***Relaciones entre Clases en Python***

Aunque algunas clases pueden existir aisladas, la mayoría no puede, y deben cooperar unas con otras. Las relaciones entre las clases expresan una forma de acoplamiento entre ellas. Según el tipo de acoplamiento que presentan las clases podemos distinguir distintos tipos de relaciones. Asociación - Agregación – Composición - Herencia

***Asociación***

* Conceptualmente la asociación en un diagrama de clases implica transitividad y bidirección de clases.
* Una instancia de la clase asociación representa una relación uno a uno.

***Agregación***

* Un objeto de una clase contiene como partes a objetos de otras clases.
* La destrucción del objeto continente no implica la destrucción de sus partes.
* Los tiempos de vida de los objetos continente y contenido no están acoplados.

***Composición***

* Un objeto de una clase contiene como partes a objetos de otras clases y estas partes están físicamente contenidas por el agregado.
* Los tiempos de vida de los objetos continente y contenido están estrechamente acoplados.
* La destrucción del objeto continente implica la destrucción de sus partes.

***Herencia***

* La herencia es el mecanismo que permite compartir automáticamente métodos y datos entre clases y subclases.
* Permite crear nuevas clases a partir de clases existentes programando solamente diferencias.
* Si en la definición de una clase indicamos que ésta deriva de otra, entonces la primera será tratada por el intérprete automáticamente como si su definición incluyese la definición de la segunda.
* Todas las clases en Python, derivan de object, por lo que disponen de los atributos y métodos de dicha clase.

***Herencia Múltiple***

* No todos los lenguajes de programación la implementan.
* Una subclase deriva de más de una clase base.
* Python provee un mecanismo de resolución de conflictos, denominado MRO (Method Resolution Order).
* MRO para la ejecución de métodos recorre el árbol de herencia de arriba hacia abajo, y al mismo nivel, de derecha a izquierda (denominada regla del diamante)

***Polimorfismo***

* la vinculación de un objeto es dinámica.
* Todos los métodos se vinculan a las instancias en tiempo de ejecución.
* Para llevar este tipo de vinculación, se utiliza una tabla de métodos virtuales por cada clase.
* El polimorfismo es la capacidad que tienen objetos de clases diferentes, a responder de forma distinta a una misma llamada de un método.
* El polimorfismo de subtipo se basa en la ligadura dinámica y la herencia.
* El polimorfismo hace que el código sea flexible y por lo tanto reusable.

Para determinar a qué clase pertenece un objeto pueden utilizarse las funciones: isinstance(x, Clase), donde x es una referencia a un objeto, Clase es el nombre de la clase de la que se quiere averiguar si un objeto es instancia o no, la función devuelve True o False, dependiendo si x es un objeto perteneciente a la clase Clase o no.

***Errores en un programa***

Tales errores se clasifican en tres tipos básicos:

* Errores de sintaxis: son errores donde el código no es válido para el compilador o intérprete.
* Errores en tiempo de ejecución: son errores donde un código sintácticamente válido falla, quizá debido a una entrada no válida.
* Errores semánticos: errores en la lógica del programa, el código se ejecuta sin problemas, pero el resultado no es el esperado.

Las excepciones son errores en tiempo de ejecución. Cuando se produce este error, el programa se detendrá y generará una excepción que luego se manejará para evitar que el programa se detenga por completo.

Python provee las instrucciones:

* **Assert:** El programador puede iniciar una afirmación de excepción con la sentencia assert. Si lo que se afirma se cumple, la ejecución continúa sin problemas. Si lo que se afirma no se cumple, la ejecución se interrumpe y se la lanza la excepción AssertionError.
* **Raise:** Lanzar en forma manual una excepción se hace con la instrucción raise
* **Bloque try-except-else-finally:** El bloque try-except puede controlar más de una excepción, por lo que el sub bloque except puede repetirse tantas veces como excepciones se quieran capturar y manejar

***Definiendo nuevas Excepciones***

Se pueden definir nuevas excepciones, extendiendo la clase BaseException o alguna de sus subclases.

***Consejos sobre excepciones:***

* sea específico sobre lo que captura.
* No atrape nada con lo que no pueda tratar.
* Si necesita tratar con múltiples tipos de excepciones, entonces tenga múltiples bloques except en el orden correcto
* Las excepciones personalizadas pueden ser muy útiles si se necesita almacenar información compleja o específica en instancias de excepción
* No cree nuevas clases de excepción cuando las integradas tengan toda la funcionalidad que necesita.
* No necesita lanzar una excepción tan pronto como se construye, esto facilitala predefinición de un conjunto de errores permitidos en un solo lugar.

***Testing***

Estos son programas que automáticamente ejecutan ciertas entradas. Se pueden correr estos programas de prueba en segundos y cubren más situaciones de entrada posibles que las que un programador pensaría en probar cada vez que cambia algo.

Cuatro razones por las que es necesario hacer pruebas de software:

* Para asegurarse de que el código funcione de la manera que el desarrollador cree que debería.
* Para garantizar que el código siga funcionando cuando se realizan cambios.
* Asegurarse de que el desarrollador entendió los requisitos.
* Para asegurarse que el código que se escribió tenga una interfaz que se pueda mantener.

***Mantras:***

1. "Escriba las pruebas primero"
2. "código no probado es código roto"

La metodología basada en pruebas tiene dos objetivos:

* Es asegurarse de que las pruebas realmente se escriben. Si la prueba ya está escrita antes de escribir el código, se sabrá exactamente cuando funciona y se sabrá en el futuro si alguna vez se rompe por un cambio que se haya hecho sobre el código.
* En segundo lugar, escribir pruebas primero obliga al programador a considerar exactamente cómo será el código. Dice qué métodos deben tener los objetos y cómo los atributos serán accedidos.

***Test de unidad***

Las pruebas unitarias se enfocan en probar la menor cantidad de código posible en cualquier prueba. Esta biblioteca proporciona varias herramientas para crear y ejecutar pruebas unitarias, siendo la más importante la clase TestCase.

TestCase proporciona un conjunto de métodos que permiten comparar valores, configurar pruebas y limpiar la memoria cuando ya se hayan terminado.

* Se pueden tener tantos métodos de prueba en una clase o subclase, TestCase, como sean necesarios.
* Cada prueba debe ser completamente independiente de otras pruebas.
* Los resultados de una prueba previa no deberían tener ningún impacto en la prueba actual.
* La clave para escribir buenas pruebas unitarias es mantener cada método de prueba lo más breve posible, probando una pequeña unidad de código en cada caso de prueba.

***Clase Base Abstracta***

* Permite construir una interfaz común a un conjunto de subclases.
* Se construye para reunir un conjunto de atributos comunes al conjunto de subclases.
* Es aquella que define una interfaz, pero no su implementación, de tal forma que sus subclases sobrescriban los métodos con las implementaciones correspondientes.
* Una clase abstracta no puede ser instanciada.
* Una clase en Python, es abstracta si al menos tiene un método abstracto.
* Las subclases que hereden de una clase base abstracta, que pretendan ser concretas, deben implementar los métodos abstractos, si no lo hacen se convierten automáticamente en clase abstracta.
* El a través del módulo abc (Abstract Base Class), Python define e implementa la abstracción de clases, mediante meta clases y decoradores.

***Interfaces***

* Una interfaz es un conjunto de firmas de métodos, atributos o propiedades eventos agrupados bajo un nombre común
* Funcionan como un conjunto de funcionalidades que luego implementan las clases.
* Las clases que implementan las funcionalidades quedan vinculadas por la o las interfaces que implementan.
* Las clases que implementan las interfaces son intercambiables con las interfaces, esto hace que a una referencia a la interface se le pueda asignar una referencia a un objeto de la clase que la implementa, la referencia a la interface tendrá acceso sólo a los métodos, atributos y propiedades declarados por la interface e implementados por la clase.
* Son parecidas a las clases abstractas, en el sentido en que no proveen implementación de los métodos que declaran.
* Se diferencian en que las clases que derivan de clases abstractas pueden no implementar los métodos abstractos, mientras que las clases que implementen una interfaz deben proveer implementación de todos los métodos de la interfaz.
* Al igual que las clases abstractas, son tipo referencia, pero no pueden crearse objetos directamente de ellas.
* Las clases pueden implementar cualquier número de interfaces.
* Una clase que implementa una interfaz, provee implementación a todos los métodos de la interfaz.
* En la POO se dice que las interfaces son funcionalidades y las clases representan implementaciones.
* Python no permite definir interfaces, es necesario incorporar una librería externa para emular el comportamiento de las interfaces que proveen otros lenguajes como Java y C#.
* Las interfaces no implican una sobrecarga en el procesamiento.
* Siempre que se prevea que dos o más clases no vinculadas por la herencia, harán lo mismo, es conveniente definir una interfaz con los comportamientos comunes.
* Pensar en la interfaz antes que en la clase en sí, es pensar en lo qué debe hacerse en lugar de pensar en cómo debe hacerse.
* Usar interfaces permite a posteriori cambiar una clase por otra que implemente la misma interfaz y poder integrar la nueva clase de forma mucho más fácil, sólo se debe modificar donde se instancian las clases, el resto del código queda igual,fomentando la reusabilidad del código.

***Diccionarios en Python***

Los diccionarios en Python, son estructuras de datos utilizadas para mapear claves a valores. Los valores pueden ser de cualquier tipo, inclusive otro diccionario, una lista, un arreglo.

***Archivos JSON***

* JSON (JavaScript Object Notation) es un formato de intercambio de datos basado en texto, se usa para el intercambio de datos en la web.
* JSON puede ser codificado y decodificado con Python.
* Los objetos de Python y los definidos por el programador pueden codificarse y decodificarse a través del lenguaje.
* Los datos en formato JSON pueden almacenarse en archivos, para luego recuperarlos y almacenar los datos en los atributos de las clases.
* los archivos JSON, constituyen un mecanismo de persistencia del estado de losobjetos.