Universidad Nacional de Asunción
Facultad Politécnica

PARADIGMAS DE LA PROGRAMACIÓN

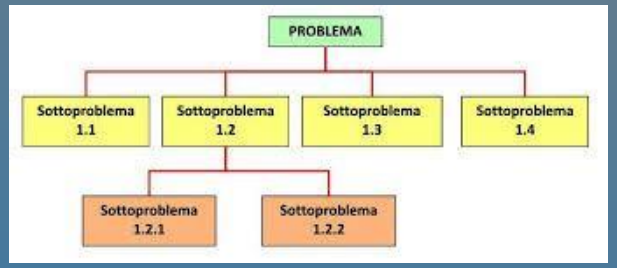
Modularización

(Model, View, Controller)

Python

MODULARIZACIÓN

La modularización permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.



MVC (MODEL – VIEW – CONTROLLER)

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC).

COMPONENTES:

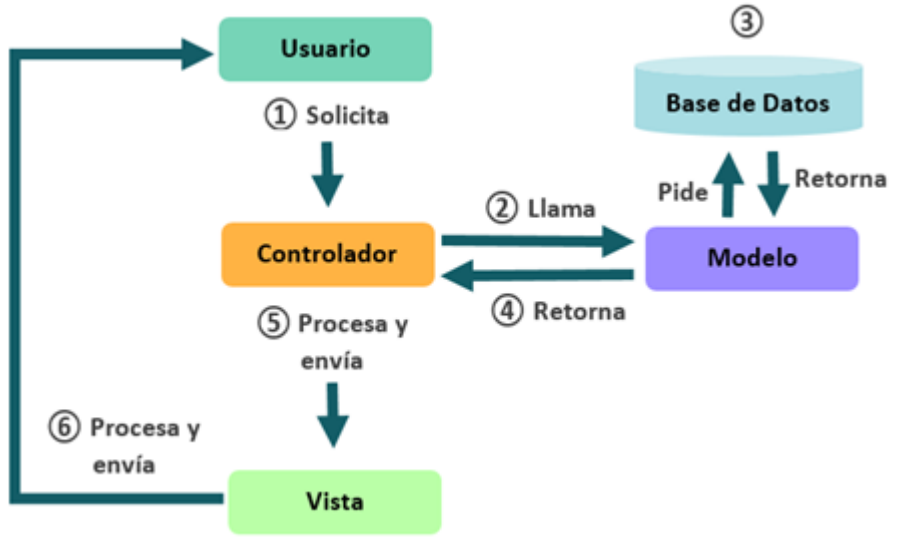
- **MODELO**
  Se encarga de los datos, generalmente (pero no obligatoriamente) consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. todo eso va aquí, en el modelo.

- **CONTROLADOR**
  Se encarga de controlar, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

- **VISTAS**
  Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.

CICLO DE VIDA DEL MVC

- El usuario realiza una petición.
- El controlador captura la petición del usuario.
- El controlador llama al modelo.
- El modelo interactúa con la base de datos, y retorna la información al controlador.
- El controlador recibe la información y la envía a la vista.
- La vista procesa la información recibida y la entrega de una manera visualmente entendible al usuario.



Ventajas de MVC

Las principales ventajas del uso del patrón MVC :

- La separación del *Modelo* y la *Vista*, logra separar los datos, de su representación visual.
- Facilita el manejo de errores.
- Permite que el sistema sea escalable si es requerido.
- Es posible agregar múltiples representaciones de los datos.

Desventajas de MVC

Las principales desventajas del uso del patrón MVC :

- La cantidad de archivos que se deben mantener incrementa considerablemente.
- La curva de aprendizaje es más alta que utilizando otros modelos.
- Su separación en capas, aumenta la complejidad del sistema.

EJEMPLO- PRÁCTICO

MODELO-VISTA-CONTROLADOR

Ejemplo: Pensemos en el inventario de una pequeña tienda de comestibles. Una lista de productos típica se vería así:

Producto	Precio	Cantidad
Bread	0.5	20
Milk	1.0	10
Wine	10.0	5

El programa debe permitir visualizar los productos existentes y crear nuevos productos.

1. Modelo

El modelo gestiona los datos y define reglas y comportamientos. Representa la lógica empresarial de la aplicación. Los datos se pueden almacenar en el propio Modelo o en una base de datos (solo el Modelo tiene acceso a la base de datos).

ModelBasic.py

```python
import basic_backend as basic_backend
import mvc_exceptions as mvc_exc


class ModelBasic(object):

    def __init__(self, application_items):
        self._item_type = 'product'
        self.create_items(application_items)

    @property
    def item_type(self):
        return self._item_type

    @item_type.setter
    def item_type(self, new_item_type):
        self._item_type = new_item_type

    def create_item(self, name, price, quantity):
        basic_backend.create_item(name, price, quantity)

    def create_items(self, items):
        basic_backend.create_items(items)

    def read_item(self, name):
        return basic_backend.read_item(name)

    def read_items(self):
        return basic_backend.read_items()
```

basic_backend.py

```python
items = list()  # global variable where we keep the data

def create_items(app_items):
    global items
    items = app_items

def create_item(name, price, quantity):
    global items
    items.append({'name': name, 'price': price, 'quantity': quantity})

def read_item(name):
    global items
    myitems = list(filter(lambda x: x['name'] == name, items))
    return myitems[0]

def read_items():
    global items
    return [item for item in items]
```

2. Vista

Ahora que la lógica empresarial está lista, centrémonos en la capa de presentación. Los datos se presentan al usuario en un shell de Python (definitivamente esto no utilizaríamos una aplicación real). Sin embargo, lo importante a notar es que no hay lógica en la clase View, y todos sus métodos son funciones normales.

Model_View.py

```
class View(object):

    @staticmethod
    def show_bullet_point_list(item_type, items):
        print('--- {} LIST ---'.format(item_type.upper()))
        for item in items:
            print('* {}'.format(item))

    @staticmethod
    def show_number_point_list(item_type, items):
        print('--- {} LIST ---'.format(item_type.upper()))
        for i, item in enumerate(items):
            print('{}. {}'.format(i+1, item))

    @staticmethod
    def show_item(item_type, item, item_info):
        print('////////////////////////////////////////////////////////////////')
        print('Buenas noticias. Encontramos el producto {}!'.format(item.upper()))
        print('{} INFO: {}'.format(item_type.upper(), item_info))
        print('////////////////////////////////////////////////////////////////')

    @staticmethod
    def display_missing_item_error(item, err):
        print('****************************************************************')
        print('Lo sentimos, no tenemos el producto {}!'.format(item.upper()))
        print('{}'.format(err.args[0]))
        print('****************************************************************')
```

En este módulo, no se mencionan los otros dos componentes del patrón MVC. Esto significa que si desea diseñar una interfaz de usuario elegante para su aplicación, solo tiene que reemplazar la clase View.

3. Controlador

Finalmente, ahora que las reglas y la lógica (el Modelo) y la representación de la información (la Vista) están hechas, podemos enfocarnos en el Controlador. El controlador acepta las entradas del usuario y delega la representación de datos a la vista y el manejo de datos al modelo.

Controller.py

```
class Controller(object):

    def __init__(self, model, view):
        self.model = model
        self.view = view

    def insert_item(self, name, price, quantity):
        assert price > 0, 'price must be greater than 0'
        assert quantity >= 0, 'quantity must be greater than or equal to 0'
        item_type = self.model.item_type
        try:
            self.model.create_item(name, price, quantity)
            self.view.display_item_stored(name, item_type)
        except:
            pass

    def show_items(self, bullet_points=False):
        items = self.model.read_items()
        item_type = self.model.item_type
        if bullet_points:
            self.view.show_bullet_point_list(item_type, items)
        else:
            self.view.show_number_point_list(item_type, items)
```

Cuando se crea una instancia de un controlador, debe especificar un modelo y una vista. En el caso de que desee utilizar un modelo diferente y / o una vista diferente, solo tiene que conectarlos cuando cree una instancia del controlador.

Test ¡Veamos cómo funciona!

1. Crea algunos elementos y crea una instancia de un controlador.

```
my_items = [
        {'name': 'bread', 'price': 0.5, 'quantity': 20},
        {'name': 'milk', 'price': 1.0, 'quantity': 10},
        {'name': 'wine', 'price': 10.0, 'quantity': 5}]
```

2. Importamos los 3 componentes: El modelo, la vista y el controlador.

```
import model_controller as model_controller
import ModelBasic as ModelBasic
import model_view as model_view
c = model_controller.Controller(ModelBasic.ModelBasic(my_items), model_view.View())
```

3. Ejecutamos la visualización y la inserción de productos.

```
c.show_items( bullet_points=True)
```

```
PRODUCT LIST ---
name': 'bread', 'price': 0.5, 'quantity': 20}
name': 'milk', 'price': 1.0, 'quantity': 10}
name': 'wine', 'price': 10.0, 'quantity': 5}
```

```
c.insert_item('chocolate', price=2.0, quantity=0)
```

```
c.show_items()
```

```
PRODUCT LIST ---
{'name': 'bread', 'price': 0.5, 'quantity': 20}
{'name': 'milk', 'price': 1.0, 'quantity': 10}
{'name': 'wine', 'price': 10.0, 'quantity': 5}
{'name': 'chocolate', 'price': 2.0, 'quantity': 0}
```

GRACIAS!