Enfoque procedimental frente al enfoque orientado a objetos

el mundo de los datos y el mundo del código. El mundo de los datos está poblado con variables de diferentes tipos, mientras que el mundo del código está habitado por códigos agrupados en módulos y funciones.

El enfoque orientado a objetos sugiere una forma de pensar completamente diferente. Los datos y el código están encapsulados juntos en el mismo mundo, divididos en clases.

Jerarquía de clases.

* De manera descendente.

Que es una clase: es un conjunto de objetos

Que es un objeto: es un pedazo de código relacionado a una clase.

* Es una encarnación de los requisitos, rasgos y cualidades asignados a una clase específica
* Las clases forman unas jerarquías.
* Un objeto que pertenece a una clase especifica. Pertenece a todas las superclases al mismo tiempo.
* Ò cualquier objeto que pertenezca a una superclase puede no tener a ninguna subclase.
* Cada subclase es mas especializada.
* La superclase es mas general.

Una clase bien definida es una herramienta para crear nuevos objetos

HERENCIA.

Cualquier objeto vinculado a un nivel específico de una jerarquía de clases **hereda todos los rasgos (así como los requisitos y cualidades) definidos dentro de cualquiera de las superclases**.

QUE CONTIENE UN OBJETO.

* Nombre.
* Conjunto de propiedades individuales.
* Conjunto de habilidades para realizar actividades específicas.

Describir:

* Un sustantivo: probablemente se está definiendo el nombre del objeto.
* Un adjetivo: probablemente se está definiendo una propiedad del objeto.
* Un verbo: probablemente se está definiendo una actividad del objeto.

 Los objetos están equipados con:

* Un **nombre** que los identifica y nos permite distinguirlos.
* Un conjunto de **propiedades** (el conjunto puede estar vacío).
* Un conjunto de **métodos** (también puede estar vacío).

**Instanciación:**

La clase en sí misma no puede crear un objeto: debes crearlo tu mismo

**PASS:**

La palabra clave reservada pass llena la clase con nada. No contiene ningún método ni propiedades.

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**PILA**

es una estructura desarrollada para almacenar datos de una manera muy específica.

**NOMBRE ALTERNATIVO**:::: UEPS.

PUSH: Para colocar un nuevo elemento

POP: para retirar un elemento

**Constructor.**

Es una función específica cuyo nombre es la palabra reservada \_\_init\_\_

* Esta **obligado a tener el parámetro**self
* **Pudiera (pero no necesariamente) tener mas parámetros** que solo self
* **Se puede utilizar para configurar el objeto**

**Dentro de la función constructor**

* Cualquier cambio que realices dentro del constructor que modifique el estado del parámetro self se verá reflejado en el objeto recien creado.

Cuando cualquier componente de la clase tiene un **nombre que comienza con dos guiones bajos (**\_\_**), se vuelve privado**, esto significa que solo se puede acceder desde dentro de la clase. ( encapsulamiento ).

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////

* . Una **pila** es un objeto diseñado para almacenar datos utilizando el modelo **LIFO**. La pila normalmente realiza al menos dos operaciones, llamadas **push()** y **pop()**.
* Un **método** de clase es en realidad una función declarada dentro de la clase y capaz de acceder a todos los componentes de la clase.
* La parte de la clase en Python responsable de crear nuevos objetos se llama **constructor** y se implementa como un método de nombre \_\_init\_\_.

Variables de instancia.

La palabra *instancia* sugiere que están estrechamente conectadas a los objetos (que son instancias de clase), no a las clases mismas.

Variables de clase:

Una variable de clase es **una propiedad que existe en una sola copia y se almacena fuera de cualquier objeto**.

* Una variable de clase **siempre presenta el mismo valor** en todas las instancias de clase (objetos).

 Una función llamada hasattr() se puede utilizar para determinar si algún objeto o clase contiene cierta propiedad especificada..

Métodos a detalle.

* **método es una función que está dentro de una clase**
* un **método está obligado a tener al menos un parámetro** (no existen métodos sin parámetros
* El parámetro self es usado **para obtener acceso a la instancia del objeto y las variables de clase**.
* El parámetro self también se usa **para invocar otros métodos desde dentro de la clase**.

Cada clase de Python y cada objeto de Python está pre-equipado con un conjunto de atributos útiles que pueden usarse para examinar sus capacidades.

* Uno de ello ses la propiedad = \_\_dict\_\_. (diccionario)
* \_\_name\_\_ (nombre de la clase)
* Type (tipo de clase)
* \_\_module\_\_ (almacena el nombre del modulo que contiene la definición de la clase)
* \_\_bases\_\_ (una tupla que contiene clases)

······························································································································································

Reflexión e introspección:

* **Introspección**, que es la capacidad de un programa para examinar el tipo o las propiedades de un objeto en tiempo de ejecución.
* **Reflexión**, que va un paso más allá, y es la capacidad de un programa para manipular los valores, propiedades y/o funciones de un objeto en tiempo de ejecución.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

**HERENCIA.**

La herencia es una práctica común (en la programación de objetos) de **pasar atributos y métodos de la superclase (definida y existente) a una clase recién creada, llamada subclase**.

* En otras palabras, la herencia es **una forma de construir una nueva clase, no desde cero, sino utilizando un repertorio de rasgos ya definido**.
* Gracias a eso, es posible **construir clases más especializadas (más concretas)** utilizando algunos conjuntos de reglas y comportamientos generales predefinidos.
* FUNCION (ISSUBCLASS) “Identifica una relacion entre 2 clases”
* (ISINSTANCE) “IDENTIFICA SI UN OBJETO ES DE CIERTA CLASE O NÓ. ”
* OPERADOR (((( IS )))) “IS” “Verifica si 2 variables se refieren al mismo objeto”

Como encuentra propiedades y métodos.

* Una clase hereda los métodos de las otras clases

super().\_\_init\_\_(name) “para tener acceso a la super clase dentro de otra clase.

* PLIMORFISMO “La sub clase puede modificar el comportamiento de la superclase” “convertir un objeto de la clase padre a la clase hija”
* Def Do \_it (self):
* Herencia simple y multiple:
* una solo clase siempre es mas fácil.
* La herencia múltiple siempre es arriesgada, ya que tienes muchas más oportunidades de cometer un error al identificar estas partes de las superclases que influirán efectivamente en la nueva clase.
* La herencia múltiple viola el **principio de responsabilidad única**
* ya que forma una nueva clase de dos (o más) clases que no saben nada una de la otra.

·······························································································································································

PROBLEMA DIAMANTE