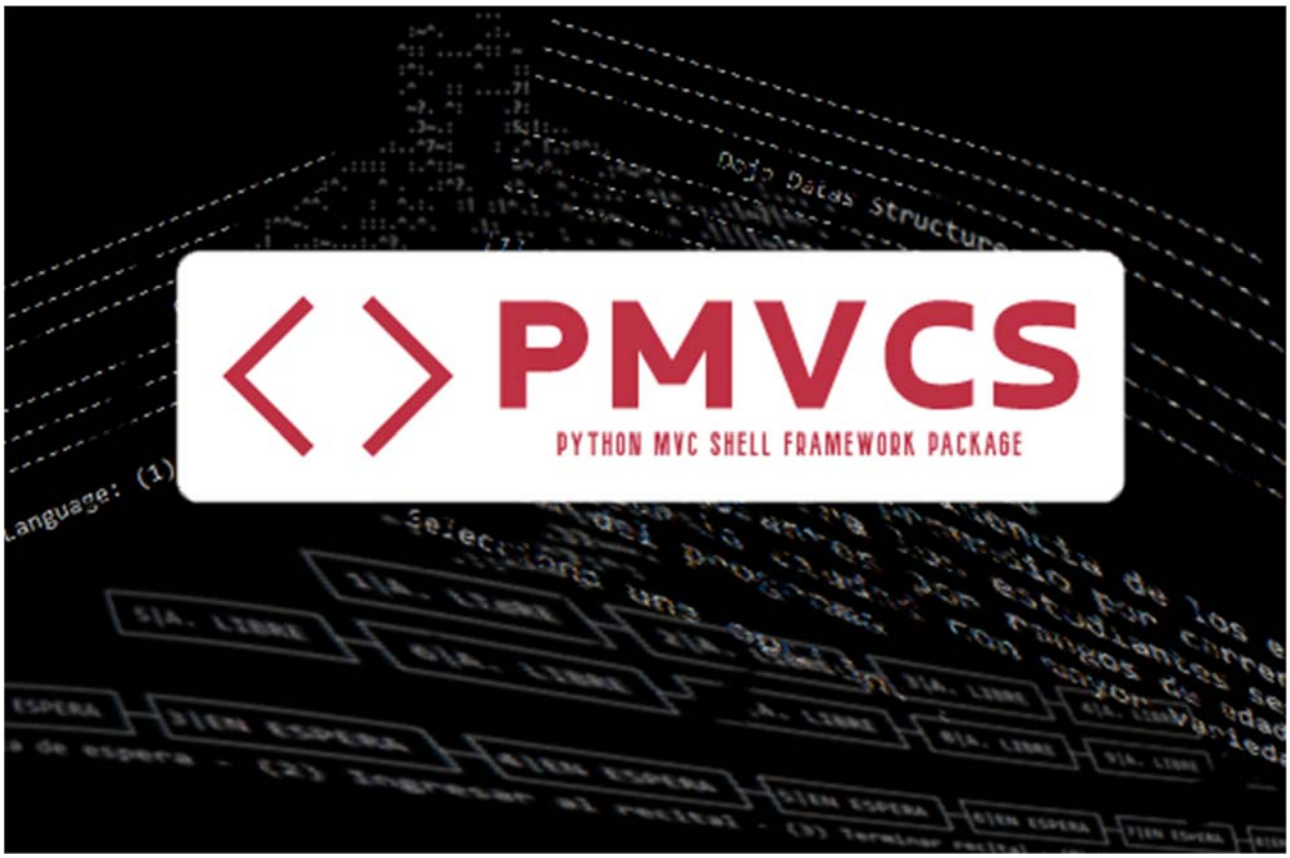
PROYECTO FINAL BOOTCAMP PYTHON AVANZADO CÓDIGO FACILITO



DATOS TÉCNICOS

- NOMBRE: Python MVC Shell Framework Package
- ALIAS: pmvcs
- VERSIÓN ACTUAL: 1.0.3 (pronto a salir una 1.0.4)
- LENGUAGE: Python 3.10+
- DESCRIPCIÓN: Python MVC Shell Framework Package (PMVCS) es un pequeño framework para proyectos en shell realizados en Python.
- REPOSITORIO: https://github.com/gsmx64/pmvcs/
- PYPI.ORG: https://pypi.org/project/pmvcs/
- README (INGLÉS): https://github.com/gsmx64/pmvcs/blob/main/README.md
- README (ESPAÑOL): https://github.com/gsmx64/pmvcs/blob/main/README.md
- CHANGELOG: https://github.com/gsmx64/pmvcs/blob/main/CHANGELOG.md
- LICENCIA: GNU General Public License v3 (GPLv3)
- EJEMPLOS DE IMPLEMENTACIÓN: https://github.com/gsmx64/pmvcs/tree/main/examples

Imágenes de PMVCS como framework

ÍNDICE

MOTIVACIÓN

La idea fue la necesidad de contar con una base de código para los retos de las tareas que nos iban dejando en las clases, no me quede con un simple código de la primera que fue el juego Codebraker, quería armar una versión 2 con código mejorado para empezar a programar en Python algo propio por primera vez.

Luego de una larga búsqueda de ejemplos que se adapten a lo que yo quería armar en formato MVC y no encontrar nada que ya traiga un archivo de configuración y de múltiples idiomas, fui armando esa base, sin aún pensar en que llegase a transformarse en un framework.

En el tema idioma ya había buscado antes alguna alternativa cuando hice cursos de Flask de CF, y no me gustó la implementación de Babel, buscaba algo más cercano a lo que hace uso Joomla para los archivos de idioma.

Entonces, teniendo en base el modelo de MVC que utiliza Joomla lo traté de llevar a Python, controladores, modelos, vistas, con helpers (que luego terminaron siendo como plugins).

Armé un parser para trabajar junto a ConfigParser que viene integrado en Python oficial, mejorándolo en varios aspectos para que en idiomas y los about (que son la info que tira al final de un proyecto) no tengan que requerir el ingreso de la "sección" del archivo ini, otro caso sería en la creación de sprintf (string de formateo) donde internamente usa fstring con format para pasarle variables a las constantes, y otra muy interesante la de traducción donde un string dado desde cualquier fuente, por ejemplo un csv tiene la columna de nombre name, busca esa constante en el archivo de idioma y la traduce.

Otra implementación importante fue la de crear opciones de menú, que necesitaba para el segundo reto que fue el de "Dojo Datas Structures", y la idea de hacerlo dinámico desde la lectura de un archivo ini para que pueda especificarse el nombre de la opción según el idioma, y teniendo como parámetro el "call" que es el nombre del módulo a levantar para cada parte de MVC y cargarlos por medio de importlib.

Y así fue madurando hasta poder tener una versión probada en funcionamiento y subirla a Pypi.org junto a su documentación; posteriormente agregué el changelog y actualicé retos realizados en clases para tener como ejemplo de implementación.

¡Espero que sea del agrado de los usuarios!

HISTORIAL

- 12/06/2023 - Primera aparición de una base (reto 1 y 5): https://github.com/gsmx64/python-adv-bootcamp-cf/commit/36d824213a1854fd54c5e345c3e6615c2fd9eeca
- 16/06/2023 - Segunda aparición: https://github.com/gsmx64/python-adv-bootcamp-cf/commit/4c675104a959a05e5deb006354ba2638288e9661
- 09/07/2023 - Tercera aparición más maduro, se llamaba "pmvcp": https://github.com/camigomezdev/dojo_datastructures/commit/f3c7a685ef085620e7c7953ddd638ca10b76db8b y https://github.com/gsmx64/python-adv-bootcamp-cf/commit/1cacddf1f472f624d448734d5c56c950d8e54a91
- 24/07/2023 – Release 1.0.0: https://github.com/gsmx64/pmvcs/commit/c06fb258faf4dbf4e9efec91c1250e245db085e2
- 26/07/2023 – Release 1.0.1: https://github.com/gsmx64/pmvcs/commit/daab2bfa9a9cc646a223f499b6f0c9b8d4152478
- 29/07/2023 – Release 1.0.2: https://github.com/gsmx64/pmvcs/commit/ed4aa7451dceccd13dd05df8eb08b8051846f06b

- 10/08/2023 – Release 1.0.3:
  https://github.com/gsmx64/pmvcs/commit/c230090f66c9d46f9a0ab1ab1f12c41c0d7402a6

RETOS Y ELABORACIÓN DEL PROYECTO

Algunos de los retos que me dio el proyecto:

- Buscar algún paquete para hacer los archivos de idioma y para las configuraciones, ahí conocí ConfigParser, tuve que leer mucho de la documentación, algunas cosas fueron prueba y error hasta dejarlo a punto, en especial con el modelo base "parser.py" que hace herencia a los demás módulos "configuration.py", "about.py" y "language.py".
- Al subirlo a pypi.org y ponerlo como módulo no leía los archivos .ini porque no los encontraba usando "pathlib", ahí me di cuenta que la altura del path en que se ubican los paquetes en un entorno virtual son distintos a tenerlo en la misma carpeta del proyecto e incluso difieren en cada sistema operativo, por lo que haciendo ingeniería inversa en paquetes de terceros encontré "importlib", busqué documentación y lo implementé solucionando ese bug.
- Subir un paquete a pypi.org, no fue sencillo, ya que la documentación sobre cómo armar el paquete para subirlo está muy desactualizado en internet, más aún usando pyproject.toml , me leí mucha documentación, junto a prueba y error en testpypi.org y la muy poca información de setuptools para esto, hasta tener casi completo el archivo toml, faltando solo un bug en que no me subía los archivos que no son .py , los terminé agregando al archivo MANIFEST.in porque ninguna opción de setuptools funcionaba.
- Hacer que la carga de módulos sea dinámica sin tener que agregarlos al inicio de los archivos, además de que pueden variar según el nombre que le dé el desarrollador, me llevo a encontrar la forma de cargarlos con importlib, haciendo incluso un método para cambiar el nombre a CamelCase para la lectura de la clase. Esto haría la carga de controladores, y posteriormente lo reutilicé para la carga de los helpers.
- El armado del instalador no fue sencillo, tenía la idea de como armarlo pero no sabía donde empezar, de a poco fue tomando forma, armando el paso a paso que le pregunta al desarrollador las opciones de configuración a elegir, además de que copia archivos base (en "pmvcs\setup\defaults\") y modifica sobre estas configuraciones el nombre a tener, agrega datos de ejemplo si se quiere (en "pmvcs\setup\example\") y luego la opción de agregar más módulos (en menú) hace lo mismo (en "pmvcs\setup\modules_base\") o bien de módulo único sin menú (en "pmvcs\setup\module_unique\")

CUMPLIMIENTO DE REQUERIMIENTOS POR PARTE DE CÓDIGO FACILITO

- Python en una versión superior igual a la 3.6.
  -> Requiere Python 3.10+ (también testeado en 3.11)
- Implementación de programación orientada a objetos.
  -> Un ejemplo: https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/router.py
- Uso de interfaces, herencias y encapsulamiento.
  -> Interfases: https://github.com/gsmx64/pmvcs/tree/main/pmvcs/core/interfaces
  -> Herencias: https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/models/language.py
  -> Encapsulamiento:
  https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/controllers/base_controller.py
- Implementación de programación funcional
  -> Un ejemplo: https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/setup/setup_parser.py

- Implementación de programación en concurrente y/o paralelo.
  -> No necesario para este proyecto
- Uso de decoradores.
  -> Ejemplo decorador:
  https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/decorators/decorators_views.py
  -> Decorado: https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/views/base_view.py
- Uso de alguna estructura de datos.
  -> Un ejemplo de transformación de ConfigParser a dict:
  https://github.com/gsmx64/pmvcs/blob/main/pmvcs/core/models/parser.py
- Archivos de pruebas unitarias.
  -> En: https://github.com/gsmx64/pmvcs/tree/main/pmvcs/tests
- Uso de módulos y paquetes documentados mediante docstrings.
  -> Todo el proyecto lo tiene.
- Uso correcto de PEP8 y PEPs en general.
  -> Todo el proyecto lo tiene con excepción de la cantidad de caracteres por línea, se me hace muy difícil la lectura de un código tan cortado y más aún si se mantienen nombres descriptivos para métodos, variables y constantes)
- Repositorio público en GitHub.
  -> https://github.com/gsmx64/pmvcs/
- Archivo README.md donde se detalla el cómo ejecutar el proyecto. (Formato Markdown).
  -> Inglés: https://github.com/gsmx64/pmvcs/blob/main/README.md
  -> Español: https://github.com/gsmx64/pmvcs/blob/main/README.ES.md
- Archivo(s) .py con el código de la implementación.
  -> Ver todo el proyecto en el repositorio.
- Archivo requirements.txt
  -> Traté de no utilizar paquetes externos, igualmente está en:
  https://github.com/gsmx64/pmvcs/blob/main/requirements.txt
- Documentación.
  -> Ver READMEs y en Wiki https://github.com/gsmx64/pmvcs/wiki
- Archivo .pdf donde se detalle el funcionamiento del programa.
  -> Ver READMEs y el presente documento.
- En el archivo se deben describir los retos presentados en la elaboración del proyecto y cómo fueron solucionados.
  -> Ver en el presente documento.

INSTALACIÓN DE PMVCS

Se debe crear el entorno virtual con:

Windows: **python -m venv env** y activarlo: **env\Scripts\activate**

Linux/Unix/Mac: **python3 -m venv env** y activarlo: **source ./env/bin/activate**

Instar PMVCS con pip o pip3: **pip install pmvcs**

Instalación inicial desde la consola en español:

```
>>> python -m pmvcs.cli setup -l es
```

o

>>> python -m pmvcs.cli setup -language es

o

>>> pmvcs-cli setup -l es

o

>>> pmvcs-cli setup -language es

Configuración, paso a paso, en pantalla:

Inserta el nombre para la APP (predeterminado: "MyApp", deja en blanco):

>>> Mi App de Testeo

Inserta el nombre de carpeta para la APP (default: "app", deja en blanco):

>>> app

¿Habilitar depuración? (predeterminado: "No")

(Y) Sí - (N) No:

>>> Y

¿Ver información del proyecto al salir? (predeterminado: "Sí")

(Y) Sí - (N) No:

>>> Y

¿Habilitar soporte Multi-Idioma? (predeterminado: "Sí")

(Y) Sí - (N) No:

>>> Y

Selecciona el idioma predeterminado (predeterminado: "(1)")

(1) English - (2) Español:

>>> 2

¿Instalar datos de ejemplo, usar menú para múltiples módulos o un único módulo personalizado?

(E) Datos de ejemplo

(M) Múltiples módulos

```
(S) Único módulo personalizado


Opción:
```

>>> E   -> Instala datos de ejemplo, un módulo en el menú con archivos controlador, modelo y vista.

>>> M   -> Instala múltiples módulos en menú con archivos controlador, modelo y vista cada uno.

>>> S   -> Instala un único módulo sin menú con archivo controlador.

Selecciona el deseado. Finalizada la configuración.

AGREGAR MÚLTIPLES MÓDULOS

Instala nuevos múltiples archivos del módulo en el menú con controlador, modelo y vista cada uno.

Para agregar nuevos módulos en el menú, ejecuta lo siguiente:

>>> python -m pmvcs.cli menu -l es

o

>>> python -m pmvcs.cli menu -language es

En pantalla:

```
Nombre actual de carpeta de la APP:
```

>>> app01

```
Inserta el nombre del módulo:
```

>>> example_two

```
¿Agregar otro módulo?
(Y) Sí - (N) No:
```

>>> N

AYUDANTES o HELPERS

Ayudantes propios de PMVCS

Puedes cargar ayudantes de PMVCS, por ejemplo:

```
viejo_valor = str('5')
print(type(viejo_valor))
```

```
>>> <class 'str'>

filters = self.pmvcs_helper.load_helper('filters', True)
nuevo_valor = filters.data_type(viejo_valor)

print(type(nuevo_valor))
>>> <class 'int'>
```

El ayudante "filters" devuelve un valor entero desde un número en cadena.

AYUDANTES PERSONALIZADOS

Puedes cargar ayudantes personalizados, por ejemplo:

```
example = self.pmvcs_helper.load_helper('example')
example.my_func()
```

Ruta para almacenar tu ayudante:

```
(carpeta_de_la_app)/helpers/
```

Formato en el archivo del ayudante:

```
from pmvcs.core.helpers.base_helper import BaseHelper

class ExampleHelper(BaseHelper):
    """ Class for Example Helper """

    def __init__(self, **kwargs) -> None:
        """
        Init PMVCS Example Helper requirements
        """
        super().__init__(**kwargs)

    def my_func(self):
        """
        Returns a float or int value
        """
        pass
```

Ayudantes Personalizados: Pasar variables

En el __init__ debería tener:

```
def __init__(self, **kwargs) -> None:
      self.pmvcs_helper = kwargs['pmvcs_helper']
      self.kwargs = { 'pmvcs_cfg': kwargs['pmvcs_cfg'],
                              'pmvcs_lang': kwargs['pmvcs_lang'],
                              'pmvcs_helper': kwargs['pmvcs_helper']}
```

Y en la función que llama al ayudante:

```
def to_string_table(self, data: dict) -> str:
      kwargs2 = { 'data': data,
                        'file_name': 'temp_file'}
      kwargs2.update(self.kwargs)
      table_helper = self.pmvcs_helper.load_helper('table', **kwargs2)
      table_helper.record_file()
```

Finalmente, en el ayudante recuperamos los valores como:

```
def __init__(self, **kwargs) -> None:
      super().__init__(**kwargs)

      self._data = kwargs['data']
      self._file_name = f"{kwargs['file_name']}.{self._file_extension}"
```

OBTENER CONSTANTES DE CONFIGURACIÓN DE CONFIG.INI

Obtiene una constante de configuración en tipo cadena desde "OPTIONS".

Código:

```
>>> self.cfg.get("EXAMPLE_CONSTANT", "OPTIONS")
```

Retorna:

```
This is an example value from config file
```

Tipo de Valor:

```
<class 'str'>
```

Obtiene una constante de configuración en tipo cadena desde "DEFAULT":

Código:

```
>>> self.cfg.get("DEFAULT_TITLE", "DEFAULT")
```

Retorna:

```
Example App
```

Tipo de Valor:

```
<class 'str'>
```

Obtiene una constante de configuración en tipo entero:

Código:

```
>>> self.cfg.get("EXAMPLE_INT", "OPTIONS", "int")
```

Retorna:

```
8
```

Tipo de Valor:

```
<class 'int'>
```

Obtiene una constante de configuración en tipo flotante:

Código:

```
>>> self.cfg.get("EXAMPLE_FLOAT", "OPTIONS", "float")
```

Retorna:

```
1.57
```

Tipo de Valor:

<class 'float'>

Obtiene una constante de configuración en tipo booleano:

Código:

```
>>> self.cfg.get("EXAMPLE_BOOLEAN", "OPTIONS", "boolean")
```

Retorna:

```
True
```

Tipo de Valor:

```
<class 'bool'>
```

Obtiene una constante de configuración en una lista:

Código:

```
>>> self.cfg.get("EXAMPLE_LIST", "OPTIONS", "list")
```

Retorna:

```
['1', '2']
```

Tipo de Valor:

```
<class 'list'>
```

Obtiene una constante de configuración en un diccionario:

```

```

Código:

```
>>> self.cfg.get("EXAMPLE_DICT", "OPTIONS", "dict")
```

Retorna:

```
{'value_one': '1', 'value_two': '2'}
```

Tipo de Valor:

```
<class 'dict'>
```

OBTENER CONSTANTES DE IDIOMA DESDE LANGUAGES/ES.INI:

Obtiene la etiqueta de idioma actual:

Código:

```
>>> self.lang.tag
```

Retorna:

```
en
```

Tipo de Valor:

```
<class 'str'>
```

Obtiene una constante de idioma:

Código:

```
>>> self.lang.get("LANG_EXAMPLE_STRING")
```

Retorna:

```
This is an example string
```

Tipo de Valor:

```
<class 'str'>
```

Obtenga una constante de idioma pasando un valor en String-Print-Format:

En el archivo de idioma verá, por ejemplo: `"El valor aquí: "{}"`

Código:

```
>>> self.lang.sprintf("LANG_EXAMPLE_SPRINTF", "3")
```

Retorna:

```
String-Print-Format value here: "3"
```

Tipo de Valor:

```
<class 'str'>
```

Obtenga una constante de idioma pasando varios valores en String-Print-Format:

En el archivo de idioma verá, por ejemplo: `"Uno: "{}". Dos: "{}". Tres: "{}".`

Código:

```
>>> self.lang.sprintf("LANG_EXAMPLE_SPRINTF2", "1", "2", "3")
```

Retorna:

```
One: "1". Two: "2". Three: "3".
```

Tipo de Valor:

```
<class 'str'>
```

Traduciendo una cadena:

Código:

```
>>> value = 'dictionary'
>>> self.lang.translate(value)
```

Retorna:

Dictionary

Tipo de Valor:

```
<class 'str'>
```

USO DE LAS FUNCIONES DE VISTA DE PMVCS

Esto muestra la pancarta con el nombre de la App:

Código:

```
>>> self.pmvcs_view.get_intro()
```

Esto muestra el mensaje de la salida:

Código:

```
>>> self.pmvcs_view.get_exit()
```

Esto inserta un salto de línea sin print():

Código:

```
>>> self.pmvcs_view.line_brake()
```

Esto inserta un salto de línea con print():

Código:

```
>>> self.pmvcs_view.line_brake(True)
```

Esto inserta una pausa para presionar ENTER para comenzar:

Código:

```
>>> self.pmvcs_view.input_start()
```

Esto inserta una pausa para presionar ENTER para continuar:

Código:

```
>>> self.pmvcs_view.input_pause()
```

Esto inserta un input de selección de opción:

Código:

```
>>> self.pmvcs_view.input_options()
```

Esto inserta un input:

Código:

```
>>> self.pmvcs_view.input_generic(text)
```