
Universidad
Tecmilenio®

Universidad Tecmilenio

Fundamentos y aplicaciones de la inteligencia artificial

Funciones en Python

Manejo de archivos, objetos y clases en Python

Introducción

Hace algunas décadas, a medida que los programas se iban complejizando, también aumentaba su tamaño y la dificultad de mantenerlos después en el tiempo, por esta razón, uno de los principales retos de los lenguajes de programación es facilitarles a los desarrolladores la mayor cantidad de elementos con los cuales puedan escribirlos.

Dividir estos pesados programas en secciones más pequeñas y reutilizables fomentó el surgimiento de las **funciones, objetos y clases**. Python es uno de los mayores exponentes de esta práctica. Por otro lado, como programadores, otro objetivo es lograr las mejores compensaciones entre el almacenamiento de datos en el disco y la RAM, por ello, saber manejarlos usando medios no volátiles como discos es deber primordial.

Durante el desarrollo de este tema aprenderás sobre las funciones en Python, el manejo de archivos, objetos y clases para simplificar, pero, sobre todo, tener mayor alcance con nuestros programas, a través del mejor uso de los recursos.

ID de la fotografía:1125274914

Explicación

Definición de funciones

Una función es un bloque de código diseñado para realizar una tarea específica. Una vez declarada, puede ser llamada dentro de nuestro programa todas las veces que se necesite realizar esta tarea nuevamente. Dentro de una función se puede escribir todo tipo de código, el cual, en algunas ocasiones, puede usarse para devolver algún resultado o simplemente para ejecutar una acción en específico. Las funciones nos permiten desarrollar programas más simples, fáciles de entender, de probar y de corregir, en caso de ser necesario (Matthes, 2019).

```
In [1]:    def nombre_de_la_funcion():
               pass
```

Te explicamos el ejemplo anterior:

- La primera línea muestra la utilización de la palabra reservada *def*, la cual le indica a Python que se comenzará a describir una función.
- A continuación, se coloca el nombre de dicha función, seguida de unos paréntesis cerrados y el símbolo de dos puntos.
- La segunda línea en el ejemplo también es una palabra reservada *pass* que significa saltar.
- La sección donde escribimos el código interno de la función se conoce como el cuerpo de esta.

Explicación

Una función puede recibir diversos valores de entrada. Cuando las variables que representan esos valores se utilizan en la declaración de la función se les denominan parámetros; entonces, es correcto expresar que en el siguiente ejemplo la función *saludar()* recibe el parámetro de entrada "*nombre*".

El valor que se le pasa a la función durante la llamada a la misma se denomina argumento. De esa forma se puede expresar que en el siguiente ejemplo la función *saludar()* es invocada con el argumento "*María*".

```
In [6]:  def  saludar(nombre):
             print(f"Buenos dias {nombre}, ¿Cómo estas hoy?")

         saludar("María")
```

```
Buenos días, María, ¿Cómo estas hoy?
```

Explicación

Universidad Tecmilenio

También podemos utilizar múltiples argumentos y hacer diversas llamadas a la función, tantas como sea necesario, simplificando así nuestro código.

In [8]:

```
def saludar(nombre, dia_semana):
  print(f"Buenos dias {nombre}, feliz {dia_semana}")

saludar("Arturo", "sábado")
saludar("Juan", "domingo")
```

```
Buenos días Arturo, feliz sábado
Buenos días Juan, feliz domingo
```

Explicación

Manejo de archivos usando la función *open, close y write*

De acuerdo con Lubanovic (2019), los archivos planos son el ejemplo de archivo más simple y antiguo. Son solo una secuencia de bytes almacenados bajo un nombre y se pueden leer (*read*) o escribir (*write*) desde y hacia su propia localidad de memoria.

Antes de leer o escribir un archivo, debe abrirse lo siguiente:

```
archivo = open('nombre', 'modo')
```

A continuación, se ofrece una breve explicación para cada elemento:

- Archivo es el archivo devuelto por open().
- 'nombre' es el nombre en forma de cadena (string) del archivo.
- 'modo' también es una cadena que indica el tipo de archivo y lo que se desea hacer con él.

La primera letra de 'modo' indica la operación:
- 'r' significa leer.
- 'w' significa escribir. Si el archivo no existe, se crea. Si el archivo existe, se sobrescribe.
- 'x' significa escribir, pero solo si el archivo aún no existe.
- 'a' significa agregar (escribir después del final) si el archivo existe.

La segunda letra de modo es el tipo de archivo:
- 't' (o nada) significa texto.
- 'b' significa binario.

Explicación

Uso de funciones para manipular archivos

Después de abrirse el archivo, se llama a las funciones correspondientes para leer o escribir datos. Por último, el archivo debe cerrarse.

A continuación, se realizará el ejemplo de un programa en el que se creará un archivo a partir de una cadena de Python.

Las siguientes líneas de código permiten almacenar un poema en una cadena y calcular su longitud:

```
In [1]: poema = '''<< La luna se puede tomar a cucharadas
        o como una cápsula cada dos horas.
        Es buena como hipnótico y sedante
        y también alivia
        a los que se han intoxicado de filosofía.>> Jaime Sabines'''
        len(poema)

        183
```

A continuación, se escribe el poema completo bajo el nombre *"LaLuna"* utilizando *write*( ):

```
In [2]: fout = open('LaLuna','wt')
        fout.write(poema)
        fout.close()
```

Otras funciones para manipulación de archivos

A continuación, se mencionan y explican otras funciones para manipular archivos con Python:

***print( )*:** para escribir texto en un archivo con el mismo nombre sería mediante la función *print( )*. ¿Qué función debería usarse?, ¿*write( )* o *print( )*? La diferencia entre ambas es que la segunda opción, *print( )*, agrega de forma predeterminada un espacio después de cada argumento y una nueva línea al final.

***read()*:** si se llama a *read()* sin especificar ningún argumento, se extrae todo lo que hay en el archivo en una sola ejecución.

***readline( )*:** se puede leer el archivo una línea a la vez, utilizando la función *readline( )*.

***readlines( )*:** lee una línea a la vez y devuelve una lista de cadenas correspondientes a cada línea del texto.

***reader( )*:** crea filas separadas por líneas de avance y las columnas están separadas por comas, lo cual permite manipular archivos de texto separados por comas CSV (archivos estructurados).

Explicación

Objetos y clases

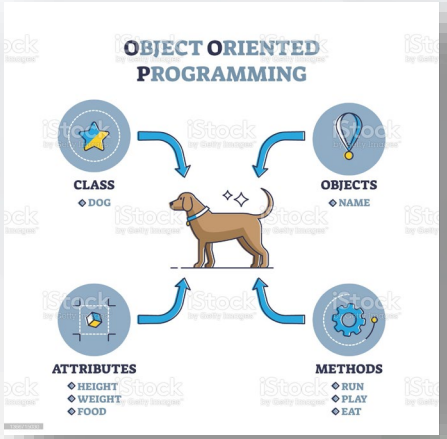
De acuerdo con Matthes (2019), en la programación orientada a objetos, a la creación de un objeto a partir de una clase se le denomina instanciación y es así como se trabaja con **instancias** de una clase.

Casi cualquier cosa puede modelarse usando clases. Se comenzará escribiendo una clase simple: *Perro*, que representa a un perro, no a un perro en particular, sino a cualquier perro. Ahora, es necesario hacerse la siguiente pregunta: ¿qué es lo que se sabe sobre la mayoría de los perros domésticos? Una respuesta sería que todos tienen nombre y edad. También se sabe que la mayoría de los perros pueden sentarse y darse la vuelta.

Esas dos piezas de información (nombre y edad) y esos dos comportamientos (sentarse y darse la vuelta) se incluirán en la clase *Perro* porque son comunes en la mayoría de los perros. Esta clase indicará a Python cómo crear un objeto que represente a un perro.

En este ejemplo, cada instancia creada a partir de la clase *Perro* almacenará un nombre y una edad, y se le dará a cada perro la capacidad de *sentarse()* y *darse_la_vuelta()*.

```
In [1]: class Perro:
            """Un intento simple de modelar un perro."""
            def __init__(self, nombre, edad):
                """Inicializar los atributos de nombre y edad."""
                 self.nombre = nombre
                 self.edad = edad
            def sentarse(self):
                """Simula un perro sentado como respuesta a una orden."""
                print(f"{self.nombre} ahora está sentado.")
            def darse_la_vuelta(self):
                """Simular darse la vuelta como respuesta a un
        comando."""
                print(f"{self.nombre} se dio la vuelta!")
```



ID de la ilustración:1366715030

Explicación

Los nombres en mayúscula se refieren a clases en Python.

Una función que forma parte de una clase es un *método,* el cual posee las mismas propiedades que cualquier función.

El método del ejemplo anterior tiene dos guiones bajos (_), al inicio y al final, esto ayuda a evitar que los nombres de métodos predeterminados de Python entren en conflicto con los nombres de sus métodos.

El parámetro *self* es obligatorio en la definición del método y debe ir primero antes que los demás parámetros.

Las variables a las que se puede acceder a través de instancias como esta se denominan **atributos**.

La clase *Perro* es un conjunto de instrucciones que le dice a Python cómo crear instancias individuales que representen perros específicos. Ahora, se procederá a crear una instancia que represente a un perro en específico:

```
In [2]: mi_perro = Perro ('Chad', 2)
        print(f"El nombre de mi perro es {mi_perro.nombre}.")
        print(f"Mi perro tiene {mi_perro.edad} años")
```

```
El nombre de mi perro es Chad.
Mi perro tiene 2 años
```

Actividad

Objetivo: programar ejercicios que implementen manejo de archivos, objetos y clases.

1. Escribe un programa en Python que cree la nueva clase *Persona,* la cual tenga nombre, apellido y edad, y pueda comer y dormir.

2. Por medio de la clase *Persona*, modela la información del punto anterior y crea una instancia que contenga tu *nombre*, *apellido* y *edad* como atributos.

3. Crea, además, los métodos para comer y dormir.

¡Éxito!

ID de la fotografía:875477964

Cierre

En este tema has aprendido la importancia de las funciones al momento de reutilizar código en programación y cómo se utilizan las mismas para manejar archivos planos (de texto) o estructurados (CSV). Por otro lado, aprendiste la importancia de administrar de manera óptima los espacios de almacenamiento.

Además, comprendiste cómo la programación orientada resulta útil para representar conceptos y situaciones del mundo real, permitiendo entender la lógica detrás del código y facilitando la comprensión del mismo, con el potencial de extenderlo a la creación de programas que aborden prácticamente cualquier problema de forma eficaz.

ID de la fotografía:1300749998

Bibliografía

Matthes, E. (2019). *Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming* (2ª ed.). Estados Unidos: No Starch Press.

Lubanovic, B. (2019). *Introducing Python: Modern Computing in Simple Packages* (2ª ed.). Estados Unidos: O'Reilly Media.