ESTRUCTURA DE DATOS Y PROGRAMACIÓN ORIENTADA A OBJETOS

Tema Nº7:El enfoque orientado a objetos: clases, métodos, objetos y las características estándar

Indicador de logro Nº7:Ejecuta los conceptos básicos de la programación orientada a objetos a través de un lenguaje de programación.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº7:**

El enfoque orientado a objetos: clases, métodos, objetos y las características estándar

**Subtema 7.1:**

Conceptos básicos de programación de orientada a objetos

Cuando te encuentras fuera del intérprete de Python y entras de nuevo, las definiciones que hiciste (funciones y variables) escribir un programa más o menos largo, es mejor que uses un editor de texto para preparar la entrada para el intérprete y ejecutarlo con ese archivo como entrada. Esto es conocido como crear un guion, o script. Si tu programa se vuelve más largo, quizás quieras separarlo en distintos archivos para un mantenimiento más fácil y definir cada programa la Programación Orientada a Objetos (POO), es un paradigma de programación cuyo núcleo central suministra la base y modelo para resolver problemas el cual se basa en las interacciones de objetos (todo lo descrito en el título anterior, Pensar en objetos) para resolver las necesidades de un sistema informático. Básicamente, este paradigma se compone de 6 elementos y 7 características que veremos en el proceso de Python, elementos de la POO, pueden entenderse como los materiales que necesitamos para diseñar y programar un sistema, mientras que las características, podrían asumirse como las herramientas de las cuáles disponemos para construir el sistema con esos materiales.

* **CONCEPTOS BÁSICOS DE PROGRAMACIÓN DE OBJETOS:**

La programación orientada a objetos (POO, u OOP según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

Está basada en varias técnicas, como las siguientes:

* Herencia.
* Cohesión.
* Abstracción.
* Polimorfismo.
* Acoplamiento.
* Encapsulación.
* **TU PRIMERA CLASE:**

La palabra "clase" tiene muchos significados, pero no todos son compatibles con las ideas que queremos discutir aquí. La "clase" que nos interesa es como una "categoría", como resultado de similitudes definidas con precisión.

En este ejemplo, discutiremos la primera subclase solo **vehículos terrestres**. Si lo desea, puede continuar con las clases restantes.

Los **vehículos terrestres** se pueden dividir aún más, según el método con el que impacten el suelo. Entonces, podemos enumerar:

* Vehículos de ruedas;
* Aerodeslizador.

La jerarquía que hemos creado se ilustra en la figura:

Gráfico 1.1

* **DEFINICIÓN DE CLASES:**

La forma más sencilla de definición de una clase se ve así:

class Clase:

<declaración-1>

.

.

.

<declaración-N>

Ejemplo de creación de la clase **Antena**, **Pelo** y **Ojo**:

class Antena:

pass

class Pelo:

pass

class Ojo:

pass

La palabra clave **pass** llena la clase con nada. No contiene ningún método o propiedad.

* **¿QUÉ ES UN OBJETO?**

Un objeto es un ser que pertenece a una clase. Un objeto es una encarnación de los requisitos, rasgos y cualidades asignados a una clase específica.

* **CREANDO OBJETOS (INSTANCIAR UNA CLASE):**

class Antena:

pass

antenas1 = Antena() # objeto perteneciente a la Clase Antena

class Pelo:

pass

pelos1 = Pelo() # objeto perteneciente a la Clase Pelo

class Ojo:

pass

ojos1 = Ojo() # objeto perteneciente a la Clase Ojo

* **¿QUÉ ES UNA PILA?**

Una pila es una estructura desarrollada para almacenar datos de una manera muy específica. Imagina una pila de monedas.

Una pila es un objeto con dos operaciones elementales, convencionalmente llamada **push** (cuando se coloca un nuevo elemento en la parte superior) y **pop** (cuando un elemento existente se retira de la parte superior).

* **PILA - UN ENFOQUE DE PROCEDIMIENTO:**

Vamos a implementar una pila en Python. Esta será una pila muy simple, y le mostraremos cómo hacerlo en dos enfoques independientes: de procedimiento y objetivo.

pila = []

Estamos listos para definir una función que pone un valor en la pila. Aquí están las presuposiciones para ello:

* El nombre de la función es **push**.
* La función obtiene un parámetro (este es el valor que se pondrá en la pila).
* La función no devuelve nada.
* La función agrega el valor del parámetro al final de la pila.

def push(valor):

pila.append(valor)

Ahora es el momento para que una función quite un valor de la pila. Así es como puedes hacerlo:

* El nombre de la función es **pop**.
* La función no obtiene ningún parámetro.
* La función devuelve el valor tomado de la pila.
* La función lee el valor de la parte superior de la pila y lo elimina.

def pop():

valor = pila[-1]

del pila[-1]

return valor

Vamos a ensamblar todas las piezas para poner la pila en movimiento. El programa completo inserta tres números en la pila, los saca e imprime sus valores en la pantalla.

pila = []

def push(valor):

pila.append(valor)

def pop():

valor = pila[-1]

del pila[-1]

return valor

push(3)

push(2)

push(1)

print(pop())

print(pop())

print(pop())

El programa envía el siguiente texto a la pantalla:

1

2

3

* **PILA DESDE CERO:**

Estos son algunos de ellos:

* La variable esencial (la lista de pila) es altamente vulnerable, cualquiera puede modificarlo de forma incontrolable, destruyendo la pila.
* También puede suceder que un día necesites más de una pila; tendrás que crear otra lista para el almacenamiento de la pila, y probablemente también otras funciones **push** y **pop**.
* La capacidad de **ocultar** (proteger) los valores seleccionados contra el acceso no autorizado se denomina **encapsulación**
* Cuando tiene una clase que implementa todos los comportamientos de pila necesarios, puede **producir tantas pilas como desee**; no necesita copiar o replicar ninguna parte del código.
* La capacidad de enriquecer la pila con nuevas funciones proviene de la **herencia**; puede crear una nueva clase (una subclase) que herede todos los rasgos existentes de la superclase y agrega algunos nuevos.

Ahora, esperamos dos cosas de ello:

* Queremos que la clase tenga una **propiedad** como almacenamiento de la pila: tenemos que “instalar” una lista dentro de cada objeto de la clase.
* Luego, queremos que la lista se **oculte** a la vista de los usuarios de la clase.

**¿CÓMO SE HACE ESTO?**

Se crea una función que se denomina **constructor**, ya que su propósito general es construir un nuevo objeto. El constructor debe saber todo sobre la estructura del objeto y debe realizar todas las inicializaciones necesarias.

class Pila:

def \_\_init\_\_(self):

print("Hola")

pila = Pila()

* El nombre del constructor es siempre **\_\_init\_\_**.
* Debe tener al menos un parámetro; el parámetro se usa para representar el objeto recién creado; puede usar el parámetro para manipular el objeto y enriquecerlo con las propiedades necesarias.
* Nota: El parámetro obligatorio generalmente se denomina **self**. Es solo una convención, pero debe seguirlo. Simplifica el proceso de lectura y comprensión de su código.

**EJECUTAR EL CÓDIGO AHORA:**

Aquí está su salida:

Hola

* **VARIABLES DE INSTANCIA Y DE CLASE:**

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos declarar una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

**Actividad:**

**EJERCICIOS:**

**EJERCICIO 1:**

Implementar una clase llamada Alumno que tenga como atributos su nombre y su nota. Definir los métodos para inicializar sus atributos, imprimirlos y mostrar un mensaje si está regular (nota mayor o igual a 4). Definir dos objetos de la clase Alumno.

class Alumno:

def inicializar(self, nombre, nota):

self.nombre = nombre

self.nota = nota

def imprimir(self):

print("Nombre: ", self.nombre)

print("Nota: ", self.nota)

def mostrar\_estado(self):

if self.nota >= 4:

print("Regular")

else:

print("Libre")

# bloque principal

alumno1 = Alumno()

alumno1.inicializar("diego", 2)

alumno1.imprimir()

alumno1.mostrar\_estado()

alumno2 = Alumno()

alumno2.inicializar("ana", 10)

alumno2.imprimir()

alumno2.mostrar\_estado()

**EJERCICIO 2:**

Confeccionar una clase que permita carga el nombre y la edad de una persona. Mostrar los datos cargados. Imprimir un mensaje si es mayor de edad (edad >= 18).

class Persona:

def inicializar(self, nombre, edad):

self.nombre = nombre

self.edad = edad

def imprimir(self):

print("Nombre: ", self.nombre)

print("Edad: ", self.edad)

def mayor\_edad(self):

if self.edad >= 18:

print("Es mayor de edad")

else:

print("No es mayor de edad")

# bloque principal

persona1 = Persona()

persona1.inicializar("diego", 40)

persona1.imprimir()

persona1.mayor\_edad()

**EJERCICIO 3:**

Desarrollar un programa que cargue los lados de un triángulo e implemente los siguientes métodos: inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no. El nombre de la clase llamarla Triangulo.

class Triangulo:

def inicializar(self):

self.lado1 = int(input("Ingrese primer lado: "))

self.lado2 = int(input("Ingrese segundo lado: "))

self.lado3 = int(input("Ingrese tercer lado: "))

def imprimir(self):

print("Valores de los lados del triángulo: ")

print("Lado 1: ", self.lado1)

print("Lado 2: ", self.lado2)

print("Lado 3: ", self.lado3)

def lado\_mayor(self):

print("Lado mayor")

if self.lado1 > self.lado2 and self.lado1 > self.lado3:

print(self.lado1)

else:

if self.lado2 > self.lado3:

print(self.lado2)

else:

print(self.lado3)

def es\_equilatero(self):

if self.lado1 == self.lado2 and self.lado1 == self.lado3:

print("El triángulo es equilátero")

else:

print("El triángulo no es equilátero")

# bloque principal

triangulo1 = Triangulo()

triangulo1.inicializar()

triangulo1.imprimir()

triangulo1.lado\_mayor()

triangulo1.es\_equilatero()