras una sola vez y que tendría que

ser almacenada en memoria,

Como usar funciones lambda en Python

Una función lambda se usa cuando necesitas una función sencilla y de rápido acceso: por ejemplo, como argumento de una función de orden mayor como los son map o filter

La sintaxis de una función lambda es lambda args: expresión. Primero escribes la palabra clave lambda, dejas un espacio, después los argumentos que necesites separados por coma, dos puntos :, y por último la expresión que será el cuerpo de la función.

Recuerda que no puedes darle un nombre a una función lambda, ya que estas son anónimas (sin nombre) por definición.

Una función lambda puede tener tantos argumento como necesites, pero debe tener una sola expresión.

Ejemplo 1

Por ejemplo, puedes escribir una función lambda que duplique sus argumentos lambda x: x * 2 y usarla con la función map para duplicar todos los elementos de una lista:

```
mi_lista = [1, 2, 3, 4, 5, 6]
lista_nueva = list(map(lambda x: x * 2, mi_lista))
print(lista_nueva)  # [2, 4, 6, 8, 10, 12]
```

Así luce sin una función lambda

```
def doble(x):
  return x*2

mi_lista = [1, 2, 3, 4, 5, 6]
lista_nueva = list(map(doble, mi_lista))
print(lista_nueva) # [2, 4, 6, 8, 10, 12]
```

Puedes notar la diferencia, entre usar una función lambda y el ejemplo con la función doble, ahora el código es más compacto y no hay una función extra utilizando espacio en memoria.

Ejemplo 2

También puedes escribir una función lambda que revise si un número es positivo, lambda x: x > 0, y usarla con la función filter para crear una lista de números positivos.

```
mi_lista = [18, -3, 5, 0, -1, 12]
lista_nueva = list(filter(lambda x: x > 0, mi_lista))
print(lista_nueva) # [18, 5, 12]
```

Una función lambda se define donde se usa, de esta manera no hay una función extra utilizando espacio en memoria.

Si una función es utilizada una sola vez, lo mejor es usar una función lambda para evitar código innecesario y desorganizado.

Ejemplo 3

También es posible que una función devuelva una función lambda.

Si necesitas funciones que multipliquen diferentes números, por ejemplo, duplicar, triplicar, etc... una función lambda puede ser útil

En lugar de crear múltiples funciones, puedes crear una sola función multiplicar_por() y llamarla con diferentes argumentos para crear una función que duplique o triplique.

```
def multiplicar_por (n):
  return lambda x: x * n

duplicar = multiplicar_por(2)
triplicar = multiplicar_por(3)
diez_veces = multiplicar_por(10)
```

La función lambda toma el valor de n de la función multiplicar_por(n) así que en duplicar el valor de n es 2, en

triplicar n vale 3 y en diez_veces vale 10. Y al llamar a estas funciones con un argumento podemos retornar el número multiplicado.

```
duplicar(6)
> 12
triplicar(5)
> 15
diez_veces(12)
> 120
```

Si no usáramos funciones lambda, tendríamos que crear una función diferente dentro de multiplicar_por, y se vería algo así:

```
def multiplicar_por(x):
  def temp (n):
    return x*n
  return temp
```

Usando una función lambda el código necesita menos líneas de código y es más legible.

Conclusión

Las funciones lambda son una manera compacta de escribir una función si solo necesitas una expresión corta. Talvez no sean algo que un programador principiante usaría, pero después de leer este artículo talvez puedas implementarlas en tu propio código.