Que es un algoritmo

Es un conjunto ordenado de operaciones sistemáticas que permite hallar la solución a un problema.

Este tipo de programación se llama procedimental o imperativa.

Tipo de dato estático

Datos inmutables son los más sencillos de utilizar en programación (suelen ser los tipos de datos simples: String, Integer, Boolean, etc.)

Tipo de dato dinámico

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

Tipo de dato abstracto

Un tipo de datos abstracto (TDA) hace referencia a:
1. un tipo de datos definido por el programador
2. un conjunto de operaciones abstractas sobre objetos de datos
3. y el encapsulamiento de los objetos de ese tipo, de tal manera que el usuario final del tipo no pueda manipular esos objetos excepto a través del uso de las operaciones definidas.

Objeto = estado + comportamiento + identidad

Un objeto es un conjunto de datos y de métodos que permiten manipular dichos datos, es una unidad atómica que encapsula estado y comportamiento. La encapsulación en un objeto permite una alta cohesión y un bajo acoplamiento.

- El estado: Agrupa los valores de todos los atributos de un objeto
- El comportamiento: El comportamiento describe las acciones y reacciones de ese objeto. Cada comportamiento se llama operación. Las operaciones se realizan por un estímulo externo llamado mensaje
    - Un mensaje: es una solicitud que pide al objeto que se comporte de una manera determinada. Todo mensaje está compuesto por la identidad del receptor, el comportamiento a ejecutar y opcionalmente información adicional
- La identidad: Además de su estado, un objeto posee una identidad que caracteriza su propia existencia. La identidad permite distinguir los objetos de forma no ambigua, independientemente de su estado.

Descripción de objetos: Los objetos informáticos definen una representación abstracta de las entidades de un mundo real o virtual, con el objetivo de controlarlos o simularlos.

- Atributos de un objeto: Los atributos describen la abstracción de características individuales que posee un objeto.
- Comportamientos: Los comportamientos de un objeto representan las acciones que pueden ser realizadas por un objeto.

Atributos y visibilidad

Un atributo es una información que cualifica al objeto que la contiene. Esos atributos corresponden a una selección de valores, entre todos los valores posibles.

Visibilidad de los atributos: indica si el atributo y/o comportamiento es visible o puede ser accedido desde otras clases. Niveles: privado, protegido y publico

- Privado: totalmente inaccesible desde otras clases
- Protegidos: son visibles solo para las subclases
- Públicos: son visibles para todas las clases

- 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.
- Variables de instancia: Es un elemento de información que define un atributo de un objeto en particular. Se usan para guardar los atributos de un objeto particular.

Métodos

Comportamiento u operaciones que caracterizan un conjunto de objetos que residen en una clase.
Determina como actúa el objeto cuando recibe un mensaje y manipulan los valores de los atributos del objeto. Son el código que se ejecuta para responder a un mensaje y el mensaje es la llamadas o invocación a un método.

POO (fundamento, abstracción y diseño de un programa)

Durante el proceso de desarrollo de requerimientos de un sistema de software el analista es responsable de identificar los objetos del problema y caracterizarlos a través de sus atributos. En la etapa de diseño se completa la representación modelando también el comportamiento de los objetos.

- Un objeto del problema: Es una entidad, física o conceptual, caracterizada a través de atributos y comportamiento. El comportamiento queda determinado por un conjunto de servicios que el objeto puede brindar y un conjunto de responsabilidades que debe asumir.
- Cuando el sistema de software está en ejecución, se crean objetos de software: Un objeto de software es un modelo, una representación de un objeto del problema. Un objeto de software tiene una identidad y un estado interno y recibe mensajes a los que responde ejecutando un servicio. El estado interno mantiene los valores de los atributos.

UML definición (lenguaje de modelado unificado)

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.
Se utiliza tanto para el modelado, diseño y construcción de sistemas de software así como también para cosas más generales y procesos científicos.

Diagramas de objetos, clases, secuencia y comunicación

Los diagramas de objetos: muestran cómo las instancias de elementos estructurales se relacionan y usan en tiempo de ejecución.

Los diagramas de clases (o estructurales): definen los bloques de construcción básica de un modelo: los materiales generales, clases y tipos que se usan para construir un modelo completo.

Representación de una clase usando UML: Una clase se representa con un rectángulo que posee tres divisiones:

- Superior: contiene el nombre de la clase
- Intermedia: contiene los atributos que caracterizan a la clase
- Inferior: contiene los métodos, los cuales indican la forma como interactúa el objeto con su entorno
- Un objeto se representa bajo la forma de un rectángulo, el nombre del objeto se subraya. La primera parte del rectángulo indica el nombre y la segunda los valores asociados a los atributos.

Los diagramas de secuencia: están estrechamente relacionados a los diagramas de comunicaciones y muestran la secuencia de mensajes pasadas entre los objetos usando una línea de tiempo vertical

El diagrama de comunicación: muestra la red y la secuencia de mensajes de comunicaciones entre objetos en tiempo de ejecución durante una instancia de colaboración.

Herencia

Es el mecanismo que permite compartir automáticamente métodos y datos entre clases y subclases.

- Generalización: Consiste en factorizar los elementos comunes (atributos, métodos y restricciones) de un conjunto de clases en una clase más general llamada superclase. La superclase es una abstracción de sus subclases. Se parte desde las “hojas” porque estas pertenecen al mundo real mientras que los niveles superiores son abstracciones construidas para ordenar y comprender.
- Especialización: Permite capturar las particularidades de un conjunto de objetos no discriminados por las clases ya identificadas. Las nuevas características se representan por una nueva clase, subclase de una de las clases existentes.

Tipos de Herencia: Existen dos tipos de herencia

1. Herencia simple: una subclase puede heredar datos y métodos de una única clase y también puede añadir nuevo comportamiento

2. Herencia múltiple: una subclase puede adquirir los datos y métodos de más de una clase padre

Relaciones entre clases: Asociación

Es una conexión bidireccional entre dos clases. Es una abstracción de los vínculos que existen entre objetos de las clases asociadas. Es el reflejo de una conexión que existe en el ámbito de la aplicación. Puede tener o no un nombre para una mayor legibilidad del diagrama. Es representada por una línea continua entre las clases asociadas.

- Clase Asociación: Se utiliza cuando es necesario mantener la información que es especifica de la propia asociación y no pertenece a ninguna de las clases, la 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. No importa la multiplicidad de ambos extremos. Ej.: Registro civil, persona, acta de nacimiento
- Clase que Modela: Básicamente permite que a la clase intermedia que se encuentra en la asociación, llegue más de un objeto. Ej.: Jugador de futbol, equipo de futbol, contrato

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.

Relaciones entre clases: Agregacion

Consiste en crear una nueva clase que va a contener la clase que queremos reutilizar. Es una asociación no simétrica (todo/parte) en la que una de las clases cumple un papel predominante respecto de la otra. Declara una dirección a la relación todo/parte. Se representa colocando un rombo vacío del lado de la clase agregado.

Criterios para detectarla:

- Los objetos de una clase están subordinados a los objetos de otra clase.
- Una acción sobre una clase implica una acción sobre otra clase
- Los valores de los atributos de una clase se propagan en los valores de los atributos de otra clase
- La relación de agregación se presenta en aplicaciones en las cuales un objeto contiene como partes a objetos de otras clases, pero la destrucción del objeto continente no implica la destrucción de sus partes.
- El objeto continente o contenedor incluye referencias a objetos de otras clases y que los objetos contenidos pueden existir independientemente del objeto contenedor

Relaciones entre clases: Composición

Con la composición el objeto parte puede pertenecer a un todo único En la relación de composición, si muere el objeto que dispara la composición, los objetos que lo componen mueren con él. En la agregación, pasa lo contrario. La relación de composición es cuando hay agregaciones donde los objetos agregados no tienen sentido fuera del objeto resultante. Los atributos de una clase hacen referencia a objetos de otra clase y estos están físicamente contenidos por el agregado. Entonces, esto quiere decir, el objeto de la nueva clase (contenedor)

incluye o contiene objetos de una clase ya definida pero cuya existencia solo tiene sentido en este contexto. Se representa con un rombo lleno que apunta al agregado.

Relaciones entre clases: Multiplicidad

- 1 uno y solo uno
- 0..1 cero o uno
- n n y solo n
- m..n de m a n
- * de cero a varios
- n..* de n a varios

Reutilización y extensión de objetos u clases

Consiste en el uso de clases u objetos desarrollados y probados en un determinado contexto, para incorporar esa funcionalidad en una aplicación diferente a la de origen.

Encapsulación

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 de la clase a la que pertenece el objeto. La comunicación entre los distintos objetos se realiza solamente a través de mensajes explícitos y sus ventajas son:

- Se protegen los datos de accesos indebidos
- El acoplamiento entre las clases disminuye
- Favorece la modularidad y el mantenimiento.
- Permite una alta cohesión y un bajo acoplamiento

Polimorfismo

El polimorfismo es la capacidad que tienen objetos de clases diferentes, a responder de forma distinta a un mismo mensaje

- Python es un lenguaje con tipado dinámico, la vinculación de un objeto con un método también es dinámica.
- Todos los métodos se vinculan a las instancias en tiempo de ejecución.
- Para llevar a cabo 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.

Clase

Es una descripción de un conjunto de objetos, es decir una clase describe un conjunto de objetos, de características similares. Contiene los atributos y funciones (que operan sobre esos datos, denominados métodos.) comunes entre los objetos del conjunto

La posibilidad de definir clases es una de las ventajas de la orientación a objetos, definir clases significa colocar código reutilizable en un depósito común en lugar de redefinirlo cada vez que se necesite. Los objetos que responden a la especificación de una clase son llamados instancias de una clase.

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.

Instancia

Instanciación: Es la operación que se realiza para obtener un objeto de una clase. La instanciación relaciona una clase y un objeto.

Mensajes

Solicitud que pide al objeto que se comporte de manera determinada. Cada objeto recibe, interpreta y responde a mensajes enviados por otros objetos. El conjunto de mensajes al que un objeto puede responder se llama protocolo del objeto.

Persistencia

Un objeto persistente es aquél que conserva su estado en un medio de almacenamiento permanente, pudiendo ser reconstruido por el mismo u otro proceso. Al objeto no persistente lo llamamos efímero.

Ventajas del diseño OO

- Beneficia al desarrollador de software proporcionando una forma natural de modelar un fenómeno del mundo real.
- Tienen menos líneas de código. Es decir, hay mayor reusabilidad, gracias a mecanismos como la herencia, ya que las subclases pueden heredar o redefinir estructuras de datos y métodos de clases existentes.
- Los objetos posibilitan integrar los datos y los métodos que actúan sobre dichos datos, lo que simplifica el mantenimiento del programa y sus posibles actualizaciones.

Desventaja: El programador debe conocer una extensa biblioteca de clases, pero este tiempo esta compensado
con el tiempo ahorrado en la reutilización de código, en lugar de tener que volverlo a crear.

Excepciones y errores

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.

- Error 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

Se utilizan para controlar situaciones excepcionales

Instrucción	Resultado/Acción
Assert	Para probar si una afirmación es verdadera o falsa, se utiliza para la depuración de un programa, si la afirmación es verdadera la ejecución del programa continua si es falsa se lanza la excepción AssertionError
Raise	Fuerza el lanzamiento de una excepción
Try-except-else-finally	El bloque Try-except-else-finally se utiliza para capturar y manejar excepciones 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 e iteradores

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

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.

Módulos

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.

Testing

Los testing son pruebas automatizadas, es decir, son programas que automáticamente ejecutan ciertas entradas (pruebas). 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 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 y se escriben métodos individuales (que deben empezar con “test”) para realizar la prueba real.

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) (Python no permite definir interfaces, es necesario incorporar una librería externa para emular el comportamiento de las interfaces)

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 (subclase abstracta), mientras que las clases que implementen una interfaz deben proveer implementación de todos los métodos de la interfaz.

Diccionarios

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.

Self

Es el primer parámetro de cualquier método (que no sea método de clase o método estático). Hace referencia al objeto que envía el mensaje. Es un parámetro implícito

Constructor

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.

Arreglo Numpy (Una tabla n-dimensional)

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 n menos uno.

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.

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.

Sobrecarga de operadores

Permite que las clases intercepten el comportamiento de las operaciones normales de Python. Se implementa al proporcionar métodos especialmente nombrados en una clase.

Colector de basura

El lenguaje de programación 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.

Unidad 4

Tkinter

Es una librería la cual nos permite programar una interfaz grafica de usuario (GUI) en python, es multiplataforma y viene incluido con python en su versión para Windows

Widgets

Los widgets son objetos predefinidos que pueden ubicarse en el interior de una ventana de tkinter, tiene un conjunto de propiedades que pueden modificar su aspecto, ubicación, evento a los que responde, etc.

Widget Class	Descripción
Label	Un widget que se usa para mostrar texto en la pantalla
Button	Un botón, contiene un texto, que describe la función que realiza cuando es presionado.
Entry	Un widget de entrada, permite ingresar una línea de texto.
Text	El widget text permite el ingreso de múltiples líneas de texto.
ListBox	Una caja con múltiples opciones, se elige una de ellas.
Frame	Una región rectangular para agrupar varios widgets relacionados o proveer relleno entre widgets.

LabelFrame	Una región rectangular que permite agrupar varios widgets relacionados, que posee un título en la forma de una etiqueta.

Geometrías

Los gestores de geometría definen como se ubican los widgets en la ventana de la aplicación.

A la hora de diseñar las ventanas, se pueden utilizar unos widgets especiales (marcos, paneles, etc.) que actúan como contenedores de otros widgets.

Estos widgets contendores, se utilizan para agrupar varios controles con el objeto de facilitar la operación a los usuarios. En las ventanas que se utilicen podrá emplearse un gestor con la ventana y otro diferente para organizar los controles dentro de estos widgets

- Pack: Con este gestor la organización de los widgets se hace teniendo en cuenta los lados de una ventana, con este gestor de geometría, es posible hacer que los controles se ajusten a los cambios del tamaño de la ventana

  'side': los valores posibles para la propiedad son: TOP (arriba), BOTTOM (abajo), LEFT (izquierda) y RIGHT (derecha). Si se omite, el valor será TOP.

  ´fill' : la propiedad' fill' se utiliza para indicar al gestor cómo expandir/reducir el widget si la ventana cambia de tamaño.

  Tiene tres posibles valores: BOTH (Horizontal y Verticalmente), X (Horizontalmente) e Y (Verticalmente). Funcionará si el valor de la propiedad 'expand' es True.

  'padx', 'pady': las propiedades 'padx' y 'pady' se utilizan para añadir espacio extra externo horizontal y/o vertical a los widgets para separarlos entre sí y de los bordes de la ventana. Hay otras equivalentes que añaden espacio extra interno: 'ipàdx' y 'ipady':

- Grid: Este gestor geométrico trata una ventana como si fuera una cuadrícula, formada por filas y columnas como un tablero de ajedrez, donde es posible situar mediante una coordenada (fila, columna) los widgets; teniendo en cuenta que, si se requiere, un widget puede ocupar varias columnas y/o varias filas.

- Place: Este gestor es el más fácil de utilizar porque se basa en el posicionamiento absoluto para colocar los widgets, aunque el trabajo de "calcular" la posición de cada widget suele ser bastante laborioso.

  Se sabe que una ventana tiene una anchura y una altura determinadas (normalmente, medida en píxeles). Este método para colocar un widget simplemente se tendrá que elegir la coordenada (x, y) de su ubicación expresada en píxeles. La posición (x=0, y=0) se encuentra en la esquina superior izquierda de la ventana. Con este gestor el tamaño y la posición de un widget no cambiará al modificar las dimensiones de una ventana.

Ventanas de aplicación y de dialogo

Los gestores de geometría vistos, se utilizan para diseñar las ventanas de una aplicación. En Tkinter 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, que en conjunto constituyen la interfaz de usuario.

Ventanas modales y no modales

Las ventanas del tipo no modales son aquellas donde es posible interactuar libremente con ellas, sin ningún límite

Un ejemplo evidente que usa ventanas no modales está en las aplicaciones ofimáticas más conocidas, que permiten trabajar con varios documentos al mismo tiempo, cada uno de ellos abierto en su propia ventana, permitiendo al usuario cambiar sin restricciones de una ventana a otra.

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.

Un ejemplo típico es el de algunas ventanas de diálogo que se utilizan para establecer las preferencias de las aplicaciones, que obligan a ser cerradas antes de permitirse la apertura de otras.

Variables de control

Las 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.

Cuando una variable de control cambia de valor el widget que la utiliza lo refleja automáticamente, y viceversa.

Las variables de control también se emplean para conectar varios widgets del mismo tipo

IntVar, DoubleVar, StringVar, BooleanVar

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. Se utiliza cuando es necesario leer el valor de un control

El método trace() se emplea para "detectar" cuando una variable es leída, cambia de valor o es borrada: widget.trace(tipo, función)

El primer argumento establece el tipo de suceso a comprobar: 'r' lectura de variable, 'w' escritura de variable y 'u' borrado de variable. El segundo argumento indica la función que será llamada cuando se produzca el suceso

Estrategias para validar y calcular datos

Cuando se construye una ventana con varios widgets se pueden seguir distintas estrategias para validar los datos que se introducen durante la ejecución de un programa:

- Una opción posible podría validar la información y realizar los cálculos después de que sea introducida, por ejemplo, después de presionar un botón.
- Otra posibilidad podría ser haciendo uso del método trace() y de la opción 'command', para validar y calcular la información justo en el momento que un widget y su variable asociada cambien de valor.

Menú de opciones

Los menús pueden construirse agrupando en una barra de menú varios submenús desplegables, mediante menús basados en botones, o bien, utilizando los típicos menús contextuales que cambian sus opciones disponibles dependiendo del lugar donde se activan en la ventana de la aplicación.

Entre los tipos de opciones que se pueden incluir en un menú se encuentran aquellas que su estado representa un valor lógico de activada o desactivada (add_checkbutton); las que permiten elegir una opción de entre varias existentes (add_radiobutton) y las que ejecutan directamente un método o una función (add_command).

Las opciones de un menú pueden incluir iconos y asociarse a atajos o combinaciones de teclas que surten el mismo efecto que si éstas son seleccionadas con un clic de ratón.

También, en un momento dado, pueden deshabilitarse para impedir que puedan ser seleccionadas.

Arquitectura MVC (modelo, vista y controlador)

Algunos de los aspectos centrales de la arquitectura MVC son los siguientes:

- 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. Por ejemplo, un 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 los datos requeridos

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 en forma de paquete, ya que cada "paquete" utiliza el mismo sistema con sus vistas, modelos y controladores.

Flask

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

Se basa en el kit de librerías como Werkzeug y Jinja2. Werkzeug es un toolkit para aplicaciones WSGI (Web Server Gateway Interface), que es una interface entre aplicaciones Python y servidores web. Jinja2 es un motor para el renderizado de plantillas (templates) web.

```
from flask import Flask
app = Flask(__name__)

app.run()
```

Agregando funcionalidades

La aplicación web puede conformarse por varias funcionalidades, además de la inicial. Para agregar funcionalidades, basta con crear otras rutas en la aplicación. Por ejemplo, si queremos controlar la ruta '/saludo' el método route se invocará como:

```
@app.route('/saludo')
```

Para vincular una aplicación Flask con una página web se puede utilizar el método render_template() que permite procesar un archivo HTML.

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. En este protocolo se definen diferentes métodos de recuperación de datos de la URL especificada.

La comunicación de los datos tiene dos posibles métodos GET y POST. Una página HTML que contiene un formulario, en la etiqueta <form> a través de la propiedad method especifica el método (GET o POST) que empleará para la comunicación.

El motor de template Jinja2 que incluye Flask, permite que la página web receptora de los datos los use a través del delimitador {{...}} para escapar de HTML. Este delimitador se usa para que las expresiones se impriman en la salida del template.

Otro delimitador de interés es {% ... %} que se usa para incluir en el template expresiones que alteran el flujo secuencial del HTML, como son los condicionales e iteraciones

Flask SQLAlchemy

Persistencia

Es muy común que una aplicación desarrollada bajo el paradigma de la orientación a objetos requiera la persistencia de los objetos que maneja. Es decir, 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.

La aplicación orientada a objetos debe realizar una serie de operaciones sobre el conjunto de tablas que contiene la base de datos, para ello deben comunicarse entre sí, convirtiéndose esto en un inconveniente que se puede resolver con librerías de mapeo objeto – relacional.

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 (SQL - Structured Query Language).

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.

Distintos paquetes han sido desarrollados para facilitar el proceso de mapeo de objetos a base de datos relacional proveyendo bibliotecas de clases que son capaces de realizar mapeos automáticamente.

SQLAlchemy

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.

Este ORM provee mecanismos para asociar clases Python definidas por el usuario con tablas de bases de datos, e instancias de esas clases (objetos) con filas (registros) en sus tablas correspondientes. Incluye un sistema que sincroniza de manera transparente todos los cambios de estado entre los objetos y sus filas relacionadas, denominado unidad de trabajo. Además de un sistema para expresar consultas de la base de datos en términos de las clases definidas por el usuario y sus relaciones definidas entre sí.

Se admiten los sistemas de administración de bases de datos más comunes disponibles.

PostgreSQL, MySQL, Oracle, Microsoft SQL Server y SQLite son ejemplos de motores que se pueden usar con SQLAlchemy.

Uso de SQLAlchemy

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. SQLAlchemy gestiona las operaciones de persistencia a través de distintos objetos. Esto es, un objeto SQLAlchemy, un objeto sesión y sus respectivos métodos. 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. Flask-SQLAlchemy provee un objeto query para cada clase del modelo. Además, a través de un objeto query se puede aplicar un filtro al conjunto de registros recuperados.