Funciones y métodos en Python.

Aporte de Fabián Andrés Cortés Tróchez, para el Foro: Funciones Integradas de Python.

El ejercicio planteado inicialmente para el foro era responder a la pregunta: ¿Cuáles son las funciones incorporadas en Python usadas para la manipulación de cadenas, números y secuencias?

Las contribuciones ya realizadas por mis compañeros de curso son excelentes y completas en cuanto a la información y documentación proporcionada para listar y explicar algunas de las funciones incorporadas en Python para la manipulación de datos. Por esto, en lugar de repetir comentarios sobre un tema que ya conocemos bien, prefierí explorar un concepto que aún abordamos en el foro: el concepto de **método**, que en mi opinión está estrechamente relacionado con nuestro tema. Esto nos ayudará a entender mucho mejor de donde viene su sintáxis y como podemos usarlos apropiadamente.

¡Acompañame a aprender juntos sobre los métodos en Python!

Empecemos por mencionar que en Python, **los métodos son funciones**, por lo tanto actuan sobre un objeto y producen un determinado **efecto**, pero no todas las funciones son métodos.
Veámos ejemplos de funciones y métodos para entender mejor a qué me refiero:

```
# Obtener la longitud de una lista
frutas = ["mango", "banana", "manzana"]
longitud = len(frutas)
print("La longitud de la lista es: ", longitud)
```

```
---
@user:~/$ python3 example.py
La longitud de la lista es:  3
```

En este ejemplo vemos el efecto de dos funciones incorporadas en el interprete de Python, `len()` y `print()`. La primera devuelve el valor de la longitud de una secuencia, en este nuestra lista `frutas`, y la segunda nos proporciona una representación visual de un objeto que resulta de concatenar una cadena de texto `"La longitud de la lista es: "` y el valor de la variable `longitud`. En ambos casos usamos la sintáxis clásica para llamar una función en Python: `funcion()`, donde el `funcion` representa el nombre de nuestra función, seguida de dos paréntesis, entre los que añadiremos nuestros argumentos.

Ahora veámos una función un poco distinta.

```
# Convertir una cadena de texto a Mayúsculas
nueva_cadena = frutas[0].upper()
print("Esta es una fruta mayúscula: ", nueva_cadena)
```

```
---
@user:~/$ python3 example.py
Esta es una fruta mayúscula:  MANGO
```

Aquí utilizamos una función que opera sobre una de las cadenas en nuestra lista `frutas`; el efecto es claro al imprimir el resultado: todos los caractéres en nuestra cadena se fueron convertidos a mayúsculas. Pero, observa la sintaxis para invocar esta función, en este caso el objeto a manipular, la cadena almacenada en la lista `frutas` en la posición `[0]`, es enunciada desde afuera de la función, seguida por el nombre de la función y los paréntesis. ¿Porqué no podemos usar una síntaxis del tipo: `upper(frutas[0])`? La respuesta rápida es que, la función `upper()` solo existe dentro del contexto de la clase `'str'`, o dicho de otro modo, `upper()` es un **método** de dicha clase.

Como mencionamos, los métodos son de hecho funciones, la diferencia es que un método se define en el contexto de una **clase** en particular, y por ende su el método está restringido a operar estrictamente sobre los **objetos** que pertenecen a dicha clase. Las definiciones de **clases** y **objetos** son extensas y son inherentes de la programación orientado a objetos, pero podemos reducir el asunto a una serie de ideas clave: Un objeto es una **instancia** de una clase. Esta clase determina su lista de atributos y la forma en que puede interactuar con otros objetos.

Los objetos son entidades individuales que representan cosas de la vida real, por poner un ejemplo: considera a `ventana_1` y `ventana_2`, objetos que pertenecen a la clase `Ventanas`; `ventana` cuenta con una serie de atributos individuales, tales como `longitud`, `material`, y `color`, y puede realizar las funciones (métodos) propias de su clase, como `abrir()` y `cerrar()`. El objeto `ventana_2` también comparte esta lista de atributos y funciones, al igual que cualquier otro objeto que pertenezca a la misma clase, aunque sus valores individuales en cada atributo pueden variar; por otro lado, un objeto `silla_1` de la clase `Muebles` no tendrá acceso a estos atributos y métodos, que no fueron definidos para su clase.

La notación de punto: `objeto.metodo()`, le indica al interprete de Python que opere sobre este objeto accediendo a los atributos propios del mismo, y solo tendrá un efecto si el `objeto` pertenece a la clase para la cual el método fue definido.

Será muco más interesante que observes el proceso de crear una clase y sus métodos para comprender las reglas que gobiernan los métodos en Python. Te invito a abrir tu editor de código favorito para que veámos juntos un ejemplo simple de la construcción de una clase, junto sus atributos, y métodos instancia y el uso de la misma:

```
class Persona:
# Aquí se definen los atributos de la clase Persona
   def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
        self.pasos = 0 # Este atributo representa los pasos que una
'Persona' recorre

# Definir el método 'caminar' que incrementa 'pasos' en 10 unidades.
   def caminar(self):
        self.distancia_recorrida += 10
        print(f"{self.nombre} acaba de dar 10 pasos. Pasos totales:
{self.pasos}")
```

En este código hemos definido la clase `Persona`, junto con su lista de atributos: `nombre`, `edad`, y `pasos`; y dentro de dicha clase también hemos creado un método denominado `caminar()`, que tiene el efecto de incrementar en 10 el valor del atributo pasos. También observa que hemos definido que el atributo `pasos` tendrá por defecto el valor de `0`, por lo que todos los objetos pertenecientes a esta clase tendrán el mismo valor de `pasos` si este no es explicitamente especificado, o posteriormente modificado un método.

A continuación, vamos a crear una instancia de la clase `Persona`:

```
# Crear un objeto de la clase Persona, nombre: "Andres", edad: 30
andres = Persona("Andres", 30)
```

Aquí hemos creado un objeto, y lo almacenamos en una variable llamada andres, a dicho objeto le hemos asignado los siguientes atributos: nombre = "Andres", y edad = 30. El siguiente paso es acceder a nuestro objeto a través del método que hemos definido antes. La sintáxis la hemos explicado antes:

```
# Invocar el método caminar() para el objeto Andres
andres.caminar()
```

```
---
@user:~/$ python3 example.py
Andres acaba de dar 10 pasos. Pasos totales: 10
```

Observamos como el método `caminar()` nos permitió acceder y modificar los atributos de nuestro objeto `andres`. No profundizaremos el tema, pero te adelanto que el uso de métodos puede tener muchas ventajas derivadas del uso de clases; por mencionar algunos, el encapsulamiento y modularidad de los datos, herencia y polimorfismo a través de diferentes objetos y clases, y en general facilita la legibilidad y mantenimiento de tu código fuente.

Entonces recapitulemos lo que hemos aprendido. Aunque los métodos son técnicamente funciones, se diferencian en que están asociados a una clase específica y operan en instancias de esa clase. También aprendimos que las clases presentan una sintaxis propia, la sintaxis punto `objeto.metodo()`, que le instruye a Python para acceder a la clase del `objeto` y operar sobre sus atributos. Mientras que las funciones pueden ser independientes y operar en cualquier contexto, los métodos están diseñados para interactuar con objetos específicos y modelar su comportamiento. Es así como funcionan, algunos métodos incorporados en Python, como los métodos de cadenas (`upper()`, `split()`), métodos de listas (`append()`, `sort()`), y entre otros; aunque como vimos, tambien podemos crear nuestras propias clases, con sus propios métodos.

Los métodos son sin duda una parte fundamental de la programación orientada a objetos en Python. Permiten que los objetos interactuen entre sí y definan su comportamiento en función de sus atributos y funciones específicas. Si disfrutaste tanto como yo aprender este tema, te invito a que sigas aprendiendo sobre programación orientada a objetos en Python siguiendo estos enlaces:

- Python Built-in Types
- Programación orientada a objetos
- Python Methods, Functions and Libraries
- A Beginner's Guide to Python Object-Oriented Programming (OOP)