ESTRUCTURA DE DATOS Y PROGRAMACIÓN ORIENTADA A OBJETOS

Tema Nº9:El enfoque orientado a objetos: clases, métodos, objetos y las características estándar, manejo de excepciones

Indicador de logro Nº9:Implementa aplicaciones con método constructor y atributos privados y públicos a través de un lenguaje de programación.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº9:**

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

**Subtema 9.1:**

Clases, métodos, objetos y las características estándar, manejo de excepciones

Python te permite separar tu programa en módulos que pueden reusarse en otros programas en Python. Viene con una gran colección de módulos estándar que puedes usar como base de tus programas, o como ejemplos para empezar a aprender a programar en Python. Algunos de estos módulos proveen cosas como entrada/salida a archivos, llamadas al sistema, sockets, e incluso crear la herencia múltiple: la capacidad de una subclase de heredar de múltiples superclases. Esto conlleva un problema, y es que, si varias superclases tienen los mismos atributos o métodos, la subclase sólo podrá heredar de una de ellas. En estos casos Python dará prioridad a las clases más a la izquierda en el momento de la declaración de la subclase.

* **MÉTODO \_\_init\_\_ DE LA CLASE:**

El método **\_\_init\_\_** es un método especial de una clase en Python. El objetivo fundamental del método **\_\_init\_\_** es inicializar los atributos del objeto que creamos.

Básicamente el método **\_\_init\_\_** remplaza al método inicializar que habíamos hecho en el concepto anterior.

Las ventajas de implementar el método **\_\_init\_\_** en lugar del método inicializar son:

1. El método **\_\_init\_\_** es el primer método que se ejecuta cuando se crea un objeto.
2. El método **\_\_init\_\_** se llama automáticamente. Es decir, es imposible de olvidarse de llamarlo ya que se llamará automáticamente.
3. Quien utiliza POO en Python (Programación Orientada a Objetos) conoce el objetivo de este método.

* **VEAMOS LA SINTAXIS DEL CONSTRUCTOR:**

def \_\_init\_\_([parámetros]):

[algoritmo]

**Actividad:**

**EJERCICIOS:**

**EJERCICIO 1:**

Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. En el método **\_\_init\_\_** cargar los atributos por teclado y luego en otro método imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000).

class Empleado:

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

self.nombre = input("Ingrese el nombre del empleado: ")

self.sueldo = float(input("Ingrese el sueldo: "))

def imprimir(self):

print("Nombre: ", self.nombre)

print("Sueldo: ", self.sueldo)

def paga\_impuestos(self):

if self.sueldo > 3000:

print("Debe pagar impuestos")

else:

print("No paga impuestos")

# bloque principal

empleado1 = Empleado()

empleado1.imprimir()

empleado1.paga\_impuestos()

EXPLICACIÓN:

Definimos el método **\_\_init\_\_** donde cargamos por teclado el nombre del empleado y su sueldo:

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

self.nombre = input("Ingrese el nombre del empleado: ")

self.sueldo = float(input("Ingrese el sueldo: "))

Este método se ejecuta inmediatamente luego que se crea un objeto de la clase Empleado:

empleado1 = Empleado()

Como vemos no llamamos directamente al método **\_\_init\_\_** sino que se llama automáticamente.

Los otros dos métodos tienen por objeto mostrar los datos del empleado y mostrar una leyenda si paga impuestos o no:

def imprimir(self):

print("Nombre: ", self.nombre)

print("Sueldo: ", self.sueldo)

def paga\_impuestos(self):

if self.sueldo > 3000:

print("Debe pagar impuestos")

else:

print("No paga impuestos")

Desde el bloque principal donde creamos un objeto de la clase Empleado debemos llamar explícitamente a estos dos métodos:

empleado1.imprimir()

empleado1.paga\_impuestos()

**EJERCICIO 2:**

Desarrollar una clase que represente un punto en el plano y tenga los siguientes métodos: inicializar los valores de x e y que llegan como parámetros, imprimir en que cuadrante se encuentra dicho punto (concepto matemático, primer cuadrante si x e y son positivas, si x < 0 e y > 0 segundo cuadrante, etc.)

class Punto:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def imprimir(self):

print("Coordenada del punto")

print("(", self.x, ",", self.y, ")")

def imprimir\_cuadrante(self):

if self.x > 0 and self.y > 0:

print("Primer cuadrange")

else:

if self.x < 0 and self.y > 0:

print("Segundo cuadrante")

else:

if self.x < 0 and self.y < 0:

print("Tercer cuadrante")

else:

if self.x > 0 and self.y < 0:

print("Cuarto cuadrante")

# bloque principal

punto1 = Punto(10, -2)

punto1.imprimir()

punto1.imprimir\_cuadrante()

EXPLICACIÓN:

En este problema el método **\_\_init\_\_** aparte del parámetro **self** que siempre va tenemos otros dos parámetros:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

Desde el bloque principal donde creamos un objeto de la clase Punto pasamos los datos a los parámetros:

punto1 = Punto(10,-2)

**EJERCICIO 3:**

Desarrollar una clase que represente un Cuadrado y tenga los siguientes métodos: inicializar el valor del lado llegando como parámetro al método **\_\_init\_\_** (definir un atributo llamado lado), imprimir su perímetro y su superficie.

class Cuadrado:

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

self.lado = lado

def mostrar\_perimetro(self):

per = self.lado \* 4

print("El perimetro del cuadrado es: ", per)

def mostrar\_superficie(self):

sup = self.lado \* self.lado

print("La superficie del cuadrado es: ", sup)

# bloque principal

cuadrado1 = Cuadrado(12)

cuadrado1.mostrar\_perimetro()

cuadrado1.mostrar\_superficie()

**EJERCICIO 4:**

Implementar la clase Operaciones. Se deben cargar dos valores enteros por teclado en el método **\_\_init\_\_**, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

class Operaciones:

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

self.valor1 = int(input("Ingrese primer valor: "))

self.valor2 = int(input("Ingrese segundo valor: "))

def sumar(self):

su = self.valor1 + self.valor2

print("La suma es: ", su)

def restar(self):

re = self.valor1 - self.valor2

print("La resta es: ", re)

def multiplicar(self):

pro = self.valor1 \* self.valor2

print("El producto es: ", pro)

def division(self):

divi = self.valor1 / self.valor2

print("La división es: ", divi)

# bloque principal

operacion1 = Operaciones()

operacion1.sumar()

operacion1.restar()

operacion1.multiplicar()

operacion1.division()

* **LLAMADA DE MÉTODOS DESDE OTRO MÉTODO DE LA MISMA CLASE:**

Hasta ahora todos los problemas planteados hemos llamado a los métodos desde donde definimos un objeto de dicha clase, por ejemplo:

empleado1 = Empleado("diego", 2000)

empleado1.paga\_impuestos()

**UTILIZAMOS LA SINTAXIS**:

[nombre del objeto].[nombre del método]

Es decir, antecedemos al nombre del método el nombre del objeto y el operador punto.

Ahora bien, que pasa si queremos llamar dentro de la clase a otro método que pertenece a la misma clase, la sintaxis es la siguiente:

self.[nombre del método]

Es importante tener en cuenta que esto solo se puede hacer cuando estamos dentro de la misma clase.

**EJERCICIOS:**

**EJERCICIO 1**:

Plantear una clase Operaciones que solicite en el método **\_\_init\_\_** la carga de dos enteros e inmediatamente muestre su suma, resta, multiplicación y división. Hacer cada operación en otro método de la clase Operación y llamarlos desde el mismo método **\_\_init\_\_**.

class Operacion:

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

self.valor1 = int(input("Ingrese primer valor: "))

self.valor2 = int(input("Ingrese segundo valor: "))

self.sumar()

self.restar()

self.multiplicar()

self.dividir()

def sumar(self):

suma = self.valor1 + self.valor2

print("La suma es: ", suma)

def restar(self):

resta = self.valor1 - self.valor2

print("La resta es: ", resta)

def multiplicar(self):

multi = self.valor1 \* self.valor2

print("El producto es: ", multi)

def dividir(self):

divi = self.valor1 / self.valor2

print("La división es: ", divi)

# bloque principal

operacion1 = Operacion()

* **COLABORACIÓN DE CLASES:**

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.

Plantearemos un problema separando las actividades en dos clases.

**EJERCICIOS:**

EJERCICIO 1:

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositado.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco.

Luego debemos definir los atributos y los métodos de cada clase:

Cliente

atributos

nombre

monto

métodos

\_\_init\_\_

depositar

extraer

retornar\_monto

Banco

atributos

3 Cliente (3 objetos de la clase Cliente)

métodos

\_\_init\_\_

operar

depositos\_totales

class Cliente:

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

self.nombre = nombre

self.monto = 0

def depositar(self, monto):

self.monto = self.monto + monto

def extraer(self, monto):

self.monto = self.monto - monto

def retornar\_monto(self):

return self.monto

def imprimir(self):

print(self.nombre, "tiene depositado la suma de: ", self.monto)

class Banco:

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

self.cliente1 = Cliente("Juan")

self.cliente2 = Cliente("Ana")

self.cliente3 = Cliente("Diego")

def operar(self):

self.cliente1.depositar(100)

self.cliente2.depositar(150)

self.cliente3.depositar(200)

self.cliente3.extraer(150)

def depositos\_totales(self):

total = self.cliente1.retornar\_monto() + self.cliente2.retornar\_monto() + self.cliente3.retornar\_monto()

print("El total de dinero del banco es: ", total)

self.cliente1.imprimir()

self.cliente2.imprimir()

self.cliente3.imprimir()

# bloque principal

banco1 = Banco()

banco1.operar()

banco1.depositos\_totales()

* **HERENCIA:**

La herencia crea nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su “**superclase**” o “**clase padre**” y además se le podrán añadir otros atributos y métodos propios.

* 1. **CLASE PADRE:** Clase de la que desciende o deriva una clase.
  2. **SUBCLASE:** Clase descendiente de otra.
  3. **LA SINTAXIS PARA UNA DEFINICIÓN DE CLASE DERIVADA SE VE ASÍ:**

class ClaseDerivada(ClaseBase):

<declaración-1>

.

.

.

<declaración-N>

**EJERCICIOS:**

**EJERCICIO 1:**

Plantear una clase Persona que contenga dos atributos: nombre y edad. Definir como responsabilidades la carga por teclado y su impresión. En el bloque principal del programa definir un objeto de la clase persona y llamar a sus métodos. Declarar una segunda clase llamada Empleado que herede de la clase Persona y agregue un atributo sueldo y muestre si debe pagar impuestos (sueldo superior a 3000). También en el bloque principal del programa crear un objeto de la clase Empleado.

class Persona:

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

self.nombre = input("Ingrese el nombre: ")

self.edad = int(input("Ingrese la edad: "))

def imprimir(self):

print("Nombre: ", self.nombre)

print("Edad: ", self.edad)

class Empleado(Persona):

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

super().\_\_init\_\_()

self.sueldo = float(input("Ingrese el sueldo: "))

def imprimir(self):

super().imprimir()

print("Sueldo: ", self.sueldo)

def paga\_impuestos(self):

if self.sueldo > 3000:

print("El empleado debe pagar impuestos")

else:

print("No paga impuestos")

# bloque principal

persona1 = Persona()

persona1.imprimir()

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

empleado1 = Empleado()

empleado1.imprimir()

empleado1.paga\_impuestos()

EXPLICACIÓN:

La clase Persona no tiene ninguna sintaxis nueva, como vemos definimos sus métodos **\_\_init\_\_** e imprimir, y sus dos atributos nombre y edad:

class Persona:

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

self.nombre = input("Ingrese el nombre: ")

self.edad = int(input("Ingrese la edad: "))

def imprimir(self):

print("Nombre: ", self.nombre)

print("Edad: ", self.edad)

En el bloque principal creamos un objeto de la clase Persona:

# bloque principal

persona1 = Persona()

persona1.imprimir()

La herencia se presenta en la clase Empleado, en la declaración de la clase indicamos entre paréntesis el nombre de la clase de la cual hereda:

class Empleado(Persona):

En el método **\_\_init\_\_** de la clase Empleado primero llamamos al método **\_\_init\_\_** de la clase padre y luego cargamos el sueldo:

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

super().\_\_init\_\_()

self.sueldo = float(input("Ingrese el sueldo: "))

Lo mismo sucede con el método imprimir donde primero llamamos al imprimir de la clase padre y luego mostramos el sueldo del empleado:

def imprimir(self):

super().imprimir()

print("Sueldo: ", self.sueldo)

La tercera funcionalidad es:

def paga\_impuestos(self):

if self.sueldo > 3000:

print("El empleado debe pagar impuestos")

else:

print("No paga impuestos")

En el bloque principal de nuestro programa definimos un objeto de la clase Empleado y llamamos a sus funcionalidades:

empleado1 = Empleado()

empleado1.imprimir()

empleado1.paga\_impuestos()

**EJERCICIO 2:**

Declarar una clase Cuenta y dos subclases CajaAhorra y PlazoFijo. Definir los atributos y métodos comunes entre una caja de ahorro y un plazo fijo y agruparlos en la clase Cuenta.

Una caja de ahorro y un plazo fijo tienen un nombre de titular y un monto. Un plazo fijo añade un plazo de imposición en días y una tasa de interés. Hacer que la caja de ahorro no genera intereses.

En el bloque principal del programa definir un objeto de la clase CajaAhorro y otro de la clase PlazoFijo.

class Cuenta:

def \_\_init\_\_(self, titular, monto):

self.titular = titular

self.monto = monto

def imprimir(self):

print("Titular: ", self.titular)

print("Monto: ", self.monto)

class CajaAhorro(Cuenta):

def \_\_init\_\_(self, titular, monto):

super().\_\_init\_\_(titular, monto)

def imprimir(self):

print("Cuenta de caja de ahorro")

super().imprimir()

class PlazoFijo(Cuenta):

def \_\_init\_\_(self, titular, monto, plazo, interes):

super().\_\_init\_\_(titular, monto)

self.plazo = plazo

self.interes = interes

def imprimir(self):

print("Cuenta de plazo fijo")

super().imprimir()

print("Plazo en días: ", self.plazo)

print("Interés: ", self.interes)

self.ganancia\_interes()

def ganancia\_interes(self):

ganancia = self.monto \* self.interes / 100

print("Importe del interés: ", ganancia)

# bloque principal

Cajaahorro = CajaAhorro("Juan", 2000)

cajaahorro.imprimir()

plazofijo = PlazoFijo("Diego", 10000, 30, 0.75)

plazofijo.imprimir()