UNIDAD 1

Programación Imperativa:

Algoritmos que se describen en base a procesos o funciones y éstos como una secuencia de tareas a realizar por la computadora

Programación Modular:

Permite dividir el programa en módulos autónomos que se pudieran programar, verificar y modificar individualmente

Tipo de Datos Abstracto:

Un tipo de datos definido por el programador, posee un conjunto de operaciones abstractas sobre objetos de ese tipo. El encapsulamiento de los objetos de ese tipo, permite que el usuario final del tipo no pueda manipular esos objetos excepto a través del uso de las operaciones definidas.

Durante el proceso de desarrollo de requerimiento de un sistema de software es necesario identificar los **objetos del problema** y caracterizarlos a través de sus atributos:

- Un obj del problema es una entidad, física o conceptual, caracterizada a través de atributos y comportamientos. (Identifica)
- Cuando el sistema de software esta en ejecución, se crean objetos de software, que es un modelo el cual representa un objeto del problema y tiene, una identidad, estado interno y recibe mensajes a los que responde ejecutando un servicio. (Abstrae)

Algunas características del diseño O.O:

- El objeto combina los datos (atributos del objeto) con los procedimientos u operaciones (métodos) que actúan sobre dichos datos. Los objetos interactúan entre sí enviando mensajes.
- Los métodos son muy similares a los procedimientos de la programación imperativa tradicional y los mensajes se podrían pensar como invocaciones a esos procedimientos.
- Se pueden agrupar características comunes de un conjunto de objetos en clases, a través de un proceso de abstracción. Los descendientes de estas clases se construyen por medio del mecanismo de subclasificación, permitiendo que sean heredados.
- El principio fundamental del paradigma de la POO es construir un sistema de software en base a las entidades de un modelo elaborado a partir de un proceso de abstracción y clasificación. El desarrollo de software implica las etapas de: requerimientos, análisis, diseño, implementación y prueba.

UML: Es un lenguaje que permite la visualización, especificación y documentación de sistemas. No es una metodología sino una notación, que aglutina distintos enfoques de orientación a objetos.

- **Diagramas de Modelado Estructurado:** Son diagramas estructurales o de clases que definen la arquitectura estática de un modelo, definen los bloques de construcción básica de un modelo: materiales generales, clases y tipos que se usan para construir un diagrama completo. Describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas
- **Diagramas de Modelado de Comportamiento**: Capturan las interacciones y el estado instantáneo dentro de un modelo mientras se “ejecuta” a través del tiempo.
    - **Diagrama de Comunicación**: Muestran las interacciones entre objetos en la estructura espacial estática, que permite la colaboración entre objetos. El tiempo no se representa de manera explícita, por lo tanto, los mensajes se numeran para indicar el orden de los envíos.
    - **Diagramas de Secuencia**: Estos diagramas muestran interacciones entre objetos según un punto de vista temporal. Muestra la cronología de los envíos de mensajes. Un objeto se

representa por un rectángulo y el tiempo de vida se representa por una barra vertical llamada línea de vida de los objetos

Objetos en UML: Un obj se representa bajo la forma de un rectángulo, el nombre se subraya, o bien usando un nombre genérico mediante ":"; y un Actor es una entidad externa al propio sistema, pero que necesita intercambiar información con él.

Fragmentos en UML:

Un fragmento combinado es una o más secuencias de procesos incluidas en un marco y ejecutadas bajo circunstancias específicas.

- Fragmento de Alternativa("alt"): modela la elección de una interacción, entre objetos, a través de una condición de guarda, es decir, modela estructuras condicionales del tipo if,then,else.
- Fragmento de iteración("loop"): el fragmento incluye un conjunto de mensajes que se ejecutan múltiples veces, según lo indique la condición de guarda

Objetos en UML con estereotipo:

- Objeto de Interfaz (Boundary): representa un elemento (ejemplo un formulario web) con el cual interactúa el usuario (actor). A menudo representan una abstracción de una ventana, un formulario, una interfaz de comunicación.
- Objetos de Control (Controllers): se ocupa de organizar y controlar la lógica requerida para alcanzar el objetivo de una determinada funcionalidad. Media entre los objetos de interfaz y los de entidad.
- Objeto de Entidad (Entity): para cada uno de los objetos (entidades) del dominio requeridos para realizar la funcionalidad. Modelan información asociada a algún fenómeno o concepto, como persona o un objeto.

Elementos básicos de la O.O(Objetos, Clases, Métodos y Mensajes, Herencia):

Objetos:

Un objeto es una unidad atómica que encapsula estado y comportamiento, de manera que la encapsulación de un objeto permite una alta cohesión y un bajo acoplamiento. Los objetos son entidades que tienen atributos(datos) y comportamiento particular(procedimientos).

Atributos de un objeto: Los atributos describen la abstracción de características individuales que posee un objeto.

Comportamientos: Los comportamientos de un objeto representan las operaciones que pueden ser realizadas por un objeto.

Objeto=Estado+Comportamiento+Identidad

- **Estado** agrupa los valores instantáneos de todos los atributos de un objeto. El estado evoluciona con el tiempo.
- **Comportamiento** describe las acciones y reacciones de ese objeto. El estado y el comportamiento están relacionados ya que el comportamiento en un instante dado depende del estado actual, y el estado puede ser modificado por el comportamiento
- **Identidad** permite distinguir los objetos de forma no ambigua, independientemente de su estado. Esto permite distinguir dos objetos en los que todos los valores de los atributos son idénticos.

Clases:

Una clase es una descripción de un conjunto de objetos, ya que consta de comportamientos y atributos que resumen las características comunes del conjunto. Una clase abstrae las características de un conjunto de objetos con comportamientos similares. Es una de las ventajas de la POO, ya que definir clases significa colocar código reutilizable en un depósito común, en lugar de redefinirlo cada vez que se necesite.

Ventajas básicas:

Se protegen los datos de accesos indebidos, el acoplamiento entre clases se disminuye y favorece la modularidad, y el mantenimiento.

Los objetos que responden a la especificación de una clase, se llaman **instancias de una clase** y la instanciación es la operación que se realiza para obtener un objeto de una clase. Relaciona una clase y un objeto. (Cada objeto es instancia de una clase)

Visibilidad de los atributos:

Los atributos de una clase no deben ser manipulables directamente por el resto de objetos, por eso existen distintos niveles de encapsulación o niveles de visibilidad como: Atributo o método público, A/M protegido, A/M privado. (Indica si el atributo o comportamiento puede ser accedido desde otras clases)

Métodos y Mensajes:

- Los objetos tienen la posibilidad de actuar y esto ocurre cuando un objeto recibe un mensaje, que es una solicitud que pide al objeto que se comporte de manera determinada. Donde cada objeto, recibe, interpreta y responde a mensajes enviados por otros objetos.
- Los comportamientos u operaciones que caracterizan un conjunto de objetos, residen en la clase y se llaman métodos, que son el código que se ejecuta para responder a un mensaje, y el mensaje se considera como la llamada o invocación a un método.
- Los métodos proporcionan la única forma de modificar los datos de un objeto.

Variables de Instancia:

Se denominan variables o atributos de instancia a aquellos atributos cuyo valor cambian para cada objeto de la clase. Se usan para guardar los atributos de un objeto particular. Es un elemento de información que define un atributo de un objeto en particular.

Variables de Clase:

Son aquellos atributos que tienen el mismo valor para cada objeto de la clase. Si el valor de la variable de clase es cambiado para una instancia, el mismo cambia para todas las instancias de la clase y subclase. Representa un área de memoria compartida por todos los objetos de la clase. Es un elemento de información que define un atributo de toda una clase, la variable se aplica para la clase por sí misma y para todas sus instancias.

Herencia:

- **Generalización:** Consiste en factorizar los elementos comunes (atributos, métodos y restricciones) de un conjunto de clases en una clase más general llamada “superclase”. (de abajo hacia arriba)
- **Especialización:** Permite capturar las particularidades de un conjunto de objetos no discriminados por las clases ya identificadas, de forma que las nuevas características se representan por una nueva clase, que es una subclase de las clases ya existentes. (Proceso de arriba hacia abajo)

Una **superclase** es una abstracción de sus subclases, y los árboles de clases no crecen a partir de una raíz, sino que parten de las hojas porque estas pertenecen al mundo real, mientras que los niveles superiores son abstracciones para ordenar y comprender.

Subclase: Identifica el comportamiento de un conjunto de objetos que hereda las características de la clase padre y adicionan algunas específicas que ésta no posee.

Clase Abstracta: Una clase es abstracta cuando no existe un objeto que sea instancia directa de ella, pero sí existe una subclase de ella que es instanciable. Generalmente se utilizan para resumir los comportamientos comunes a un conjunto de subclases.

Clase Concreta: Una clase concreta es aquella que es instanciable, es decir que existe al menos un objeto de esa clase.

Herencia Simple: Con esta una subclase puede heredar datos y métodos de una única clase y también puede añadir nuevo comportamiento.

Herencia Múltiple: Con esta una subclase puede adquirir los datos y métodos de más de una clase padre.

En resumen: Los elementos del mundo a ser modelado y su comportamiento son transformados en objetos. Los objetos con comportamientos comunes son organizados en clases. Las clases se organizan en jerarquías y los mecanismos de herencia proporcionan a cada subclase los métodos y datos de la clase padre.

Relaciones entre clases:

Las relaciones muestran el **acoplamiento** de las clases.

El **acoplamiento** entre clases es el número de clases con las que una clase concreta está relacionada (acoplada). Una clase está acoplada si sus objetos lo están. Un objeto está acoplado con otro si uno de ellos actúa sobre el otro.

Asociación:

La relación de asociación representa una conexión semántica bidireccional entre dos clases. Una asociación es una abstracción de los vínculos que existen entre objetos instancias de las clases asociadas. La asociación no es contenida por las clases, ni subordinada a las clases asociadas; es el reflejo de una conexión que existe en el ámbito de la aplicación.

En **UML,** el número de instancias (cardinalidad) que participan en la relación, y se anota en cada extremo de la relación, y se llama multiplicidad.

En algunas situaciones es necesario agregar información que es propia de la relación y no de las clases:

- Clase Asociación: Para una dupla o par de objetos, existe un único objeto asociado a la clase respectiva. Principio Fundamental: Una dupla de objetos, instancias de cada una de las clases que participan en la asociación, se relaciona con una única instancia de la clase asociación, sin importar la multiplicidad en ambos extremos. Una instancia de la clase asociación representa una relación uno a uno
- Clase que modela la asociación: Para una dupla o par de objetos, existe más de un objeto asociado a la clase respectiva, donde por lo tanto existe una relación entre 2 clases, que se modela a través de otra. En el contexto planteado, se presenta una multiplicidad al momento de mostrar que tipo de relación, por ejemplo, mensualmente, anualmente, etc.

Reutilización y Extensión:

Se denomina **reutilización** al uso de clases u objetos desarrollados y probados en un determinado contexto, para incorporar esa funcionalidad en una aplicación diferente a la de origen. La forma más simple de reutilizar una clase es simplemente haciendo una nueva clase que la contenga. Esto es posible a través de la composición. En cambio, la **extensión** se basa en aprovechar las clases desarrolladas para una aplicación, utilizándose para la construcción de nuevas clases, en la misma u otra aplicación. Un caso típico de extensión se logra a través de la herencia.

Agregación:

Es una asociación no simétrica (todo/parte) en la que una de las clases cumple un papel predominante respecto de la otra. Consiste en definir como atributos de una clase a objetos de otras clases ya definidas. La agregación se utiliza para modelar la relación todo-parte (continente-contenido) y se describe mediante la relación tiene-un o parte-de.

- Agregación: Ocurre cuando un objeto contiene como partes a objetos de otras clases., pero de tal modo que la destrucción del objeto continente no implica la destrucción de sus partes. En este caso se dice que el objeto continente o contenedor incluye referencias a objetos de otras clases y que los objetos contenidos pueden existir independientemente del objeto contenedor.
    - Los tiempos de vida de los objetos continente y contenido no están estrechamente acoplados, de modo que se pueden crear y destruir instancias de cada clase independientemente.
    - No hay restricciones en la multiplicidad del agregado u objeto continente.

- Composición: Las composiciones generan una relación de existencia entre el todo y cada una de sus partes. Un objeto de una clase contiene como partes a objetos de otras clases y estas partes están físicamente contenidas por el agregado.
    - Los objetos agregados no tienen sentido fuera del objeto resultante.
    - Los tiempos de vida de los objetos continente y contenido están estrechamente acoplados
    - La destrucción del objeto continente implica la destrucción de sus partes

Criterios para detectar una agregación:

Los objetos de una clase están subordinados a los objetos de otra clase, una acción sobre objetos de una clase implica una acción sobre los objetos de otra clase, y los valores de los atributos de un objeto de una clase, se propagan en los valores de los atributos de un objeto de otra clase.

Polimorfismo: Capacidad que tienen objetos de clases diferentes, relacionados mediante la herencia, a responder de forma distinta a una misma llamada de un método.

- Fomenta la extensibilidad del software:
    - Software escrito en forma independiente del tipo de los objetos a los cuales los mensajes son enviados.
    - Nuevos tipos de objetos, que pudieran responder a mensajes existentes, pueden ser agregados en dicho sistema sin modificar el sistema base.

Encapsulamiento: Término formal que describe al conjunto de métodos y de datos de un objeto de manera tal que el acceso a los datos se permite solamente a través de los métodos propios del objeto.

Abstracción: A partir de un conjunto de objetos, se piensa en los comportamientos comunes de los mismos para situarlos en superclases, las cuales constituyen un depósito para elementos comunes y reutilizables.

Persistencia:

Designa la capacidad de un objeto de trascender el tiempo o el espacio. La persistencia se refiere a la permanencia de un objeto, es decir, al tiempo en que se le asigna espacio y permanece accesible. Un objeto persistente conserva su estado en un sistema de almacenamiento permanente (pasivación del objeto) y el objeto puede ser reconstruido (activación del objeto) por otro proceso y se comportará exactamente como en el proceso inicial.

Ventajas del D.O.O:

Proporciona una forma natural de modelar un fenómeno complejo del mundo real. Los programas tienen menos líneas de código. Otra ventaja es la herencia ya que las subclases pueden heredar o redefinir estructuras de datos y métodos de clases ya existentes y los objetos posibilitan integrar los datos y métodos que, sobre dichos datos, simplificando el mantenimiento del programa y sus actualizaciones.

UNIDAD 2

Tipos de datos:

- **Datos inmutables:** son los más sencillos de utilizar en programación (suelen ser los tipos de datos simples: String, Integer, Boolean, etc.) (cada vez que una variable, que apunta a un valor inmutable, «cambia» su valor, la misma apuntará a una nueva ubicación de memoria. Se pasan por valor y son los que más afectan a la memoria, no están optimizados para ocupar menos memoria al copiarse, se escriben más veces).
- **Datos mutables:** son los más "complejos" de utilizar en programación (suelen ser las estructuras de datos como: dict, list, clases, etc.). Los datos del tipo mutables, se pasan por referencia. (Son los que menos perjudican a la memoria se escriben 1 sola vez y se reutilizan)

Una **clase** describe un conjunto de objetos, de características similares. Contiene: atributos y funciones, que operan sobre esos datos, denominadas formalmente: funciones miembro o métodos.

Self: Es el primer parámetro de cualquier método (que no sea método de clase). Hace referencia al objeto que envía el mensaje. Es un parámetro implícito (Nunca se pasa como parámetro cuando se llama a un método).

Objeto: Es un agregado de datos y de métodos que permiten manipular dichos datos, y un programa en Python es un conjunto de sentencias que interaccionan con los objetos de la aplicación a través de mensajes. Los datos y métodos contenidos en una clase se llaman miembros de la clase y se accede a ellos siempre mediante el operador "."

Control de acceso a miembros de una clase (Encapsulamiento):

- Publico: Puede ser accedido desde cualquier código, inclusive fuera de la clase. Es un atributo o método sin ningún decorador.
- Protegido: Sólo puede ser accedido desde el código de la clase a la que pertenece y pueden accederlo las subclases también sin hacer uso de métodos, es decir en forma directa, usando el operador «.». Esta precedido por "_"
- Privado: Sólo puede ser accedido desde el código de la clase a la que pertenece. Miembro privado precedido por "__"

Toda clase tiene un método predeterminado especial, llamado **constructor** y que tiene como función inicializar los atributos del objeto. Genera espacio en memoria (heap) para almacenar un objeto de clase, y devuelve una referencia a dicha ubicación de memoria. Un constructor es un método que es invocado por el intérprete al momento de creación de un objeto y se encarga de llevar a cabo todas las tareas de inicialización de los datos miembro del objeto.

Cuando se crea una instancia de una clase, se ejecutan dos métodos: __new__ e __init__. El método __new__ es un método de clase, que necesariamente pide el espacio de memoria, y devuelve la referencia que luego es usada en todos los métodos de la clase (que no sea métodos de clase). Generalmente no reescribiremos el método __new__ porque su función no puede ser reemplazada, puede ser extendida.

Listas de Python: Son estructuras de datos secuenciales, que vienen incluidas con el core del lenguaje, y que permiten almacenar en su interior referencias a objetos.

Organización de los programas Python:

Los módulos en Python, son una forma de agrupar funciones, métodos, clases y datos, que se relacionan. Es un archivo que contiene definiciones y declaraciones de Python. El nombre del archivo es el nombre del módulo con el sufijo .py agregado.

Arreglos:

Una tabla n-dimensional (**Arreglo Numpy**) es un tipo especial de variable capaz de almacenar en su interior y de manera ordenada uno o varios datos de un determinado tipo. Este objeto arreglo es una colección de ítems todos del mismo tipo, que puede accederse usando un subíndice desde la posición 0 hasta la dimensión menos uno. Una de las formas de creación de un arreglo del tipo ndarray, es utilizando la función empty, para crear un arreglo de referencias nulas. Para crear el arreglo de un tipo de datos específico, el tipo de datos debe establecerse en el parámetro dtype. (Ej., np.empty (0, dtype=Clase))

Archivos CSV:

Son un tipo de documento en formato libre, sencillo para representar datos en forma de tabla, las columnas se separan por comas o punto y coma, o cualquier otro delimitador. Para trabajar con archivos almacenados en disco, se deben utilizar las clases especializadas que Python posee para dichas operaciones.

Apertura del flujo de datos
```
archivo = open('librosPOO.csv')
reader = csv.reader(archivo,delimiter=';')
```

Lectura de líneas
```
for fila in reader:
    ...
```

Cierre del flujo de datos
```
archivo.close()
```

Referencia Self:

En Python todo método de una clase (que no sea método de clase), recibe un primer parámetro formal que es una referencia al objeto que recibe el mensaje. Python hace obligatorio el uso de la referencia implícita en el interior de los métodos de la clase para acceder a los atributos y otros métodos.

Funciones y Métodos con parámetros por defecto:

Los constructores o métodos de una clase pueden generarse con valores por defecto. Ésta forma de trabajo permite asegurarse que los objetos de una clase, tengan una inicialización en todos sus atributos, aunque el programador no pase los valores iniciales al instanciar un objeto.

Datos miembro estáticos y Funciones miembro estáticas:

Un dato miembro de una clase se puede declarar estático, se declara sin decoradores y con el valor inicial que tendrá para todos los miembros de la clase. Un **dato miembro estático** es, por lo tanto, una variable de clase que representa información "propia de la clase" (es decir, un atributo de la clase, no un atributo de un objeto específico). Un dato miembro estático es compartido por todas las instancias (objetos) de una clase y existe, incluso cuando ningún objeto de esta clase existe.

Las **funciones miembros estáticas** sólo pueden acceder a otras funciones y datos miembros estáticos. Las funciones miembros estáticas representan los métodos de clase, por cuanto los métodos de clase manipulan solamente las variables de clase. En Python, para declarar una función como método de clase, se utiliza el decorador @classmethod. Un método de clase recibe la clase (cls) como primer argumento implícito

Sobrecarga de Operadores: Significa simplemente, capturar las operaciones básicas incluidas en el lenguaje estándar de Python, como lo son + (suma), - (resta), * (multiplicación), / (división), operadores de comparación > (mayor que), < (menor que), == (igual), etc., en las clases definidas por el programador**.** Permite que las clases intercepten el comportamiento de las operaciones normales de Python. Se implementa al proporcionar métodos especialmente nombrados en una clase.

Destructores y Recolección de Basura: Python, utiliza el mecanismo de recolección de basura (Garbage Collector), para obtener el espacio de objetos que ya no están referenciados. El principio fundamental que utiliza, se basa en el ciclo de vida de los objetos, si un objeto deja de estar referenciado, es candidato a la recolección de basura. Si el programador no provee el código de un destructor, la clase tendrá un destructor por defecto u omisión. El garbage collector estándar de Python tiene dos componentes: El recolector de basura por conteo de referencias y El recolector de basura generacional, conocido como el módulo gc.

El algoritmo de conteo de referencias es increíblemente eficiente, pero no puede detectar referencias cíclicas. Es por eso que Python tiene un algoritmo suplementario llamado GC generacional cíclico, que trata específicamente con las referencias cíclicas.

UNIDAD 3

Asociación:

Conceptualmente la asociación en un diagrama de clases implica transitividad y bidirección de clases. La cardinalidad de la asociación indicará si hace falta una (un manejador) para almacenar los objetos, podría ser una Lista, un arreglo, una lista definida por el programador.

Referencia Circular:

Una referencia circular consta de una serie de referencias donde el último objeto hace referencia al primero, dando lugar a un bucle. Una referencia cíclica ocurre cuando uno o más objetos están haciendo referencia entre sí.

Herencia:

La herencia es el mecanismo que permite compartir automáticamente métodos y datos entre clases y subclases. Este mecanismo potente, 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 -a la que se le suele llamar clase hija- será tratada por el intérprete automáticamente como si su definición incluyese la definición de la segunda –a la que se le suele llamar clase padre o clase base.

Con la función super (), se accede a atributos, y métodos de la clase base, también es posible realizar la llamada al constructor invocando: Clase.__init__(self, parámetro). Ventajas: super () es mejor porque es una referencia indirecta calculada y no es necesario especificar de qué clase hereda, de manera que la función super () determina la siguiente clase principal utilizando el Orden de resolución del método (MRO).

Todas las clases en Python, derivan de object, por lo que disponen de los atributos y métodos de dicha clase, es decir object, es la clase base o superclase de todas las clases de Python, y las definidas por el programador.

Herencia Múltiple:

La herencia múltiple ocurre cuando una clase es derivada de dos clases base o más. Las clases base se indican de la misma forma, separando cada una con una coma. Las clases derivadas heredan todos los atributos y métodos de las clases que tomen como base. Python provee un mecanismo de resolución de conflictos, denominado MRO (Method Resolution Order).

El MRO aplica también al orden de inicialización de las clases bases cuando se hace usando el método __init__ desde la función super (). 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:

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, además hace que el código sea flexible y por lo tanto reusable.

Clase de Objetos Referenciados:

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.
- type(x), donde x es una referencia a un objeto, devuelve la clase a la que pertenece dicho objeto.

La función correcta para saber si un objeto es instancia de una clase es la función isinstance(), ya que también funciona para subclases.

Errores en un programa:

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

Excepciones:

Las excepciones son errores que ocurren cuando se ejecuta un programa. Cuando se produce este error, el programa se detendrá y generará una excepción que luego se maneja para evitar que el programa se detenga por completo. (LAS EXECPCIONES SON PARA CONTROLAR SITUACIONES EXCEPCIONALES)

- assert: para probar si una afirmación es verdadera o falsa. El programador puede iniciar una afirmación de excepción con la sentencia assert. Si lo que se afirma se cumple, es verdadero, la

ejecución continúa sin problemas. Si lo que se afirma no se cumple, es falso, la ejecución se interrumpe y se la lanza la excepción AssertionError. (La utilidad principal de assert, es la depuración de un programa.)

- raise: para forzar el lanzamiento de una excepción. Python lanza excepciones en forma automática. El programador puede necesitar lanzar una excepción y para lanzar en forma manual una excepción se hace con la instrucción raise.
- try-except-else-finally: bloque para manejar y capturar excepciones, y determinar qué hacer cuando sea capturada una excepción. El bloque try-except-else-finally, se utiliza para capturar y manejar excepciones. Los bloques else y finally son opcionales. 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.

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

Listas definidas por el Programador:

Una lista enlazada es una estructura de datos dinámica. La cantidad de nodos en una lista no es fija y puede crecer y contraerse a demanda. Cualquier aplicación que tenga que tratar con un número desconocido de objetos necesitará usar una lista vinculada.

Iterador sobre la Clase Lista:

Un iterador es un objeto adherido al iterator protocol de Python. Esto significa que tiene una función next(), es decir, cuando se le llama, devuelve el siguiente elemento en la secuencia, cuando no queda nada para ser devuelto, lanza la excepción StopIteration, lo que causa que la iteración se detenga.

Para que la clase Lista, o cualquier otra clase definida por el programador, sea iterable, la clase debe proveer los métodos:

- __iter__(self), que devuelve un iterador de Python, en general se deja que devuelva el iterador de object
- __next__(self), que devuelve el siguiente elemento de la secuencia, y cuando no hay más elementos lanza la excepción StopIteration

En el caso de la clase Lista, se deberán agregar tres nuevos atributos:

- __actual: para saber cual es el elemento actual, y poder devolverlo al iterar.
- __index: lleva la cuenta de los pasos de iteración, se actualiza en uno para pasar el siguiente elemento.
- __tope: lleva la cuenta total de elementos de la lista, se actualiza sumando uno cuando se agregan elementos a la lista, y se decrementa cuando se eliminan elementos de la lista.

Testing:

Los testing son pruebas automatizadas, es decir, son programas que automáticamente ejecutan ciertas entradas (pruebas).

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 de que el código que se escribió tenga una interfaz que se pueda mantener.

Python provee la biblioteca unittest incorporada al core. Esta biblioteca proporciona una interfaz común para pruebas unitarias. Las pruebas unitarias se enfocan en probar la menor cantidad de código posible en cualquier prueba.

Cuando se escribe un conjunto de pruebas unitarias para una tarea específica, se crea una subclase de TestCase. Se escriben métodos individuales, donde estos métodos todos DEBEN comenzar con el nombre de test (se ejecutarán automáticamente).

Normalmente, las pruebas establecen algunos valores en un objeto, luego ejecutan un método, y usan los métodos de comparación incorporados para asegurarse de que se hayan calculado los resultados correctos.

Método assert: Las afirmaciones son declaraciones que se pueden hacer dentro del código, mientras se está desarrollando, las afirmaciones, se pueden usar para probar la validez del código, si la declaración no resulta ser cierta, se genera un AssertionError y el programa se detendrá. El diseño general de un caso de prueba es establecer ciertas variables o atributos, en valores conocidos, y ejecutar uno o mas métodos o procesos, y luego "probar" el valor esperado con los resultados que se devolvieron o calcularon, el chequeo se hace mediante el uso de métodos de aserción (assert) de TestCase.

Clase 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 sobrescriben 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.
- A través del módulo abc (Abstract Base Class), Python define e implementa la abstracción de clases, mediante meta clases y decoradores. El decorador para establecer que un método es abstracto, es @abc.abstractmethod.

Interfaces:

Una interfaz es un conjunto de firmas de métodos, atributos o propiedades eventos agrupados bajo un nombre común (los miembros de una interfaz no poseen modificador de acceso, por defecto son públicos). 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.

- 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 (subclase abstracta), 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 (solo de las clases que las implementen).
- Una clase que implementa una interfaz, provee implementación a todos los métodos de la interfaz.

Cuando definirlas:

- 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.
- 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, fomentando la reusabilidad del código.
- El uso de interfaces, permite restringir el acceso a funciones miembro de las clases que las implementen.

Diccionarios Python:

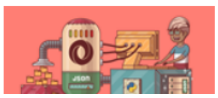
Los diccionarios en Python, son estructuras de datos utilizadas para mapear claves a valores. Los valores pueden ser de cualquier tipo, inclusive otro diccionario, una lista, un arreglo. Para crearlo, se declara el identificador, que será el nombre del diccionario, seguido de un signo igual y entre llaves los pares Clave: Valor separados por comas.

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 se ha hecho fuerte como alternativa a XML, otro formato de intercambio de datos que requiere más metainformación y, por lo tanto, consume más ancho de banda y recursos. JSON puede ser codificado y decodificado con Python, se utiliza la librería JSON de Python. Los archivos JSON, constituyen un mecanismo de persistencia del estado de los objetos.

El flujo de trabajo será el siguiente:

1. Importar la librería json
2. Proveer a las clases un método para codificar la representación tipo diccionario necesaria para JSON
3. Leer los datos usando las funciones load() o loads()
4. Procesar los datos
5. Escribir los datos usando las funciones dump() o dumps()



UNIDAD 4

Modelo de Ejecución o Computadora Virtual:

Cuando se implementa un lenguaje de programación, las estructuras de datos y algoritmos que se utilizan en tiempo de ejecución de un programa escrito en ese lenguaje, definen un modelo de ejecución o computadora virtual definida por la implementación del lenguaje.

El lenguaje de máquina de esta computadora virtual es el programa ejecutable que produce el traductor del lenguaje.

Las estructuras de control de secuencia, control de datos y gestión de almacenamiento son las que se emplean en tiempo de ejecución, independientemente de la representación de software, hardware o microprograma.

Implementación de un lenguaje de programación y Computadora Virtual:

Durante la implementación de un lenguaje, el implementador construye la computadora virtual a partir de los elementos de hardware y software que provee la computadora subyacente, está determinada por múltiples decisiones que debe tomar el implementador en función de los recursos de hardware y software disponibles en la computadora subyacente y los costos de su uso.

Jerarquías de computadoras:

La computadora virtual que un programador utiliza cuando decide hacer un programa en algún lenguaje de alto nivel está formada, por una jerarquía de computadoras virtuales. El último nivel es una computadora de hardware real. El programador rara vez tiene trato directo con esta computadora.

Esta computadora de hardware se transforma sucesivamente a través de capas de software. El segundo nivel de computadora virtual (o tercero si un microprograma forma el segundo nivel) está definido por la compleja colección de rutinas que se conoce como sistema operativo.

Cada capa es una máquina virtual que abstrae a las máquinas del nivel inferior. Las máquinas, en su nivel, «interpretan» sus instrucciones particulares, utilizando servicios de su capa inferior para implementarlas. En última instancia los circuitos terminan haciendo el trabajo.

Computadora Virtual y Objeto:

El paradigma orientado a objetos se basa en la idea de un objeto que puede describirse como una colección de localizaciones de memoria, junto con todas las operaciones que pueden cambiar los valores de dichas localizaciones de memoria.

En el fondo existe una conexión con la computadora, cada objeto se parece un poco a una pequeña computadora virtual, éste tiene un estado y tiene operaciones (métodos) que uno puede ordenarle ejecutar.

Los Lenguajes Orientados a Objetos (LOO) encaran 3 problemas de Diseño de Software: Necesidad de volver a utilizar componentes de Software, la necesidad de modificar comportamiento (cambios mínimos) y la necesidad de conservar independencia de diferentes componentes

Maneras para modificar un componente de software de modo que pueda reutilizarse:

- Extensión de los datos y/o de las operaciones
- Restricciones de los datos y/o de las operaciones
- Redefinición de una o más de las operaciones
- Abstracción o la reunión de operaciones similares de dos componentes diferentes en uno nuevo.
- Polimorfización o extensión del tipo de datos a los cuales pueden aplicarse las operaciones

Herencia y Polimorfismo:

- Polimorfismo paramétrico o genericidad, donde los parámetros de tipo pueden quedarse sin especificar(Generics en C# y Template en C++).Permite generar métodos y funciones que responden de manera similar a argumentos de diferentes tipos (ej. Listas de números, Listas de autos, etc.)
- La sobrecarga (polimorfismo ad-hoc) donde diferentes funciones o declaraciones de método comparten el mismo nombre, eliminando la ambigüedad mediante el uso de distintos tipos de los parámetros en cada declaración
- Polimorfismo de subtipo, donde todas las operaciones de un tipo pueden aplicarse a otro tipo. En la programación orientada a objetos se refiere a la capacidad de un objeto de clase o tipo A aparecer y ser utilizado como si fuera de otra clase B.

Programación Orientada a Eventos (Paradigma):

La estructura y la ejecución de los programas van determinados por los sucesos o acciones que ocurren en el sistema, definidos por el usuario o por el propio sistema. Con los lenguajes orientados a eventos se pueden realizar en poco tiempo aplicaciones sencillas y muy funcionales, utilizando interfaces gráficas en las que se insertan componentes o controles a los que se le programan eventos. Los eventos permiten al usuario realizar una serie de acciones lógicas para un determinado programa.

Secuencia, interacción, eventos:

- Secuenciales o batch, el programa inicia su ejecución, abre un archivo, lee los datos, los procesa, y entrega un resultado, no hay intervención del usuario final.
- Interactivos, el usuario provee datos necesarios para la ejecución del programa, el usuario decide qué parte del programa se ejecuta a través de un menú de opciones, por ejemplo.
- Basados en eventos, el programa se presenta a través de una interface gráfica, y se queda esperado a que el usuario o el sistema, produzca un evento, puede ser un clic del mouse o una pulsación de una tecla, que hace que el programa se ejecute una parte del mismo, este tipo de programas pasa la mayor parte del tiempo esperando eventos.

Los Eventos son las acciones sobre el programa, que permite al usuario realizar una serie de acciones lógicas, que fueron programadas por el programador en respuesta a dicho evento.

Propiedades:

Una propiedad define el tamaño de un objeto en pantalla, el título de una ventana, la etiqueta de un botón, el evento al que responde el objeto, etc. Los objetos que se ubican en una interface gráfica pueden ser: Una ventana, un botón de comando, una caja de texto etc.

Las propiedades pueden asignarse al momento de diseñar la interfaz gráfica, y pueden cambiar en tiempo de ejecución.

Interfaz gráfica de Usuario (GUI) Usando TKinter:

Con Python hay varias posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil de usar. Es multiplataforma y, además, viene incluido con Python en su versión para Windows, para Mac y para la mayoría de las distribuciones GNU/Linux.

Widgets – Tkinter:

Es un objeto predefinido, que puede ubicarse en el interior de una ventana, tiene un conjunto de propiedades que pueden modificar su aspecto, ubicación, eventos a los que responde, etc. Como un label, button, entry, text, etc.

Gestores de geometría:

Los gestores de geometría definen cómo se ubican los widgets en la ventana de la aplicación, existen tres gestores: pack, grid y place.

- Pack: Con este gestor la organización de los widgets se hace teniendo en cuenta los lados de una ventana: arriba (TOP), abajo (BOTTOM), derecha (RIGHT) e izquierda (LEFT).
- Grid: Este gestor geométrico trata una ventana como si fuera una cuadrícula, formada por filas y columnas, donde es posible situar mediante una coordenada (fila, columna) los widgets.
- Place: Se basa en el posicionamiento absoluto para colocar los widgets, aunque el trabajo de "calcular" la posición de cada widget suele ser bastante laborioso. Este método para colocar un widgets simplemente se tendrá que elegir la coordenada (x,y) de su ubicación expresada en píxeles.

Ventanas de aplicación y de diálogo:

Existen dos tipos de ventanas: las ventanas de aplicación, que suelen ser las que inician y finalizan las aplicaciones gráficas; y desde las que se accede a las ventanas de diálogo (ventanas hijas diferentes), que en conjunto constituyen la interfaz de usuario.

Ventanas modales y no modales:

En las ventanas hijas del tipo no modales es posible interactuar libremente con ellas, sin ningún límite, excepto que si se cierra la ventana principal se cerrarán todas las ventanas hijas abiertas.

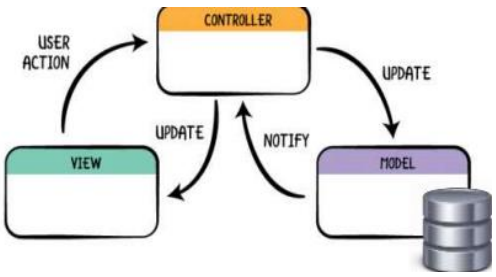
El caso contrario, es el de las ventanas modales. Cuando una ventana modal está abierta no será posible interactuar con otras ventanas de la aplicación hasta que ésta sea cerrada.

Variables de control: Son objetos especiales que se asocian a los widgets para almacenar sus valores y facilitar su disponibilidad en otras partes del programa. Pueden ser de tipo numérico, de cadena y booleano.

- El método set() asigna un valor a una variable de control. Se utiliza para modificar el valor o estado de un widget.
- El método get() obtiene el valor que tenga, en un momento dado, una variable de control.

Arquitectura MVC (Model-View-Controller):

El Modelo es donde residen los datos. Cosas como la persistencia, los objetos de modelo, los analizadores, los manejadores, residen allí. La Vista es la cara visible de la aplicación. Sus clases a menudo son reutilizables ya que no contienen ninguna lógica específica del dominio del problema. (Ej: Frame es una Vista que incluye a otros widgets, es reutilizable y extensible).



El Modelo y la Vista nunca interactúan directamente. El Controlador media entre la Vista y el Modelo. Cada vez que la Vista necesita acceder a datos de back-end, solicita al Controlador su intervención con el Modelo para obtener el requerido dato.

La gran ventaja que posee esta técnica de programación es que permite modificar cada uno de ellos sin necesidad de modificar los demás, de modo de desarrollar aplicaciones modulares y escalables que se puedan actualizar fácilmente y añadir o eliminar nuevos módulos o funcionalidades.

UNIDAD 5

Flask: Es un framework de aplicaciones web (Web Application Framework) escrito en Python.

Para dar funcionalidad a una aplicación web se debe asociar una ruta con una función. La forma de definir una ruta en una aplicación Flask es a través del decorador @app.route.

Con la asociación de la ruta a una función, se manejan las peticiones (Request) y las respuestas (Response) para la aplicación. Las peticiones se resuelven en el servidor y las respuestas es lo que envía al cliente para que se muestre al usuario.

HTML en Flask:

Es un lenguaje de marcas (HyperText Markup Language) interpretado por los navegadores web. Es básico para las aplicaciones web, se usa para definir el sentido y estructura del contenido en una página web. Para vincular una aplicación Flask con una página web se puede utilizar el método render_template() que permite procesar un archivo HTML.

En el desarrollo de una aplicación Flask se utilizan distintos entornos, editor→consola→navegador.

Una aplicación web es un conjunto de paginas web, donde cada pagina tiene asignada una dirección o ruta, a partir del decorador @app.route.

Rutas dinámicas:

A diferencia de las rutas estáticas, las rutas creadas con reglas variables aceptan parámetros, y esos parámetros son las propias variables de ruta. Entonces es posible construir una URL dinámicamente, agregando una parte variable al parámetro de la regla. La variable de la ruta se pasa a la función como argumento y la URL puede contener variables.

Formularios:

Si el objetivo es crear una página web que permita ingresar datos para que sean transferidos al servidor, allí se procesen y se retorne una respuesta al navegador, la página web deberá incluir una etiqueta <form>. La comunicación entre el cliente y el servidor se realiza a través del protocolo HTTP, que es la base de la comunicación de datos en la red mundial.

La comunicación de los datos tiene dos posibles métodos GET y POST. En la etiqueta a través de la propiedad method especifica el método (GET o POST) que empleará para la comunicación.

Request:

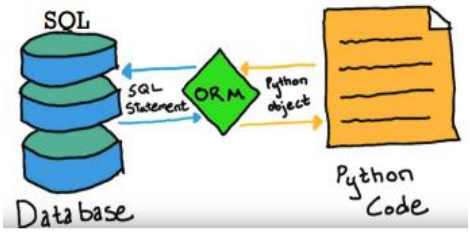
Los datos de la página web cliente se envían al servidor encapsulados en un objeto request global. El objeto request contienen información del formulario y posee el atributo method que permite saber el tipo de petición que hace el cliente (GET o POST).

El motor de template Jinja2 que incluye Flask permite formar el contenido HTML, de manera que: el Delimitador {{…}} se utiliza para incluir datos recibidos en el template, y el Delimitador {%...%} se utiliza para incluir expresiones que alteran el flujo secuencial de HTML, como iteraciones o bloques condicionales.

Persistencia:

Es necesario almacenar los objetos en un medio (por ejemplo, el disco rígido) que permita recuperarlos en cualquier momento futuro, independientemente de la ejecución de la aplicación. Para esto, se suele usar base de datos relacionales, donde la información se guarda en tablas formadas por filas y columnas.

Mapeo de objetos a base de datos relacional:



Una API ORM (Object Relational Mapping) proporciona métodos para traducir la lógica de los objetos a la lógica relacional sin tener que escribir instrucciones en el lenguaje de consulta de base de datos.

Las ventajas de usar un ORM son:

- Independencia de la aplicación con el motor de datos empleado en producción.
- Soporta los distintos motores de base de datos.
- Cambiando una línea en el archivo de configuración ya se puede emplear una base de datos distinta.

SQLAlchemy:

Proporciona una interfaz estándar que permite a los desarrolladores crear código independiente de la base de datos para comunicarse con una amplia variedad de motores de bases de datos. Permite:

- Asociar clases Python definidas por el usuario con tablas de bases de datos.
- Asociar instancias de las clases (objetos) con filas(registros) en sus tablas correspondientes.
- Sincronizar de manera transparente todos los cambios de estado entre los objetos y sus registros relacionadas, denominado unidad de trabajo.
- Sistema para expresar consultas a la base de datos en términos de las clases definidas por el usuario y las relaciones definidas entre sí.

Uso de SQLAlchemy:

Contiene funciones para operaciones ORM. También proporciona una clase Model que es la clase base para generar los modelos definidos por el usuario para el mapeo. Algunos de los métodos más comunes para realizar operaciones sobre la base de datos son:

- add: inserta un registro en la tabla correspondiente de acuerdo al objeto que se pase como parámetro.
- delete: elimina el registro de la tabla correspondiente según el objeto que se pase como parámetro.
- query.all (): recupera todos los registros de la tabla correspondiente según el objeto model al que se aplique.

Conceptos de Base de Datos:

Una base de datos es un conjunto de tablas. Una tabla es un conjunto de filas, donde cada fila es un registro. y cada columna es un campo de datos de un registro.

El diseño de las columnas, en términos de nombre, tipo de datos y distintos atributos, como valores predeterminados y si una columna puede o no ser NULL (quedar en blanco), se conoce como esquema.