CURSO DE PROGRAMACIÓN FULL-STACK

Python / Guía 7

Ejercicios

Código ordenado, limpio y reutilizable

Ejercicios de aprendizaje

1. Crea las funciones necesarias para que el usuario ingrese dos números y el programa devuelva el resultado de una división. Prevé los posibles errores del usuario. Luego, crea un paquete llamado `'mi_division'`. En él crea un archivo para cada función respectivamente. Fuera del paquete, en un nivel superior, crea un archivo `'main.py'` para que sea el script que importe el paquete y ejecute el programa. Pon en práctica las mejores prácticas que has aprendido.

Ver video "Paquetes"

2. Mira la guía anterior llamada "Funciones" (en la parte de Ejercicios de Aprendizaje Extra. A partir de los ejercicios 1, 2, 3, 4 5 y 6 que ya están resueltos, crea un paquete que contenga los siguientes módulos:

   a. `área_rectángulo`
   b. `área_círculo`
   c. `relación`
   d. `intermedio`
   e. `recortar`
   f. `separar`

   Agrega las anotaciones de tipo para cada función y crea un script que, por medio de un menú, el usuario pueda elegir qué desea hacer. Maneja los posibles errores para que el programa no se detenga. Haz pruebas unitarias.

3. A partir de los ejercicios del apartado anterior, Funciones, crea un paquete que calcule el `año_bisiesto.` Crea los módulos necesarios para la validación de datos y la interacción con el usuario. Usa la anotación de tipos. Haz pruebas unitarias creando una carpeta tests en el proyecto.

4. A partir de los ejercicios del apartado anterior, Funciones, crea un paquete `conversor_tiempo`, con los mismos requisitos de los ejercicios anteriores. Haz pruebas unitarias creando una carpeta tests en el proyecto.

5. Aplicación

Siguiendo con el ejercicio del apartado anterior, crea una aplicación donde el usuario vea el siguiente menú:

NEGOCIO
1. Artículos
2. Clientes
3. Ventas
x. Salir

Al ingresar x, saldrá del programa.
Las demás opciones no están disponibles en este ejercicio.

Al ingresar "1", que será la única opción disponible, verá el siguiente menú:

ARTÍCULOS
1. Ver artículos
2. Agregar un artículo
3. Modificar un artículo
4. Eliminar un artículo
x. Salir

Al ingresar x, volverá al menú anterior.
Las únicas opciones disponibles en este ejercicio serán 1 y 2.

Al ingresar la opción "2", el usuario ingresará a un formulario, en el que validará y guardará:

Código (tipo de dato: **int** mayor a 0 y menor a 1.000.000)
Nombre (tipo de datos: **str** de longitud mayor a 3 y menor a 100)
Precio (tipo de datos: **float** mayor a 0)
Cantidad (tipo de datos: **int** mayor a 0 y menor a 1.000.000)

Si el código del artículo ya existe, mostrar el mensaje "El código ya existe en otro artículo". En caso de error en uno de los campos, el usuario debe volver a ingresar el valor, pero sin llenar todo de vuelta.

Al ingresar la opción "1", el usuario podrá ver todos los artículos que ingresó, aun si regresó al menú anterior y luego volvió a agregar artículos.

Requisitos

Crea un nuevo proyecto llamado **Negocio**
Crea un **entorno virtual ".venv"**, y actívalo.
Instala **MyPy y Pytest**

Te recomiendo que uses Git en este momento, ya que es una herramienta fundamental para cualquier desarrollador. Te permite trabajar en equipo, volver hacia atrás cuando te equivocas, tener versiones de lo que desarrollas, y muchas cosas más. La industria del desarrollo de software ha adoptado esta herramienta de forma universal. En este momento deberías crear un archivo `.gitignore` agregar .venv en ese archivo, y luego inicializar Git.

Parte Técnica I: Paquete validación

El usuario va a ingresar datos. Es necesario validar sus entradas.

a) Crea el **paquete "validación".**

b) En "validación": Crea el **módulo "convertir"** y en él crea las siguientes funciones para que el programa pueda hacer la conversión de tipos de datos:

> **`a_int :`** recibe un parámetro "entrada" de cualquier tipo de datos. La función valida la entrada para convertir a un int. Devuelve la misma entrada convertida a int. Devuelve False si hay un error.
>
> **`a_float :`** recibe un parámetro "entrada" de cualquier tipo de datos. La función valida la entrada para convertir a un float. Devuelve la misma entrada convertida a int. Devuelve False si hay un error.
>
> **`a_str:`** recibe un parámetro "entrada" de cualquier tipo de datos. La función valida la entrada para convertir a un str. Solo acepta convertir int, float y str. Devuelve la misma entrada convertida a str. Devuelve False si hay un error.

c) Corre MyPy sobre el módulo "convertir", y corrige las anotaciones de tipo.

d) En "validación": Crea el **paquete "tests"** para hacer pruebas unitarias para el paquete "validaciones". Dentro de "tests", crea el **módulo "test_convertir".**

Dentro de "test_convertir" crea las siguientes funciones para probar cada función del módulo "convertir":

```
test_convertir_a_int
test_convertir_a_float
test_convertir_a_str
```

e) Corre Pytest, y corrige si es necesario.

Ver video "Aplicación Negocio 1 - Validaciones / Tests"

f) En "validación": Crea el **módulo "validar"** y en él crea las siguientes funciones para que el programa valide datos ingresados de forma genérica: enteros naturales, cadenas no vacías, números reales positivos. Utiliza el módulo "convertir" importándolo.

`entero_natural` (sirve para validar que un número sea un entero mayor a 0)
`cadena_vacía` (sirve para validar que la longitud de una cadena no sea 0)
`real_positivo` (sirve para validar si un número con decimales sea mayor a 0)

g) Corre MyPy sobre el módulo "validar", y corrige las anotaciones de tipo.

h) En "validación/tests" crea el **módulo "test_validar"** que contendrá las siguientes funciones para probar cada función del módulo "convertir":

```
test_validar_entero_natural
test_validar_real_positivo
test_validar_cadena_no_vacía
```

i) Corre Pytest, y corrige si es necesario.

Ver video "Aplicación Negocio 2 - Validaciones / Tests"

Parte Técnica II: Estructura de la aplicación

a) Crea paquetes y módulos según la siguiente estructura:

```
.
├── validación
│   ├── __init__.py
│   ├── convertir.py
│   ├── validar.py
│   └── tests
│       ├── __init__.py
│       ├── test_convertir.py
│       └── test_validar.py
├── usuario
│   ├── __init__.py
│   ├── menú.py
│   └── entrada_datos.py
├── artículos
│   ├── __init__.py
│   ├── menú.py
│   ├── datos.py
│   ├── formulario.py
│   ├── funcionalidades.py
│   └── validaciones.py
└── main.py
```

El objetivo del módulo **usuario** será ser el medio de interacción con el usuario. Mostrará cualquier **menú**, para que se elija una opción. También tendrá una interfaz donde el usuario pueda llenar formularios de cualquier paquete, mediante la **entrada de datos**. La entrada de datos se servirá del paquete **validaciones**

El módulo **artículos** tendrá un **formulario** para que el usuario llene:

- código
- nombre
- precio
- cantidad

Cada uno de estos registros tendrá una validación general (del paquete **validación**) y una validación propia (módulo que estará en el paquete artículos):

Validación general:

Código debe ser un número entero natural (mayor a 0).
Nombre debe ser una cadena no vacía.
Precio debe ser un número real positivo (mayor a 0)
Cantidad debe ser un número entero natural (mayor a 0)

Validación propia:

Código debe ser menor a 1.000.000 y no debe repetirse.
Nombre debe tener una longitud mayor a 3 y menor a 100.
Precio debe ser menor a 1.000.000.000
Cantidad debe ser menor a 1000

Los datos del formulario deben guardarse en una variable declarada en el módulo **datos**. La estructura de la variable será la siguiente, se puede llamar artículos, de tal forma que cuando importes la variable su dirección será **artículos.datos.artículos:**

```
{
    "artículo": {
        código: {
            "nombre": str,
            "precio": float,
            "cantidad": int
        }
    }
}
```

El campo "código" es de tipo int, y es la clave del diccionario que contiene nombre, precio y cantidad. En tu módulo, puedes reemplazar la palabra código por "id", para que el módulo y el paquete sean reutilizables con cualquier tipo de datos, como pueden ser clientes, proveedores, etc.

El módulo **funcionalidades** contendrá solamente las 2 partes activas del menú de esta aplicación: ver artículos y agregar artículos.

Parte Técnica III

Desde main.py de la raíz del proyecto debes mostrar un menú principal. Si el usuario ingresa 1, mostrará el menú de artículos. Para esto, es necesario crear **usuario.menú** que tendrá las funciones necesarias para mostrar el menú y ejecutar una acción. Desde main.py se debería pasar como argumentos el

título del menú y las opciones del menú con sus respectivas acciones. Simplemente, trata de hacer funcionar el menú principal, y el menú de artículos, sin efectuar ninguna acción todavía. Por lo tanto, debes crear también **artículos.menú** que contendrá el título, las opciones y las acciones propias de este paquete.

Ver video "Aplicación Negocio 3 - Menú Principal"

Crea **artículos.datos** con el diccionario correspondiente. Crea una variable temporal, con un artículo guardado como valor, para probar desde el menú la funcionalidad: ver artículos. Por lo tanto, crea artículos.funcionalidades que contendrá la función ver, la cual mostrará los artículos guardados.

Crea **artículos.formulario**, que contendrá un diccionario con las claves id, nombre, precio, cantidad (recuerda que cambiamos el nombre código por id, solamente en formulario.py para hacer el código reutilizable) Cada valor de cada clave, será un diccionario, que tendrá como clave "etiqueta" y "acciones", de tal forma que etiqueta es texto que el usuario verá cuando ingrese texto, y las acciones serán las funciones que deben ejecutarse para validar el ingreso. Ejemplo:

```
FORMULARIO = {
    "id": {
        "etiqueta": "Código: ",
        "acciones": (validar.entero_natural,)
    },
}
```

```
Crea artículos.validaciones donde se definirán las funciones
que validen que código sea menor a..., nombre no vacío...,
etc. Agregar este módulo a las acciones del formulario.
```

```
FORMULARIO = {
    "id": {
        "etiqueta": "Código: ",
        "acciones": (validar.entero_natural,
validaciones.código)
```

```
    },
}
```

Ver video “Aplicación Negocio 4 - Funciones Artículos”

En **artículos.funcionalidades** crea la otra función agregar, para agregar **artículos**. Esta función llamará al módulo **usuario.entrada_datos** y le pasará como argumentos a FORMULARIO, de tal forma que **usuario.entrada_datos** ejecutará las acciones correspondientes para interactuar con el usuario. Si el formulario es válido, devolver la estructura de datos `artículos.datos.artículos y actualizarla en artículos.funcionalidades.agregar.`

Ver video “Aplicación Negocio 5 - Ingreso de Datos”