

Tecnicatura Universitaria en Programación

PROGRAMACION I

Trabajo Práctico Integrador

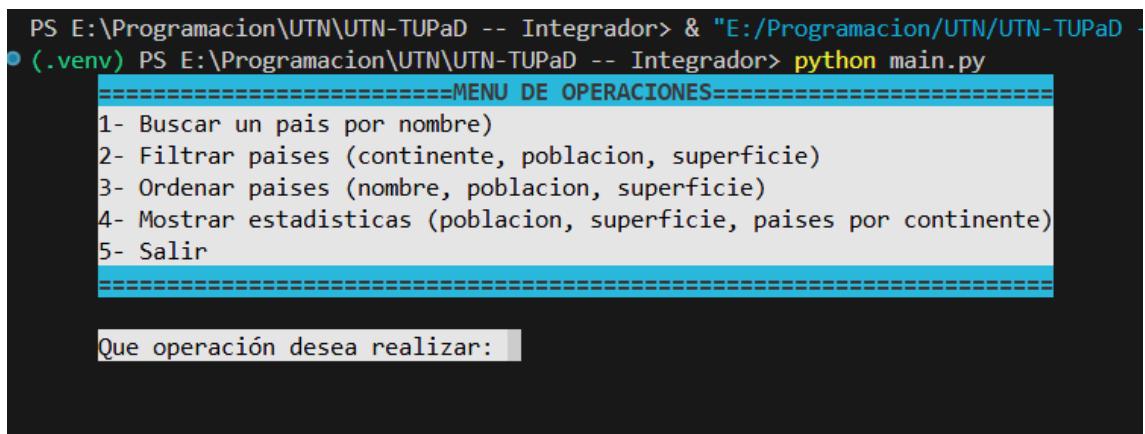
Comisión: 1 prog 2

Integrantes: Morillas Andrés, Ciro Cattaneo

Informe teórico:

Primer paso – Creación de archivo .CSV:

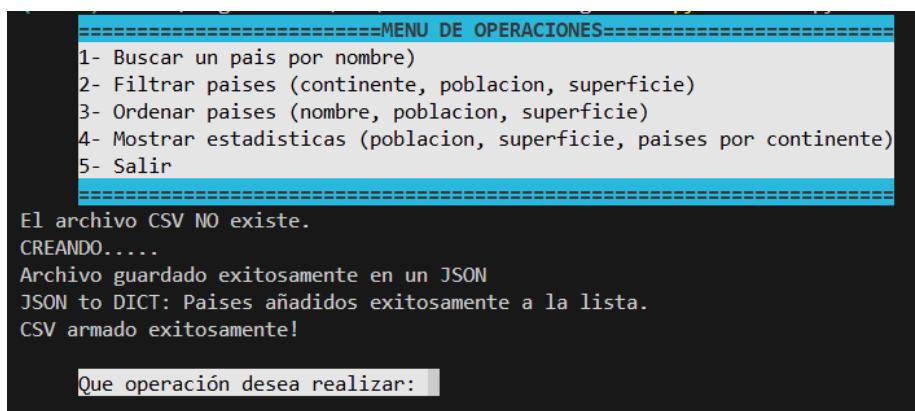
Al momento de desarrollar el proyecto en cuestión, se implementaron todos los conceptos básicos desarrollados y explicados en la materia hasta el día de la fecha. En una aproximación global e inicial se buscó, implementar la persistencia de un menú, a través de bucles con salida por parte del usuario; para implementar 4 funciones principales, encargadas de implementar subfunciones particulares. Cada subfunción, en términos generales, buscaba hacer un manejo de listas de diccionarios, utilizando en ocasiones las keys de los distintos campos, y en otras pasando los valores de dichas keys a strings, o integers que permitían cálculos o verificaciones sencillas.



```
PS E:\Programacion\UTN\UTN-TUPaD -- Integrador & "E:/Programacion/UTN/UTN-TUPaD - (.venv) PS E:\Programacion\UTN\UTN-TUPaD -- Integrador> python main.py
=====
=====MENU DE OPERACIONES=====
1- Buscar un pais por nombre)
2- Filtrar paises (continente, poblacion, superficie)
3- Ordenar paises (nombre, poblacion, superficie)
4- Mostrar estadisticas (poblacion, superficie, paises por continente)
5- Salir
=====
Que operación desea realizar:
```

Vista inicial del menú.

Entrando más en detalle, al requerirse la implementación y uso de un archivo .CSV para almacenar la información de países del mundo; se tuvo que construir una función inicial (*api_build.py*) para encargarse de verificar la existencia de un archivo .CSV; y dada la negativa hacer un llamado a la API para que lo cree.



```
=====
=====MENU DE OPERACIONES=====
1- Buscar un pais por nombre)
2- Filtrar paises (continente, poblacion, superficie)
3- Ordenar paises (nombre, poblacion, superficie)
4- Mostrar estadisticas (poblacion, superficie, paises por continente)
5- Salir
=====
El archivo CSV NO existe.
CREANDO.....
Archivo guardado exitosamente en un JSON
JSON to DICT: Paises añadidos exitosamente a la lista.
CSV armado exitosamente!

Que operación desea realizar:
```

Confirmación de la creación de un archivo .CSV

Tras una breve verificación de la existencia o no de los archivos, mediante la librería OS; la función que llamar a una API de <http://restcountries.com> a través de la función .get() proveniente de la librería Requests.

Tras leer la documentación que ofrecía, se indica que para obtener campos específicos, sin traer toda la información completa, se pueden utilizar parámetros denominados “fields”, los cuales limitan a requests.get() a que obtenga únicamente campos seleccionados por nosotros.

Dicho método genera un archivo .JSON, el cual va a brindar todos los datos obtenidos. Sin embargo, al seguir siendo un archivo que no es el apropiado para el manejo de datos dentro de nuestro sistema, se generó una segunda función para convertir dicho archivo en una lista, donde cada país es un diccionario. Es allí donde se hizo la conversión apropiada de datos, para poder generar un archivo .CSV con las condiciones fijadas por el trabajo integrador. Es decir, con nombre, población, superficie y continente.

Finalmente, una tercera función, constituiría el archivo .CSV usando los métodos de la librería CSV. Es decir el método csv.DictWriter()

Las tres funciones, serían reunidas dentro de un solo archivo; el cual se denominaría api_build.py, con lo cual conceptualmente todo lo relacionado a la generación de .CSV mediante APIs, existiría únicamente aquí.

Como mencioné inicialmente; en el caso de que inicialmente se compruebe la existencia de un archivo .CSV en la raíz del proyecto; esta función no se ejecuta. Encontré que no sería necesario imprimir por pantalla la presencia de un archivo .CSV en la raíz, por lo tanto el programa no ofrece mensaje alguno en esta situación.

Segundo paso – Utilización del archivo .CSV dentro del programa:

Tras haber constituido exitosamente el archivo .CSV a nivel local con toda la información necesaria para la constitución del programa; se necesitaba una función aparte la cual cumpliese con dos funciones principales:

- En primer lugar detectar si existía un archivo .CSV a nivel local, y dada la negativa generarlo (a través de la función `api_build.py`).
- En segundo lugar, existiendo un archivo .CSV, lo convirtiese en una lista de elementos tipo diccionario, para un apropiado manejo de información por parte del programa.

Nuevamente se utilizan métodos de la librería .CSV para el manejo de archivos, permitiendo la lectura de este a través de la función `csv.DictReader()` y la composición de elementos de tipo diccionario, con las limitaciones fijadas por las premisas del trabajo:

- -String para nombre del país.
- -INT para dato de población y superficie territorial.
- -String para dato - continente.

Leido el archivo mediante una función clásica de manejo de archivos, se utiliza un bucle `for`, para la lectura de “`reader`” y la posterior conversión de dicha información en variables locales que, reunidas las cuadros, compondrán un objeto de tipo Diccionario denominado “`pais`”. Dicho elemento, será agregado mediante el método “`.append()`” a la lista principal del menú; la cual será la que se utilizarán en todas las funciones.

Tercer paso: Implementación de funcionalidades, subfunciones, y lógica detrás de su implementación

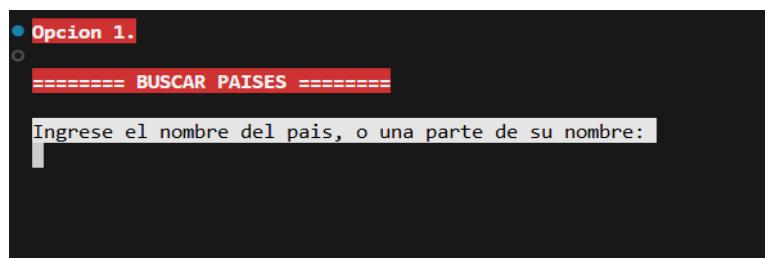
La rúbrica del trabajo práctico exige implementadas las siguientes funcionalidades:

- La búsqueda exacta y parcial de países por información ingresada por el usuario
- El filtrado y procesamiento de información de cada país por distintos criterios (población, continente y superficie territorial)
- El filtrado y ordenamiento de información de cada país también por distintos criterios (alfabético, cantidad de habitantes, y extensión territorial)
- El filtrado y ofrecer estadísticas particulares en el análisis de cada país (países con mayor o menor población, el promedio global de población por país, el promedio de superficie por país, y cuantos países existen en cada continente)

Para implementar las mismas, la mejor manera de realizarlo era a través de una implementación combinada de funcionalidades, que persistían en la consola, gracias a la implementación de bucles. Los bucles que permitían la persistencia de datos se manejan con un input del usuario, quien voluntariamente va a decidir si seguía utilizando otras funcionalidades, o salía completamente del programa.

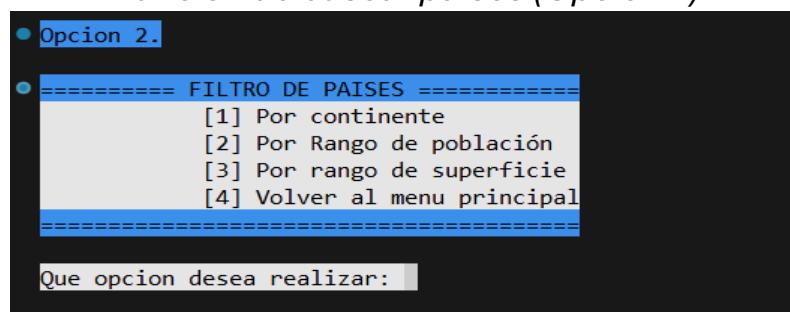
Así se crean 4 funcionalidades principales, las cuales, cumpliendo los puntos mencionados, se denominarían:

- -menu_1_buscar.py
- -menu_2_filtrar.py
- -menu_3_ordenar.py
- -menu_4_estadisticas.py



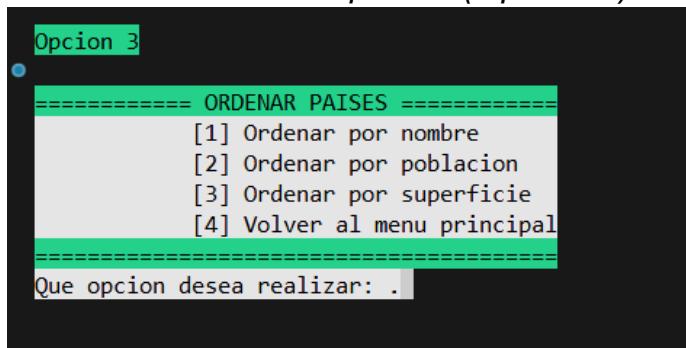
```
● Opcion 1.  
○ ===== BUSCAR PAISES =====  
Ingrese el nombre del pais, o una parte de su nombre:  
I
```

Funcion de buscar países (Opcion 1)



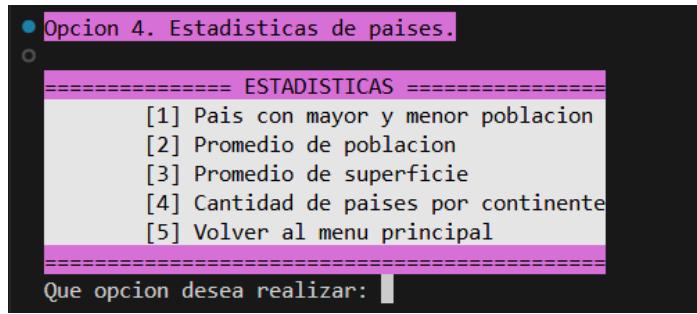
```
● Opcion 2.  
● ===== FILTRO DE PAISES =====  
[1] Por continente  
[2] Por Rango de población  
[3] Por rango de superficie  
[4] Volver al menu principal  
=====  
Que opcion desea realizar:
```

Funcion de filtrar paises (Opcion 2)



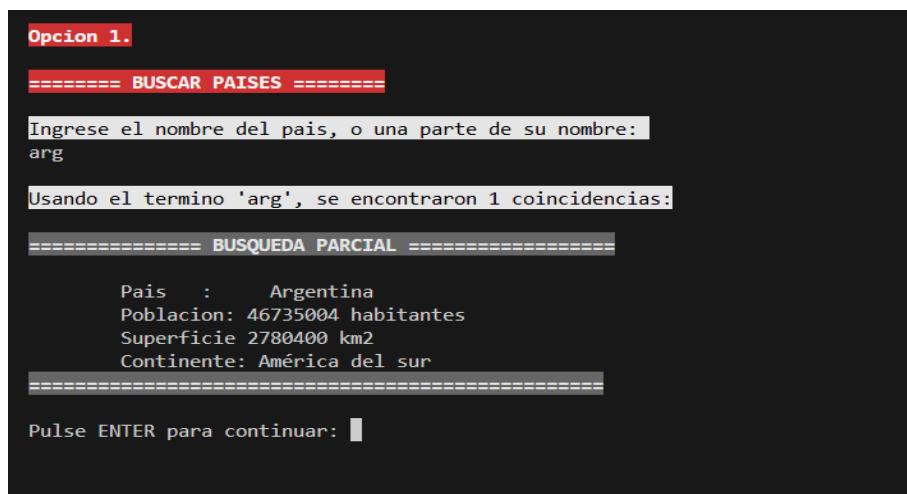
```
Opcion 3  
● ===== ORDENAR PAISES =====  
[1] Ordenar por nombre  
[2] Ordenar por poblacion  
[3] Ordenar por superficie  
[4] Volver al menu principal  
=====  
Que opcion desea realizar: .
```

Funcion de ordenar paises (Opcion 3)

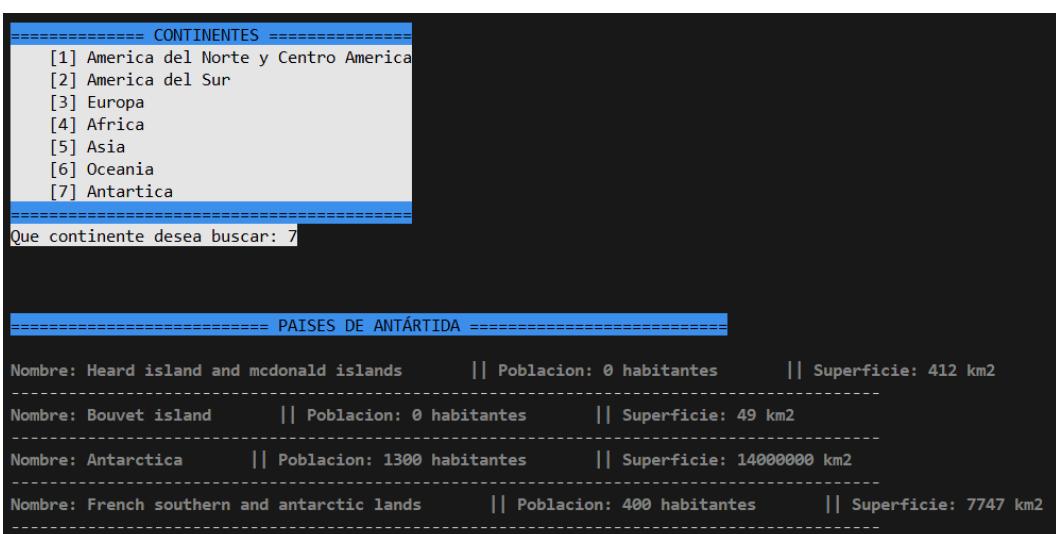


Funcion de obtención de estadísticas (Opción 4)

El usuario tendría por pantalla impresa la opción de ingresar en cada función, por una opción distinta, el cual a su vez tendría subfunciones encargadas de cumplir cada función por separado. Cada función, por una cuestión de claridad y sencillez, tendría creado un archivo, con una nomenclatura sencilla de identificar de a cuál menú corresponde cada script de funciones. A continuación algunas de las funcionalidades que están presentes en los distintos menús.



Busqueda de paises por exactitud parcial



Filtración de países existentes en Antártida

```
===== LISTA ORDENADA: ASCENDIENTE =====
nombre: afghanistan poblacion: 43844000 superficie: 652230 continente: asia
-----
nombre: albania poblacion: 2363314 superficie: 28748 continente: europa
-----
nombre: algeria poblacion: 47400000 superficie: 2381741 continente: africa
-----
nombre: american samoa poblacion: 49710 superficie: 199 continente: oceania
-----
nombre: andorra poblacion: 88406 superficie: 468 continente: europa
-----
nombre: angola poblacion: 36170961 superficie: 1246700 continente: africa
-----
nombre: anguilla poblacion: 16010 superficie: 91 continente: américa del norte
-----
nombre: antarctica poblacion: 1300 superficie: 14000000 continente: antártida
-----
nombre: antigua and barbuda poblacion: 103603 superficie: 442 continente: américa del norte
-----
nombre: argentina poblacion: 46735004 superficie: 2780400 continente: américa del sur
-----
Desea seguir imprimiendo paises[S/N]: [ ]
```

Ordenamiento de países por criterio alfabético ascendente

(se imprime de a 10 países, permitiendo al usuario observarlos a su ritmo)

```
● ===== PROMEDIO POBLACION =====
○ El promedio de poblacion mundial es de 32077981.84 personas por pais.
=====
Pulse ENTER para continuar: [ ]
```

Estadísticas y cálculos, por ej promedio poblacional por país

A su vez, existirían funciones generales que serían utilizadas indistintamente en los cuatro archivos, las cuales no tendrían numeración vinculante, dando a entender su uso general en todo el programa. Principalmente f_validaciones reunirían específicamente las que busquen validar el ingreso de información por parte del usuario, y f_varias reunirían las que no tengan propósito específico.

De esta manera, organizando la estructura de archivos, se podía facilitar la eventual necesidad de agregar o remover funciones, como así testear o debuggear las mismas.

Cuarto paso – Empleo de Colorama como embellecimiento y funcionalidad:

Al momento de integrar funcionalidades de accesibilidad a la consola; se consideró que la utilización de librerías como Colorama, podría

brindarle algun aporte, no solamente desde lo estético, sino tambien de la propia funcionalidad.

Es por ello, que se tomó como criterio principal, darle un color en particular a la utilización de las funcionalidades de cada una de las opciones. Por lo tanto, si nos encontrábamos buscando realizar funciones de ordenamiento de paises, siempre predominaría el color verde; mientras que por otro lado, si se buscaba observar estadísticas, el color magenta estaria presente. De esta manera, como se menciono no solo se buscaba embellecer, sino darle una funcionalidad útil, de cara a la presentación frente al usuario.

```
=====MENU DE OPERACIONES=====
1- Buscar un pais por nombre)
2- Filtrar paises (continente, poblacion, superficie)
3- Ordenar paises (nombre, poblacion, superficie)

Opcion 1.

===== BUSCAR PAISES =====

Opcion 2.

===== FILTRO DE PAISES =====
[1] Por continente
[2] Por Rango de poblacion

Opcion 3

===== ORDENAR PAISES =====
[1] Ordenar por nombre
[2] Ordenar por poblacion
[3] Ordenar por superficie

Opcion 4. Estadisticas de paises.

===== ESTADISTICAS =====
[1] Pais con mayor y menor poblacion
[2] Promedio de poblacion
[3] Promedio de superficie
```

Presentación final

Quinto paso: Composición de imagen y contenedor en Docker

Finalmente, una vez finalizado el proyecto en su totalidad, se realizó la composición de una imagen de Docker.

```

Containers / integrador_contenedor
integrador_contenedor
< 75ede62fa57f integrador:1.2
STATUS Running (44 seconds ago)
Logs Inspect Bind mounts Exec Files Stats

=====MENU DE OPERACIONES=====
1- Buscar un pais por nombre)
2- Filtrar paises (continente, poblacion, superficie)
3- Ordenar paises (nombre, poblacion, superficie)
4- Mostrar estadisticas (poblacion, superficie, paises por continente)
5- Salir
=====

Que operación desea realizar:
=====MENU DE OPERACIONES=====
1- Buscar un pais por nombre)
2- Filtrar paises (continente, poblacion, superficie)
3- Ordenar paises (nombre, poblacion, superficie)
4- Mostrar estadisticas (poblacion, superficie, paises por continente)
5- Salir
=====
```

Presentación del contenedor donde existe el proyecto integrador final

Para poder construir el contenedor con los archivos necesarios y correspondientes, se construyó una imagen utilizando Python:3, y al momento de componer dicho archivo, se generó un “requirements.txt” agregando las dependencias necesarias a instalar. En este caso, únicamente dos las consideramos necesarias: Colorama y Operator.

Una vez reunidos los archivos, se generó una imagen (“integrador:1.2”) ; y a través de dicha imagen se creó un container (“integrador_contenedor”).

Actualmente se encuentra hosteado en hub.docker, en el siguiente enlace:

https://hub.docker.com/repository/docker/agmorillas/tp_integrador_prog1/general

The screenshot shows the Docker Hub interface for a repository named `agmorillas/tp_integrador_prog1`. The repository has a size of 407.9 MB and was last pushed 7 minutes ago. It contains 1 tag, labeled `final`, which is an Image type. The repository is marked as **INCOMPLETE** in the overview section.

Tags

| Tag | OS | Type | Pulled | Pushed |
|--------------------|----|-------|-----------------|-----------|
| <code>final</code> | | Image | less than 1 day | 8 minutes |

[See all](#)

Repository overview INCOMPLETE

An overview describes what your image does and how to run it. It displays in [the public view of your repository](#) once you have pushed some content.

[Add overview](#)

Vista de repositorio actualmente online