Universidad Carlos III de Madrid

Curso Desarrollo de Software 2021-22

Ejercicio Guiado 2

Fecha: **25/02/2022**

GRUPO: **50** GRUPO DE PRÁCTICAS: **07**

Alumno: **José David Rico Días** NIA: **100441800**

Alumno: **Raúl Alejando Álvarez Conesa** NIA: **100408885**

Organización de archivos

A continuación se especifica la forma de organizar los archivos fuente, así como la leyenda necesaria para todos los archivos fuente.

Archivos de Código Fuente

Organización jerárquica de los archivos

Todos los proyectos deberán organizarse jerárquicamente de la siguiente forma:

```
main
├── build
├── data
│   ├── datosproyecto1
│   └── datosproyecto2
├── dist
├── doc
│   └── documentación
├── nombre_módulo1
│   ├── include
│   │   └── include1
│   ├── src
│   │   └── archivofuente1
│   └── archivo_fuente2
├── scripts
│   ├── script1
│   └── script2
├── tests
│   ├── archivotestgloblal1
│   ├── archivotestglobal2
│   ├── ...
```

- `main`: archivo fuente principal del proyecto.
- `build`: ficheros intermedios necesarios para generar el ejecutable final.
- `dist`: programa ejecutable final.
- `doc`: documentación relativa al proyecto.
- `include`: archivos fuente de definición de clases, si la definición de la clase está separada de su implementación.
- `src`: archivos fuente de implementación de métodos de clases, si la definición de la clase está separada de su implementación.
- `scripts`: scripts, que sin ser código fuente, son necesarios para el proyecto.
- `tests`: archivos de test del proyecto.

Esta norma puede modularse dependiendo de las características del proyecto. No todos los lenguajes necesitan compilación intermedia (véase C++), ni todos los proyectos necesitan archivos de datos.

El archivo `main` **nunca** contendrá la definición de los métodos, sólo su uso.

Leyenda de archivos de Código Fuente

Todos los ficheros fuente deberán seguir el siguiente formato de leyenda al principio del archivo; antes de cualquier instrucción de código:

```
"""
@description:
@author:
@email:
@version:
@date:
@warning:
"""
```

- `@description` será incluida una descripción de máximo 10 líneas donde será explicado brevemente el fichero.
- `@email` será incluido el autor del fichero con nombre y apellidos. Si hay más de un autor, se incluyen todos.
- `@version` será incluido un código del formato `XX.XX.XX` especificando la primera parte la *major release*, la segunda parte la *minor release* y la última parte el *bugfix*. Cuando un fichero pasa por primera vez a producción, se considera su primera *major release*.
- `@date` se incluye la fecha del último *bugfix*. En formato DD/MM/YY
- `@warning` se incluye una descripción de máximo 15 líneas donde serán explicados fallos conocidos, bugs no corregidos, casos límites no cubiertos, etc. Es opcional rellenar esta etiqueta.

Ejemplo:

```
"""
@description: Fichero de pruebas para el módulo ejemplo.py
@author: José David Rico Días
@email: 100441800@alumnos.uc3m.es
@version: 0.1.0
@date: 24/02/22
@warning:
"""
```

Gestión de archivos de Código Fuente

Todos los proyectos deberán gestionarse usando la herramienta de control de versiones Git. Cada desarrollador deberá usar únicamente un usuario en Git. En los `commit` deberá figurar el nombre del desarrollador.

Ficheros de Clases

Todas las definiciones de clases deben estar en un único fichero. Sin embargo, la definición de los métodos de clases pueden realizarse en archivos de mismo nombre pero distinta extensión.
Ejemplo:
`Cronómetro.h` incluye la definición de la clase `Cronómetro`.

`Cronómetro.src` incluye la implementación de los métodos de la clase `Cronómetro`.
`Dirección.py` incluye la definición e implementación de los métodos de la clase `Dirección`.

Todos los ficheros de clases deben seguir la leyenda especificada en *Leyenda de Archivos de Código Fuente*.

Las clases incluirán un docstring siguiendo el siguiente formato:

```
"""
@description:
@throw:
@warning:
"""
```

Nombres y Variables

Variables

Las variables serán nombradas usando el estilo snake_case.
Se prohíben las variables `foo, bar, baz, toto, tutu, tata, temp, asd, bla.` Así mismo se prohíben caracteres individuales (regex: `\b[a-zA-Z]\b`), variables que empiezen por var (regex: `\bvar*\b`), y variables que empiezen por id (regex: `\bid*\b`).

Se permiten explícitamente los siguientes nombres de variables: `i, j, k, ex, Run, counter`.

Clases y miembros de clases

Nombrado de clases

Todas los nombres de clases y los ficheros de clases deberán seguir el formato `PascalCase`.

Nombrado de atributos de clases

Los atributos de las clases deberán seguir la siguiente expresión regular:
`\b__[a-z]+(_[a-z]+)*__\b`
Ejemplo:

```
class Usuario:
    def __init__(self, telefono, alias):
        self.__telefono__ = telefono
        self.__alias_jugador__ = alias
```

Visibilidad

Variables de instancia de clases

El número máximo de atributos de clase son 15.
Las variables de clases deberán accederse a través de un método de clase de mismo nombre en formato `camelCase`. Deberá usar el decorador `@property`.
Para establecer el valor de la variable, deberá usarse una función de mismo nombre en formato `camelCase`, pero que acepte un argumento adicional.
Ejemplo:

```
class Login:
   def __init__(self, nombre_usuario):
       self.__nombre_usuario__ = nombre_usuario
   @property
   def nombreUsuario(self):
       return self.__nombre_usuario__
   @nombreUsuario.setter
   def nombreUsuario(self, nombre_usuario):
       self.__nombre_usuario__ = nombre_usuario
```

Inicialización de variables

Las variables de uso local a los métodos deberán inicializarse (preferentemente) justo antes de su uso.
Sería correcto:

```
def rellenaBotella(cantidad):
    ...
    disponible = self.__capacidad__ - self.__volumen_actual__
    if (disponible >= cantidad):
        ...
```

Sería incorrecto:

```
def rellenaBotella(cantidad):
    disponible = self.__capacidad__ - self.__volumen_actual__
    ...
    ...
    if (disponible >= cantidad):
       ...
```

Métodos y estructuras condicionales

Estructura de los métodos

El número máximo de líneas en un método serán 90.
Si el número de argumentos del método hace exceder el máximo de ancho de línea, deberán alinearse al principio primer argumento, en la siguiente línea.

```
def generaMapa(posicion_x, posicion_y, posicion_z, altura_maxima,
               semilla, premio_maximo):
    ...
```

Declaración de los métodos

Todos los métodos siguen el formato `camelCase`; a excepción de métodos de clase por defecto (constructores) o instanciación (__new__, __init__).
En la línea anterior a la definición del método se debe incluir un docstring que siga el siguiente formato:

```
"""
@description:
@param:
@param:
@return:
@return:
@throw:
@warning:
"""
```

Donde:

- `@description` : Descripción del método.
- `@param`: Parámetro 1 del método.
- `@param`: Parámetro 2 del método.
- `@return`: Return 1 del método.
- `@return`: Return 2 del método.
- `@throw`: Excepciones del método.
- `@warning`: Precauciones o bugs del método.

La etiqueta `@param` puede repetirse hasta el máximo número permitido de argumentos.
La etiqueta `@return` puede repetirse hasta el máximo número permitido de return.
La etiqueta `@throw` puede repetirse ilimitadamente, una por cada excepción.

Estructura de los if-else

Todos las estructuras condicionales if deben contener paréntesis para encerrar la condición. Como máximo se podrán incluir 5 condiciones.
El bloque a ejecutar si se cumple la condición deberá situarse en la línea inferior.
Si las condiciones superan el máximo de líneas permitido, se deberá usar el formato de tabla.
Ejemplo:

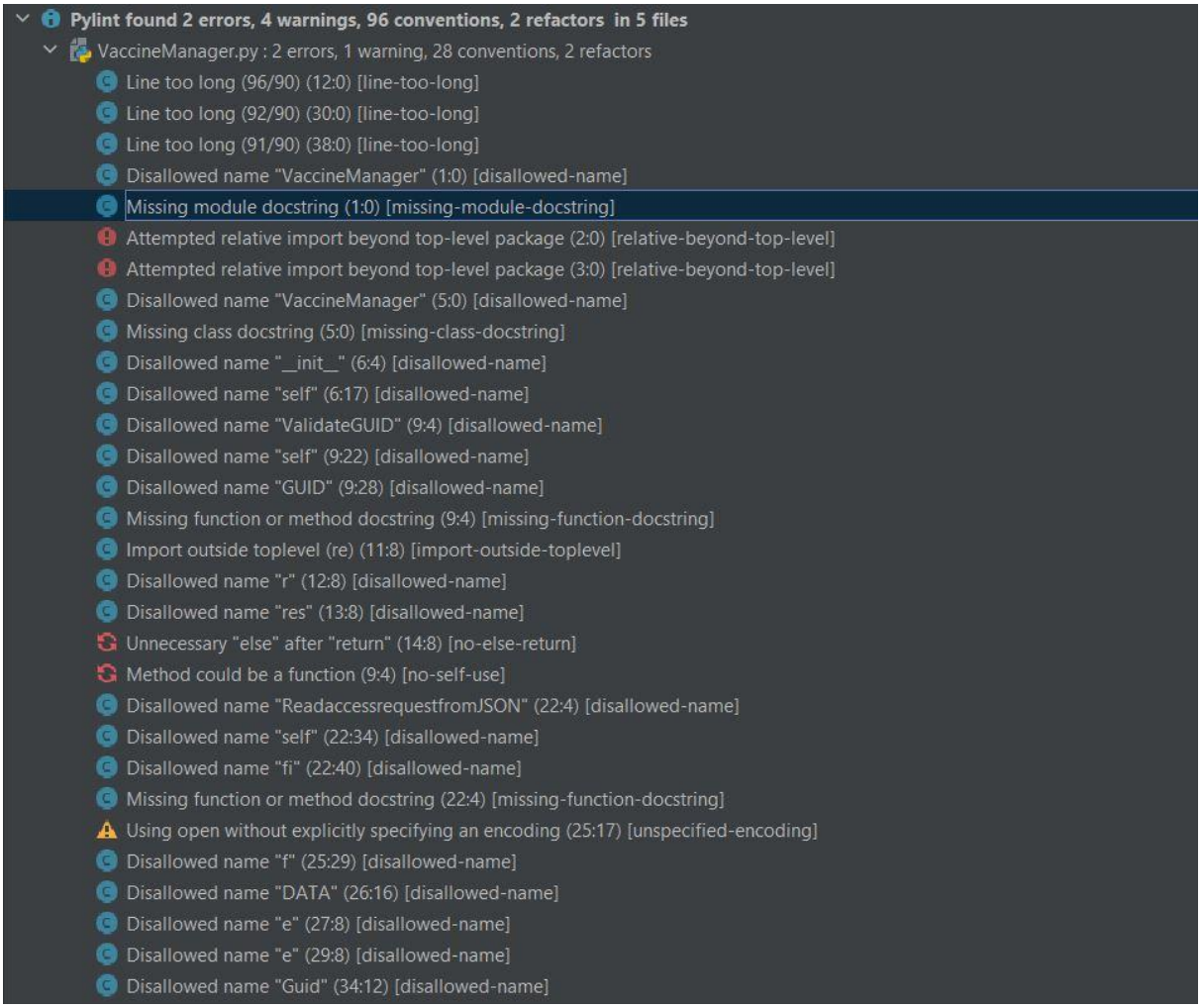
```
if(altura > 1500):
    descender()
else:
    altura = avion.Altura()
```

Tratamiento de Excepciones

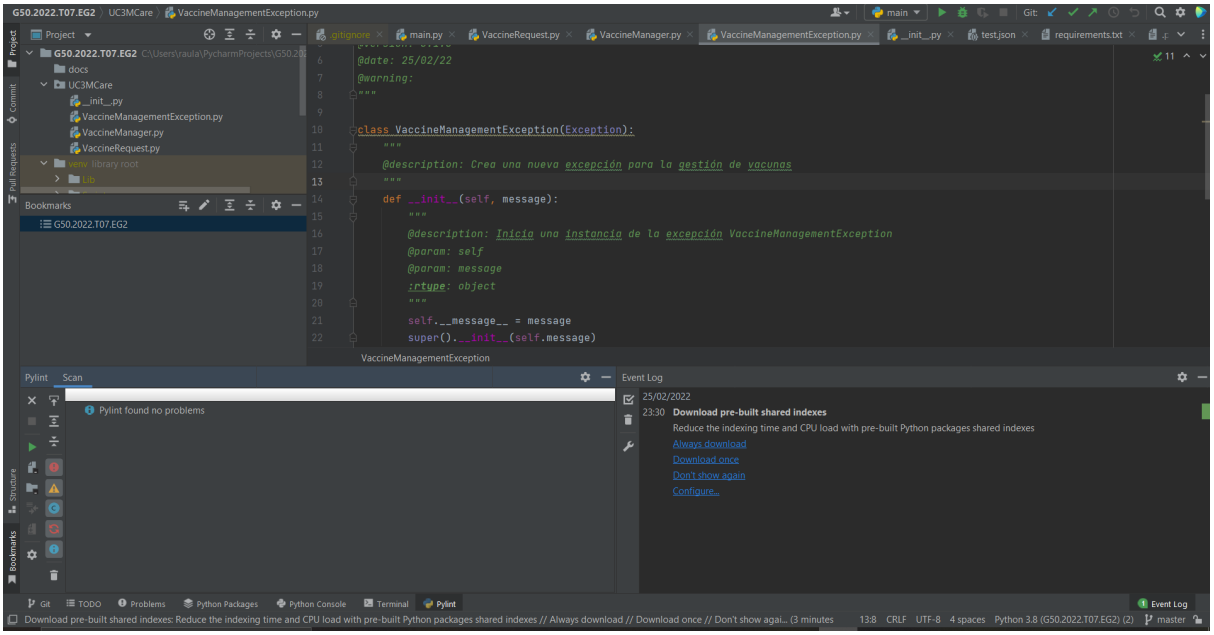
Todos los métodos que puedan, potencialmente, lanzar una excepción deberán encerrar el código conflictivo usando la estructura try-except. En primer lugar se tratarán las excepciones estándar del lenguaje, a continuación las excepciones creadas en el proyecto y por útltimo la excepción base.

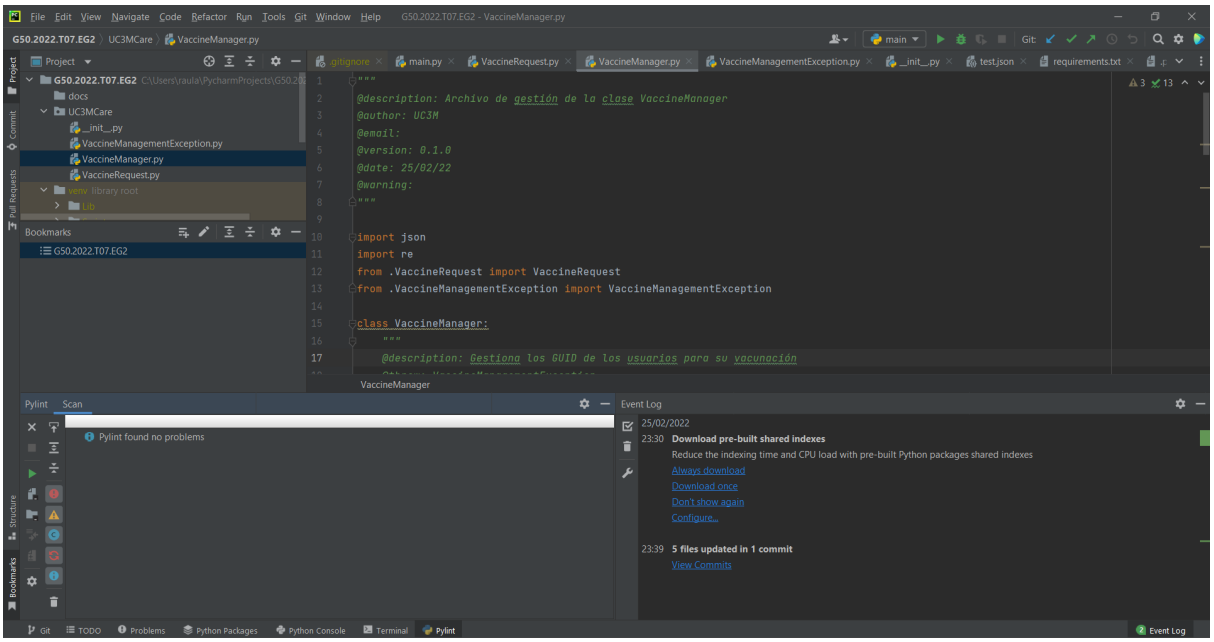
Aplicación del estándar al código propuesto

Código pre-limpieza



Código post-limpieza


G50.2022.T07.EG2 > UC3MCare > VaccineManagementException.py
main
Git:
Project
G50.2022.T07.EG2
docs
UC3MCare
__init__.py
VaccineManagementException.py
VaccineManager.py
VaccineRequest.py
venv library root
Lib
Bookmarks
G50.2022.T07.EG2
.gitignore
main.py
VaccineRequest.py
VaccineManager.py
VaccineManagementException.py
__init__.py
test.json
requirements.txt
@date: 25/02/22
@warning:
"""
class VaccineManagementException(Exception):
"""
@description: Crea una nueva excepción para la gestión de vacunas
"""
def __init__(self, message):
"""
@description: Inicia una instancia de la excepción VaccineManagementException
@param: self
@param: message
:rtype: object
"""
self.__message__ = message
super().__init__(self.message)
VaccineManagementException
Pylint
Scan
Pylint found no problems
Event Log
25/02/2022
23:30 Download pre-built shared indexes
Reduce the indexing time and CPU load with pre-built Python packages shared indexes
Always download
Download once
Don't show again
Configure...
Git
TODO
Problems
Python Packages
Python Console
Terminal
Pylint
Event Log
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show agai... (3 minutes
13:8
CRLF
UTF-8
4 spaces
Python 3.8 (G50.2022.T07.EG2) (2)
master


File Edit View Navigate Code Refactor Run Tools Git Window Help
G50.2022.T07.EG2 - VaccineManager.py
G50.2022.T07.EG2 > UC3MCare > VaccineManager.py
main
Git:
Project
G50.2022.T07.EG2
docs
UC3MCare
__init__.py
VaccineManagementException.py
VaccineManager.py
VaccineRequest.py
venv library root
Lib
Bookmarks
G50.2022.T07.EG2
.gitignore
main.py
VaccineRequest.py
VaccineManager.py
VaccineManagementException.py
__init__.py
test.json
requirements.txt
"""
@description: Archivo de gestión de la clase VaccineManager
@author: UC3M
@email:
@version: 0.1.0
@date: 25/02/22
@warning:
"""
import json
import re
from .VaccineRequest import VaccineRequest
from .VaccineManagementException import VaccineManagementException
class VaccineManager:
"""
@description: Gestiona los GUID de los usuarios para su vacunación
VaccineManager
Pylint
Scan
Pylint found no problems
Event Log
25/02/2022
23:30 Download pre-built shared indexes
Reduce the indexing time and CPU load with pre-built Python packages shared indexes
Always download
Download once
Don't show again
Configure...
23:39 5 files updated in 1 commit
View Commits
Git
TODO
Problems
Python Packages
Python Console
Terminal
Pylint
Event Log