PythonBasicCourse-es005

October 15, 2022

1 Curso básico de Python

1.1 Apuntes

Curso básico de Python. Apuntes por Marcelo Horacio Fortino. Versión 2.3. Octubre 2022.

Versión	Autor/es	Fecha	Observaciones
1.0	Marcelo Horacio Fortino	2021/Marzo	Curso Python
1.1	Marcelo Horacio Fortino	2021/Junio	Convertido a markdown - ipynb
1.2	Marcelo Horacio Fortino	2021/Agosto	Actualizados contenidos
1.3	Marcelo Horacio Fortino	2021/Octubre	Agregado Flask microframework
1.4	Marcelo Horacio Fortino	2021/Noviembre	Agregado Pandas - datascience
1.5	Marcelo Horacio Fortino	2021/Diciembre	Agregado Devops - Ansible
2.0	Marcelo Horacio Fortino	2022/Abril	Nueva estructura: core / module
2.1	Marcelo Horacio Fortino	2022/Junio	Módulo apuntes intermedio
2.2	Marcelo Horacio Fortino	2022/Agosto	Actualizado temario y ejercicios
2.3	Marcelo Horacio Fortino	2022/Octubre	Actualizado despliegue a render.com

cluso sin la garantía MERCANTIL implícita o sin garantizar la CONVENIENCIA PARA UN PROPÓSITO PARTICULAR. El autor no asume ninguna responsabilidad si el lector hace un mal uso de la misma.

Estos apuntes se basan en: - La documentación oficial de Python, https://docs.python.org/es/3/tutorial/index.html, - La bibliografía presentada al final de este documento, y - Documentación propia recogida a lo largo de los años de diversas fuentes.

1.2 Objetivo del curso

- Objetivo general del curso:
- Aprender a usar Python para crear scripts y programas simples plenamente funcionales, desarrollar aplicaciones web y realizar análisis de datos.

- Objetivos específicos:

- Reconocer las características principales de Python y su utilidad práctica.
- Identificar tipos de datos (simples y compuestos) y operadores.
- Aplicar variables y estructuras de control de flujo.
- Construir funciones y clases (POO).
- Clasificar los módulos y paquetes por sus funcionalidades y objetivos.
- Comparar el lenguaje con otros similares de scripts, procedimentales y orientados a objetos.
- Probar bibliotecas para conexiones REST a aplicaciones web y bases de datos.
- Resolver problemas y errores en el código fuente proponiendo soluciones alternativas (refactoring).
- Utilizar la biblioteca pandas junto con matplotlib y numpy para realizar análisis estadísticos de datos y gráficos.

- Como resultado práctico al final del curso cada estudiante habrá creado una aplicación web utilizando el microframework de Python Flask.

1.3 Temario

- Introducción, instalación y compilación
- Datos, expresiones y sentencias
- Variables y funciones, control de flujo
- Clases y objetos, herencia, polimorfismo
- Entradas y salidas con Python
- Gestión de módulos, paquetes y bibliotecas
- Servicios y programas en red, REST API
- Desarrollo de aplicaciones web con Flask
- Análisis de datos con pandas, matplotlib y numpy

- Módulos opcionales
  - SQL ejemplos con pandas
  - Plotting con Python
  - Machine Learning con Python
  - DevOps con Ansible
  - SDK de GCP para Python
  - Kubernetes con Python y Docker

1.4 Bibliografía

- Downey, A., Elkner, J., Meyers, C. Aprenda a Pensar Como un Programador con Python. (2015).
  Recuperado de https://argentinaenpython.com/quiero-aprender-python/aprenda-a-pensar-como-un-programador-con-python.pdf
- Kent D. Lee. Python, Programming Fundamentals Second Edition. 2014.
- Marzal Varó, A., Gracia Luengo, I., García Sevilla, Pedro. Introducción a la programación con Python 3. (2014).
  Recuperado de http://repositori.uji.es/xmlui/handle/10234/102653
- Miller, B., Ranum, D. Solución de problemas con algoritmos y estructuras de datos usando Python. Traducido por Mauricio Orozco-Alzate, Universidad Nacional de Colombia - Sede Manizales.
  Recuperado de https://runestone.academy/ns/books/published/pythoned/index.html?mode=browsing
- Shaw, Z. A., Learn Python 3 the Hard Way. (2016).
  Recuperado de https://learnpythonthehardway.org/
- Van Rossum, G. and the Python development team. Documentación de Python en español. (2020).
  Recuperado de https://python-docs-es.readthedocs.io/es/3.10/

1.5 Entradas y salidas con Python

- Expresiones regulares, Recursos, Globbing, Codificación de caracteres, Operaciones sobre ficheros, Leer un fichero, Escribir un fichero, Rutas a ficheros, Copiar, mover y renombrar ficheros, Importar y exportar CSV/TSV, Exportar XLSX (Excel), Comparar ficheros, Hashs (MD5, Familia SHA).

1.6 Expresiones regulares

- Python cuenta con una biblioteca ya incluida que permite trabajar con expresiones regulares: *re*.
- Las expresiones regulares son herramientas para expresar patrones en el texto de dos formas: la básica y la extendida.
- La forma básica (*Basic Regular Syntax - BRE*) utiliza los corchetes `[]`, el punto `.`, el quilate `^`, el asterisco '* `el símbolo de peso o dólar ($`') y los caracteres de las letras del abecedario.
- La forma extendida (*Extended regular expressions - ERE*) utiliza el símbolo más `+`, la barra vertical `|`, el símbolo de interrogación `?`, las llaves `{}`y los paréntesis `()`.
- En los sistemas operativos Unix/Linux se utilizan los programas *grep* y *sed* para buscar texto dentro de los ficheros y editarlos desde la línea de comandos respectivamente.
- Las expresiones entre corchetes (`[]`) coinciden con cualquier carácter dentro de los corchetes.
- Por ejemplo, la expresión regular `c[aeo]sa` coincide con las palabras casa, cesa, cosa.

- Las expresiones de rango enumeran los puntos inicial y final separados por un guion (`-`), como en `a[2-4]b`.
- Esta expresión regular coincide con a2b, a3b y a4b.

- Cualquier carácter individual: el punto (`.`) representa cualquier carácter individual excepto una nueva línea.
- Por ejemplo, `a.z` coincide con a2z, abz, aQz o cualquier otra cadena de tres caracteres que comience con a y termine con z.

- Inicio y final de línea: el quilate (`^`) representa el comienzo de una línea y el signo de dólar (`$`) el final.

- Operadores de repetición: un asterisco (`*`) es cero o más ocurrencias, un signo más (`+`) coincide con una o más ocurrencias y un signo de interrogación (`?`) Especifica cero o una coincidencia.

- Varias cadenas posibles: la barra vertical (`|`) separa dos posibles coincidencias: `coche | furgoneta` coincide con coche o furgoneta.

- Los paréntesis (`()`) rodean las subexpresiones y se utilizan a menudo para especificar cómo se aplicarán los operadores.

- Escapar: si se desea hacer coincidir uno de los caracteres especiales, como un punto, debe precederle con una barra invertida (\).
- Por ejemplo, para hacer coincidir el nombre de host de una maquina (webmail.ejemplo.com), la sintaxis sería la siguiente: `webmail\.ejemplo\.com`.

[ ]:
```python
# Ejemplo de uso de la biblioteca re
import re

# Encontrar coincidencias
texto = "Hoy es un día magnífico y maravilloso"
exp_reg = re.search("^Hoy.*oso$", texto)
print(exp_reg)
```

[ ]:
```python
# Encuentra todas las coincidencias
exp_reg2 = re.findall("ma", texto)
print(exp_reg2)
```

[ ]:
```python
# Reemplazar espacio vacío
exp_reg3 = re.sub("\\s", "    ", texto)
print(exp_reg3)
```

[ ]:
```python
# Obtener todo el texto
exp_reg4 = re.match(r'.*', texto)
print(exp_reg4.group(0))
```

- Para verificar una expresión desde la línea de comando:

[ ]:
```
python3 -W error -c '"\s"'
```

1.6.1 Recursos

- La documentación de Python: https://docs.python.org/es/3/library/re.html#regular-expression-syntax.
- La entrada de la Wikipedia: https://es.wikipedia.org/wiki/Expresi%C3%B3n_regular.
- Un manual https://desarrolloweb.com/manuales/expresiones-regulares.html.
- Aplicaciones web para probar expresiones regulares: https://regexr.com/ y https://regex101.com/.

1.7 Globbing

- El módulo *glob* encuentra los nombres de rutas siguiendo el patrón especificado para las terminales en UNIX.
- Se pueden usar comodines con muchos comandos (se denomina *globbing*) y hay tres tipos que son muy comunes en GNU/Linux:

- Símbolo de interrogación (`?`): un único carácter,
- Un asterisco (`*`): coincide con cualquier carácter o conjunto de caracteres, incluido ningún carácter,
- Valores entre corchetes (`[]`): coinciden con cualquier carácter del conjunto.

- Estos comodines se pueden utilizar con el módulo anteriormente mencionado.
- La documentación oficial se encuentra en: https://docs.python.org/es/3/library/glob.html.

[ ]:
```
>>> import glob
>>> glob.glob('*')
```

[ ]:
```
import glob
for fichero in glob.glob('directorio/*'):
    print(fichero)
```

- Para buscar todos los ficheros con extensión *.py*:

[ ]:
```
ficheros_py = glob.iglob("*.py")
print("Lista de ficheros .py: ")
for ficheros in ficheros_py:
    print(ficheros)
```

- Para buscar coincidencias con patrones específicos en Python se pueden utilizar el módulo *fnmatch* y las funciones *os.scandir()*, os.path.expandvars()* y *os.path.expanduser()*.
- *Fnmatch* busca las coincidencias de nombres de ficheros utilizando los comodines de la shell (`*`, `?`, `[secuencia]`, `[!secuencia]`) y devuelve verdadero o falso.
- Las funciones *os.path.expandvars()* y *os.path.expanduser()* se utilizan para la expansión de variables en la *shell* de los nombres de ficheros.
- La documentación: https://docs.python.org/es/3/library/fnmatch.html#module-fnmatch.

[ ]:
```
import fnmatch
import os
```

```
# Busca todos los ficheros con el patrón especificado
patron = 'clase*.py'
print('Patrón :', patron)
print('*****')

ficheros = os.listdir('./')
for nombre in ficheros:
    print('Nombre: %-25s %s' % (nombre, fnmatch.fnmatch(nombre, patron)))
```

- *fnmatch* utilizando filtros:

[ ]:
```
import fnmatch
import os


patron = 'clase*.py'
ficheros = os.listdir('./')

print('Nombre: ', ficheros)
print ('Coinciden: ', fnmatch.filter(ficheros, patron))
```

1.8 Codificación de caracteres

- La codificación en la cual vienen los ficheros para ser manipulados no es trivial.
- En computación, la codificación de caracteres es usada para representarlos asignándoles un número a cada uno para su representación digital.

"Los más populares tipos de codificación de caracteres son ASCII y Unicode. Mientras que ASCII es todavía soportado por casi todos los editores de texto, Unicode es más utilizado porque soporta un mayor número de caracteres. Unicode es generalmente definido como UTF-8, UTF-16, o UTF-32, que referencian a diversos estándares Unicode. UTF significa formato de transformación Unicode (Unicode Transformation Format) y los números indican la cantidad de bits usados para representar cada carácter."

- Fuente: Christensson, P. (2010, September 24). Character Encoding Definition. Recuperado el 29 agosto 2021 de https://techterms.com.

- Hay extensa documentación en Internet, entre la cual se citan:
  - El libro Programming with Unicode https://unicodebook.readthedocs.io/nightmare.html,
  - las páginas de docs.python.org https://docs.python.org/es/3/library/codecs.html#standard-encodings,
  - la pregunta hecha en Stackoverflow sobre el tema: https://stackoverflow.com/questions/18171739/unicodedecodeerror-when-reading-csv-file-in-pandas-with-python.
  - Allí mencionan, entre otras cosas, el famoso artículo de Joel Spolsky:

  - https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-develop

- En los sistemas operativos GNU/Linux se puede verificar el tipo de codificación ejecutando el comando *file*:

```
[ ]: file -bi nombre_del_fichero.txt
```

- Luego para cambiar el formato al fichero se utiliza *iconv*:

```
[ ]: iconv -f iso-8859-1 -t utf-8 -o fichero_nuevo_utf8.txt fichero_antiguo_8859-1.
     ↪txt
```

- En los sistemas operativos Windows se puede abrir el fichero con la aplicación notepad y luego al *guardar como* se puede escoger la codificación deseada.
- Si está instalado *git* desde la consola *cmd* se puede utilizar el comando *file* como en GUN/Linux o Unix.
- Fuente: https://stackoverflow.com/questions/3710374/get-encoding-of-a-file-in-windows.

- Por último, las bibliotecas de Python que permiten detectar la codificación de un fichero son:
  - *Chardet* https://pypi.org/project/chardet/
  - *cChardet* https://pypi.org/project/cchardet/
  - *charset_normalizer* https://pypi.org/project/charset-normalizer/
- Esta última es la más actual.

```
[ ]: # Para instalarlas
     pip install chardet
     pip install cchardet
     pip install charset-normalizer -U
```

```
[ ]: # Ejemplo utilizando la biblioteca cChardet
     # Adaptado de https://pypi.org/project/cchardet/

     >>> import cchardet as chardet
     >>> with open(r'catalogo_cf.csv', 'rb') as f:
     ...     msg = f.read()
     ...     result = chardet.detect(msg)
     ...     print(result)
     ...
     {'encoding': 'ISO-8859-1', 'confidence': 0.8832640051841736}
```

- Una biblioteca útil es *ftfy* https://pypi.org/project/ftfy/ que repara código *unicode*.
- La documentación se encuentra en https://ftfy.readthedocs.io/en/latest/.

```
[ ]: # Para instalarla
     pip install ftfy
```

```
[ ]: # Ejemplo de uso
     >>> import ftfy
     >>> ftfy.fix_text('âœ" No problems')
     '  No problems'
```

1.9 Operaciones sobre ficheros

- Una de las tareas más frecuentes que realizaremos en Python será trabajar con ficheros: abrirlos, editarlos, eliminarlos, etc.
- La manera estándar de abrir un fichero es usando la opción *open()* del módulo *io*.

- Descargar el fichero del libro "Don Quijote" del autor Miguel de Cervantes Saavedra del sitio web Project Gutemberg desde https://www.gutenberg.org/files/2000/2000-0.txt.
- Otra opción es copiar el texto desde https://www.gutenberg.org/files/2000/2000-h/2000-h.htm en un fichero con formato de texto.

[ ]:
```
fichero = open("2000-0.txt", 'r')
for linea in fichero:
    print(linea)
fichero.close()
```

Modos que se pueden usar con *open()*:

- r → Abre para lectura (predeterminado)
- w → Abre o crea el fichero para escritura, borrando (truncating) primero su contenido
- a → Abre o crea el fichero para escritura, pero anexa (appends) al final
- x → Crea y abre un nuevo fichero para escritura pero no abre ficheros existentes
- + → Abre el fichero para lectura y escritura
- t → Trabaja con el fichero en modo texto (predeterminado)
- b → Trabaja con el fichero en modo binario

1.10 Leer un fichero

- Para leer un fichero se pueden usar las funciones:
  - *read()* que lee el contenido como una única cadena (se puede especificar el número de caracteres),
  - *readline()* que se detiene cuando encuentra un final de párrafo, y
  - *readlines()* que devuelve el fichero como una lista de strings.

- Algunos ejemplos:

[ ]:
```
# Lee los primeros 200 caracteres de un fichero
with open("2000-0.txt", 'r') as fichero:
    contenido = fichero.read(200)
    print(contenido)
```

[ ]:
```
# Lee la primera línea de un fichero
with open("2000-0.txt", 'r') as file:
    contenido = file.readline()
    print(contenido)
```

[ ]:
```
# Lee los párrafos de un fichero.
# Strip() evita una línea en blanco entre ellos
with open("2000-0.txt", 'r') as file:
    contenidos = file.readlines(2000)
```

```
    for c in contenidos:
        print(c.strip())
```

1.11 Escribir un fichero

- Para escribir en un nuevo fichero utilizamos la función *write()*.
- En los siguientes ejemplos, primero se crea el fichero y se escriben los párrafos de la variable entrada con *w (write)*, para luego agregar el texto de la segunda variable con *a (append)*:

[ ]:
```
entrada = """Primera parte del ingenioso hidalgo don Quijote de la Mancha


"""
# Creamos un fichero y pegamos el texto de la variable entrada
with open("quijote-ext01.txt", 'x') as file:
    file.write(entrada)
```

[ ]:
```
# En este caso vamos a agregar el texto de la variable "entrada_agregar" al␣
 ↪final del fichero

entrada_agregar = """ En un lugar de la Mancha, de cuyo nombre no quiero
acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de
algo más vaca que carnero, ...
"""
# Abrimos el fichero y adjuntamos el texto de la variable "entrada_agregar"
with open("quijote-ext01.txt", 'a') as file:
    file.write(entrada_agregar)
```

- La función *writelines()* escribe en cambio una lista de cadenas de texto (strings).
- Por otro lado, para buscar texto en un fichero se utiliza la función *seek()*:

[ ]:
```
# Busca a partir del carácter 18 e imprime los 42 caracteres siguientes
with open("quijote-ext01.txt", 'r') as file:
    file.seek(18)
    contenido = file.read(42)
    print(contenido)
```

1.12 Rutas a ficheros

- En cada sistema operativo (GNU/Linux, Windows®, MacOS, etc.) las rutas a ficheros y directorios se gestionan de diferente manera.
- Python utiliza dos bibliotecas para poder trabajar con ellas: *os* y *pathlib*, siendo que en la mayoría de los casos se utiliza *pathlib.path*.
- *Pathlib*, o rutas de sistemas orientada a objetos, crea una instancia de ruta para la plataforma en la que se ejecuta el código.
- La información completa se encuentra en la documentación oficial de Python: https://docs.python.org/3/library/pathlib.html.

[27]:
```python
# Un ejemplo creando un directorio en la carpeta inicio del usuario
import os
from pathlib import Path


# Se obtiene el directorio inicio del usuario en SO Linux y Windows
fichero_path = Path(Path.home(), "directorio_ficheros")

# Si no existe el directorio, se crea
if not fichero_path.exists():
    os.makedirs(fichero_path)

# Se agrego el fichero al path
fichero_path = fichero_path.joinpath("quijote-ext02.txt")

# se escriben las siguientes líneas en el fichero
with fichero_path.open('w') as file:
    lineas = [
        "Primera parte del ingenioso hidalgo don Quijote de la Mancha \n"
        "Capítulo primero. Que trata de la condición y ejercicio del famoso \n"
        "hidalgo don Quijote \n"
    ]
    file.writelines(lineas)
```

- Para mostrar los ficheros de la ruta actual:

[ ]:
```python
import os

ruta = '.'
print(os.listdir(ruta))
```

- Otros ejemplos leyendo de un directorio en la actual ruta, y de un directorio superior:

[ ]:
```python
# Lee del directorio "directorio1"
fichero_path = Path("directorio1", "quijote-ext02.txt")
with fichero_path.open('r') as file:
print(file.read())
```

[ ]:
```python
# Lee del directorio superior
fichero_path = Path(os.pardir, "./", "quijote-ext01.txt")
with fichero_path.open('r') as file:
print(file.read())
```

1.13 Copiar, mover y renombrar ficheros

- En Python es bastante sencillo copiar, mover, y renombrar ficheros si utilizamos las bibliotecas *os* y *shutils*.
- *copy()* copia los datos del archivo y el modo de permiso del archivo (véase *os.chmod()*).

- Otros metadatos, como los tiempos de creación y modificación de archivos, no se preservan.
- Para preservar todos los metadatos del archivo, el método *copy2()* es utilizado en su lugar.
- Extraído de: https://docs.python.org/es/3/library/shutil.html#shutil.copy

[ ]:
```
import shutil


# Copiar ficheros situados en el mismo directorio
shutil.copy(src="quijote-ext01.txt", dst="quijote-ext03.txt")

# Copiar ficheros preservando los metadatos
shutil.copy2(src="quijote-ext01.txt", dst="quijote-ext04.txt")
```

[ ]:
```
# Mostrar metadatos en GNU/Linux
stat quijote-ext01.txt
stat quijote-ext03.txt
stat quijote-ext04.txt
```

[ ]:
```
import shutil


# Copiar al igual que el comando cp -R en GNU/Linux
shutil.copytree(src="directorio1/", dst="directorio3/")

# Ignorar ficheros .jpg al copiar
shutil.copytree(src="directorio1", dst="directorio4",
                ignore=shutil.ignore_patterns('*.jpg'))
```

[ ]:
```
# Mover (o cambiar de nombre al fichero) al igual que el comando mv en GNU/Linux
shutil.move(src="directorio1/imagen.jpg", dst="directorio4/imagen.jpg")
```

[ ]:
```
# Mover un fichero usando la biblioteca os
os.rename("directorio4/imagen.jpg", "directorio1/imagen.jpg")
```

[ ]:
```
# Borrar un fichero
os.remove("directorio1/imagen.jpg")
```

1.14 Importar y exportar CSV/TSV

- Para trabajar con ficheros con formato .csv (separa los datos con una coma) y .tsv (separa los datos con un tabulador) se utiliza la biblioteca *pandas*.
- *Pandas* es una biblioteca de alto desempeño y fácil de usar que incluye herramientas para manipular estructuras de datos y hacer análisis en ellos.
- Tiene una extensa documentación en su sitio web, de la cual para principiantes se puede mencionar:

- La guía oficial: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html.

- 10 minutes to Pandas: https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html.
- Cookbook: https://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html.
- Pandas Cheat Sheet: https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf.
- El libro *Pandas for Data Analysis* https://pandas.pydata.org/pandas-docs/version/1.4.4/pandas.pdf.

- Para instalar Pandas:

[ ]:
```
pip install pandas
```

[ ]:
```
# Ejemplo de uso de la biblioteca Pandas
import pandas as pd

# Variables con los ficheros a importar
fichero_csv = "catalogo_cf.csv"
fichero_tsv = "catalogo_cf.tsv"

# Nombres de los ficheros a escribir
escribir_csv = 'catalogo_csv_ext.csv'
escribir_tsv = 'catalogo_tsv_ext.tsv'

# Lee los datos de los ficheros
# Para la codificación estándar europea también se puede utilizar latin_1
leer_csv = pd.read_csv(fichero_csv, encoding="ISO-8859-1")
leer_tsv = pd.read_csv(fichero_tsv, sep='\t', encoding="utf-8-sig")

# Imprime los primeros 10 registros
print(leer_csv.head(10))
print(leer_tsv.head(10))

# Escribe en los ficheros (en el .csv solamente los 10 primeros registros)
with open(escribir_csv, 'w', encoding='utf-8') as write_csv:
    # index=False no muestra el índice
    write_csv.write(leer_csv.head(10).to_csv(sep=',', index=False))

with open(escribir_tsv, 'w', encoding='utf-8-sig') as write_tsv:
    write_tsv.write(leer_tsv.to_csv(encoding='utf-8',
                                    sep='\t', index=False))
```

- UTF-8 BOM es una secuencia de bytes al inicio de un flujo de texto (`0xEF, 0xBB, 0xBF`) que permite al usuario confirmar si un fichero tiene codificación UTF-8.
- Fuente: https://en.wikipedia.org/wiki/Byte_order_mark.

[ ]:
```
# Eliminar BOM de UTF-8
file catalogo_cf.tsv
vi -c ":set nobomb" -c ":wq" catalogo_cf.tsv
```

- Fuente: https://stackoverflow.com/questions/2223882/whats-the-difference-between-utf-8-and-utf-8-without-bom.

1.15 Exportar XLSX (Excel)

- Para trabajar con ficheros con los formatos de Microsoft Excel *xlsx/xlsm/xltx/xltm* se pueden utilizar las biblioteca *Pandas* y *openpyxl*.

[ ]:
```
pip install pandas
pip install openpyxl
```

- Convertimos el fichero con formato .csv a .xlsx:

[ ]:
```
import pandas as pd

leer_fichero = pd.read_csv(r'catalogo_csv_ext.csv')
leer_fichero.to_excel(r'catalogo_cf.xlsx', index=None, header=True)
```

- Exportamos una selección de datos del fichero con formato .xlsx:

[ ]:
```
# Ejemplo exportar fichero con formato .xlsx
import pandas as pd


# Se crea el dataframe con pandas de un fichero Excel
df = pd.read_excel("catalogo_cf.xlsx")
print(df)

# Selecciona la primera y la cuarta columna
print(df.iloc[:, [0, 3]])

# se exporta a excel
df.iloc[:, [0, 3]].to_excel('catalogo_cf_resumen.xlsx')
```

- El tutorial oficial de la biblioteca *openpyxl* con ejemplos:
- https://openpyxl.readthedocs.io/en/stable/tutorial.html

1.16 Comparar ficheros y directorios

- Para comparar ficheros y directorios se utiliza el módulo *filecmp*:

[ ]:
```
import filecmp

# Directorios a comparar
dir1 = "directorio1"
dir2 = "directorio3"
```

[ ]:
```
# Ficheros comunes dentro de los directorios a comparar
comunes = ["quijote-ext01.txt", "quijote-ext02.txt", "quijote-ext03.txt"]
```

[ ]:
```
# Comparar los metadatos de los ficheros comunes
match, mismatch, errors = filecmp.cmpfiles(dir1, dir3, comunes)

print("Verifica los metadatos:")
print("Coinciden:", match)
print("NO coinciden:", mismatch)
print("Errores:", errors, "\n")
```

[ ]:
```
# Compara también los datos dentro de los ficheros
match, mismatch, errors = filecmp.cmpfiles(dir1, dir3, comunes, shallow=False)
```

[ ]:
```
print("Deep comparison:")
print("Coinciden:", match)
print("NO coinciden:", mismatch)
print("Errores :", errors)
```

- Además del módulo *filecmp* se puede utilizar *difflib* para comparar diferencias de archivos.
- La documentación: https://docs.python.org/es/3/library/difflib.html#module-difflib.

1.17 Hashs (MD5, Familia SHA)

- Actualmente las aplicaciones que necesitan guardar información sensible sobre el usuario, como por ejemplo el nombre y una clave, lo hacen a través del uso de algoritmos que cifran la información y almacenan el resultado en un *hash*.
- Por otro lado, crear el hash de un fichero nos permite generar una secuencia que sirve para identificarlo o para compararlo con otro fichero.

- En Python la biblioteca utilizada para este fin se denomina *hashlib*.
- Se pueden utilizar distintas versiones de algoritmos pero cabe aclarar que MD5 y SHA-1 ya no son seguros.
- En la actualidad se está utilizando, entre otros, el algoritmo SHA-2 en sus versiones SHA-256 y SHA-512.

[ ]:
```
import hashlib
# Se crea una variable con el fichero a ser usado para el hash
nombre = 'directorio1/quijote-ext01.txt'

# Se convierte el formato a bytes usando 'encode'
# Las funciones hash solamente aceptan esta codificación
nombre_codificado = nombre.encode()

# Se le pasa nombre_codificado a la función sha512
nombre_hash = hashlib.sha512(nombre_codificado)

# Ya creado el hash, se imprime la versión
# hexadecimal del mismo usando el método 'hexdigest()'
print("Object:", nombre_hash)
```

```
print("Hexadecimal format:", nombre_hash.hexdigest())
```

- La documentación: https://docs.python.org/es/3/library/hashlib.html.
- Otras bibliotecas que se pueden instalar:
- https://pypi.org/project/oscrypto/
- https://cryptography.io/en/latest/