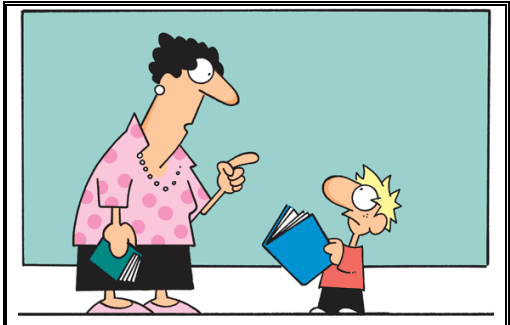
POO - Ejercicios Resueltos y para Resolver

Primeros Pasos

(Programas que el alumno resuelve, incluye en la carpeta y/o explica en clase)

PRIMERO LO PRIMERO: Una Clase es un tipo de dato, tal como lo es un numero entero **(int)** o un numero real **(float)**; pero a este tipo de dato, lo definimos nosotros como programadores.

Además, a este tipo de dato **(clase)**, le podemos incluir funciones a las que llamaremos **métodos**, para que hagan cosas con el contenido o datos que contendrá, a los que llamaremos **atributos**.



Leer y Estudiar es la Forma de Actualizar e Instalar Nuevo Software en el Cerebro

Luego, cuando definimos una variable de esta clase (de este tipo de dato) decimos que estamos instanciando la clase, es decir, definimos un objeto de la clase que creamos nosotros como programadores. Esto último, dicho de otra forma será: Un objeto es la instancia de una clase y podemos definir tantas instancias **(objetos)** de una misma clase como sean necesarios (igual que podemos definir tantas variables del tipo "**int**", como necesitemos).

Y algo que siempre debes recordar, si dos objetos hacen cosas diferentes (por que tienen métodos diferentes) se dice que tienen comportamientos diferentes.

Para definir una clase, comenzamos con la palabra reservada **class** y a continuación el **nombre** que queramos darle.

Ver ejemplo a continuación:

Nro	Parte	Enunciados / Ejemplos	Fin
1)	a)	**Ejemplo:** Definir la Clase Persona, en la que su único atributo sea el nombre y posea dos métodos: **Inicializar** que recibe el nombre de la persona como parámetro y el método **Mostrar**, que simplemente muestra el nombre que ya tiene la persona.	29/12

```
# -*- coding: utf-8 -*-
"""
Created on 2022 Abril 17
@author: Castelli Horacio
"""
print("----------------------------------------------------------")
print(" Definición Clase Persona - Iniciando POO")
print("----------------------------------------------------------\n")

# Definiendo la Clase
class Persona:
    def Inicializar(self, nom):
        self.nombre=nom

    def Mostrar(self):
        print("Nombre",self.nombre)

# Inicia Programa
P_01=Persona()  # Defino variable P_01 de la Clase Persona
P_01.Inicializar("Pedro") # Uso el método Inicializar
P_01.Mostrar() # Uso el método Mostrar

P_02=Persona()
P_02.Inicializar("Roxana")
P_02.Mostrar()

# Termina Programa
print("\n----------------------------------------------------------")
print("Programa Terminado.")
```

Programa	POO_Codigo/00_POO_Iniciando_01_Class_Persona_definicion_y_Uso_001.txt

	b)	Agregar al programa anterior, una función que permita leer un nombre y entregárselo al método que inicializa el objeto. Si no te imaginas como hacerlo, a continuación te dejo la dirección del programa para que lo analices.	

Programa	POO_Codigo/00_POO_Iniciando_01_Class_Persona_definicion_y_Uso_002.txt

	c)	Ahora, podrías crear un vector que contenga Empleados (Donde cada elemento del vector será un objeto de la Clase Persona). Y el vector contenga cierta cantidad de Empleados (cantidad definida como una constante del programa). Como vimos anteriormente, cada objeto (instancia de la clase) tiene una función que lo ayuda entregándole el nombre (el nombre que le tocará como persona). Luego para terminar el programa, recorrer el vector, haciendo que cada objeto muestre su nombre de persona. Como siempre, si no entiendes o no te sientes capas de enfrentar este desafío, a continuación	

dispones del programa para que lo analices.

Programa	POO_Codigo/00_POO_Iniciando_01_Class_Persona_definicion_y_Uso_003_Vector.txt

Hasta este punto, todo esta bien programado, pero hay que tener en cuenta, que si nuestro programa fuera llamado nuevamente a ejecutarse, las porciones de memoria que se asignaron para cada elemento del vector, pueden haber quedado reservadas, aun cuando el programa que las reservo, haya finalizado su proceso. Y ahora, que se ejecuta nuevamente, se reservarán otra vez (ignorando que ya están reservadas de la vez anterior), mermando la cantidad disponible de memoria, ya que ahora estarán dos veces reservadas (ocupando el doble de espacio), restando memoria a los programas que se ejecutarán a continuación. Por este motivo, siempre es una buena práctica liberar la memoria que fué reservada dinámicamente (en cualquier momento de la ejecución del programa), al finalizar la ejecución del programa que las reservo, sin necesidar de esperar al recolector de Basura.

A continuación se pide que agregues una nueva funcionalidad al programa, en la que deberás eliminar todos los elementos que contenga el vector, liberando así la memoria usada. Este proceso deberá ser ejecutado justo antes de terminar el programa.

Programa	POO_Codigo/00_POO_Iniciando_01_Class_Persona_definicion_y_Uso_004_Elimina_Vector.txt

2)

Antes de continuar, debes conocer dos funciones (métodos) muy útiles en la POO. Estas son el Constructor y Destructor. No todos los lenguajes las poseen, pero aun cuando no las tengan, siempre es posible implementarlas, aunque no sean automáticas y deban ser ejecutadas mediante un llamado a la función.

Constructor: Un constructor es un método que se ejecuta automáticamente al instanciar un objeto de la clase (Cuando lo declaras o instancias). El constructor tiene como finalidad la inicialización de las variables de la clase y posiblemente ejecutar (cuando haga falta) algunos métodos de la clase (preparando todo para que funcione bien). Es decir, dejar todo preparado para que comience a funcionar.

En PYTHON el constructor se lo llama **"_ _*init*_ _( )"** (Pero en C y C++ llevará el mismo nombre de la clase) Este método **(función)** es llamado automáticamente por Python cada vez que instanciamos (declaramos) un objeto de la clase que lo contiene (siempre y cuando lo hayamos definido en nuestro programa). Tal como se dijo un poco mas arriba, el constructor crea el estado inicial del objeto, con el conjunto mínimo de parámetros que necesita para existir.

Destructor: El destructor es un método de la clase que se usa para destruir objetos (liberar memoria y recursos) de la clase que lo define. Generalmente no tiene parámetros de entrada ni valor de retorno, ya que usa los atributos (variables) de su propia clase. No todos los lenguajes de programación lo soportan.

Si lo soportan se ejecuta solo y automáticamente el terminar el proceso, de lo contrario, siempre es posible definirlo y llamarlo manualmente justo antes de terminar el proceso.

Python cuenta con destructores, aunque no son tan comunes ni tan utilizados como en otros lenguajes. En Python, el destructor es un método especial llamado "__del__" que se define dentro de una clase y se ejecuta cuando el objeto es destruido, es decir, cuando ya no tiene referencias en el programa y el recolector de basura decide liberarlo. Aquí tienes un ejemplo sencillo:

```
class MiClase:
    def __init__(self, nombre):
        self.nombre = nombre
        print(f"Objeto {self.nombre} creado.")

    def __del__(self):
        print(f"Objeto {self.nombre} destruido.")

# Crear un objeto de la clase
obj = MiClase("Ejemplo")

# El destructor se llamará automáticamente cuando el objeto sea eliminado
del obj  # Aquí se llama al destructor explícitamente, pero no seria necesario
```

Ver ejemplo completo:	POO_Codigo/00_POO_Iniciando_00_Ejemplo_Constructor_Destructor_001.txt

Es verdad que en la mayoría de los casos, se recomienda no llamar al destructor manualmente y dejar que Python gestione la memoria automáticamente. Sin embargo, si requieres una limpieza más predecible, o necesitas liberar recursos específicos (por ejemplo, cerrar archivos o conexiones de red) en un momento preciso, puedes generar funciones que lo hagan cuando lo requieras o llamar (invocar) en el código al destructor.
Debo mencionar que Python dispones de otros métodos más robustos para la liberación de recursos, pero en este momento no los utilizaremos.

INTERESANTE: C y C++ si lo soportan y su nombre será el nombre de la clase precedido por el caracter virgulilla **"~"**, el que puedes escribir con la secuencia de teclas **Alt+126**

a) Crear e instancia la Clase Persona, que administre el listado de los alumnos (también son personas!!!) de un curso (nombre y edad). El constructor de la clase, debe recibir como parámetro un número que indicará la cantidad de alumnos que serán cargados al iniciar el programa. Mediante un método, muestre el listado completo de alumnos que fue cargado. Al finalizar el programa (con el destructor)

		liberar completamente la memoria. Programa: POO_Codigo/00_POO_Iniciando_02_Class_Alumno_001_Constructor_y_Destructor.txt Analiza en el código ejemplo, que pasa cuando no se le da un parámetro al inicializar el objeto.	
	b)	En este paso usaremos archivos de texto (lectura y escritura), si no recuerdas como usarlos, acá te dejo un resumen, dale un vistazo y continúa. Repaso Manejo de archivos: Archivos_txt_Como_Usarlos.pdf Configuración de Spyder: Configurar_Spyder_para_Archivos_txt.pdf Aprovecha que ya sabes usar un constructor, y realiza una pequeña modificación para que, si en la inicialización del objeto, en vez de la cantidad de alumnos que debe cargar, le pasas como parámetro el nombre del archivo para que cargue los datos de los alumnos desde ese archivo (en los ejemplos usaremos "**Alumnos_01.txt**"). Y por ultimo, Usa el destructor para Actualizar el archivo (Grabándolo nuevamente) y liberes la memoria (eliminando todos los elementos del vector). Si te animas, completa este programa, acá te dejo el modelo que puedes usar. **Analízalo**. Programa: POO_Codigo/00_POO_Iniciando_02_Class_Alumno_002_Archivos.txt	
	c)	Agregar los métodos necesarios a la clase (Un menú de opciones), para que el usuario pueda a voluntad, agregar alumnos y/o eliminar un alumno del listado. Muestre el listado actualizado y/o actualice (grabe o regrabe) los datos en el archivo "**Alumnos_01.txt**".	
	d)	Si todavía no lo has programado, modifica el destructor, para que antes de liberar la memoria, grabe los datos en el mismo archivo desde donde lo leyó o deberá estar grabado.	
	e)	Agregar un nuevo dato en la clase persona. El nuevo dato será el número de legajo. Este dato debe estar representado por numero entero entre 1 y 1000, sin repeticiones, aunque los valores pueden no estar correlativos, pero siempre serán crecientes (de menor a mayor) es decir por ejemplo el primer alumno puede tener como numero de legajo el 13, el segundo el 30, el tercero 31 y el cuarto 50, etc. Este nuevo dato debe encontrarse en el archivo, Vector (tabla de la clase) y se manejado por los métodos de la clase, etc.	
3)			
	a)		
	b)		
	c)		
	d)		
4)			
	a)		
	b)		
	c)		
	d)		
5)			
	a)		
	b)		
	c)		
	d)		
6)	a)		
	b)		
	c)		
	d)		
	e)		
7)	a)		
	b)		
	c)		
	d)		
	e)		

La programación orientada a objetos incluye cuatro pilares principales:

1. Abstracción

La abstracción oculta al usuario la funcionalidad interna de una aplicación. El usuario puede ser el cliente final u otros desarrolladores.

Podemos encontrar **abstracción** en nuestra vida cotidiana. Por ejemplo, sabes cómo usar tu teléfono, pero probablemente no sepas exactamente lo que ocurre dentro de él cada vez que abres una aplicación.

Otro ejemplo es el propio Python. Sabes cómo usarlo para construir software funcional, y puedes hacerlo aunque no entiendas el funcionamiento interno de Python.

Aplicar lo mismo al código permite reunir todos los objetos de un problema y **abstraer la** funcionalidad estándar en clases.

2. Herencia

La herencia nos permite definir múltiples **subclases** a partir de una clase ya definida.

El propósito principal es seguir el principio DRY. Podrás reutilizar mucho código implementando todos los componentes compartidos en **superclases**.

Puedes pensar en ello como el concepto de **herencia genética** en la vida real. Los hijos (subclases) son el resultado de la herencia entre dos padres (superclases). Heredan todas las características físicas (atributos) y algunos comportamientos comunes (métodos).

3. Polimorfismo

El polimorfismo nos permite modificar ligeramente los métodos y atributos de las **subclases** previamente definidas en la **superclase**.

El significado literal es «**muchas formas**«. Esto se debe a que construimos métodos con el mismo nombre pero con diferente funcionalidad.

Volviendo a la idea anterior, los niños también son un ejemplo perfecto de polimorfismo. Pueden heredar un comportamiento definido **get_hungry()** pero de una manera ligeramente diferente, por ejemplo, tener hambre cada 4 horas en lugar de cada 6.

4. Encapsulación

La encapsulación es el proceso en el que protegemos la integridad interna de los datos en una clase.

Aunque no hay una declaración **privada** en Python, se puede aplicar la encapsulación mediante el uso de mangling en Python. Existen métodos especiales llamados **getters** y **setters** que nos permiten acceder a atributos y métodos únicos.

Imaginemos una clase **Humana** que tiene un único atributo llamado **_altura**. Este atributo solo se puede modificar dentro de ciertas restricciones (es casi imposible ser más alto que 3 metros).