MÓDULO: HERRAMIENTAS BIG DATA

Tema: Herramientas de análisis: Programación en Python

Ferran Carrascosa Mallafrè

Licenciado en Matemáticas por la Universidad de Barcelona. Data Scientist

Índice

[Objetivos específicos 3](#__RefHeading___Toc2630_3123840034)

[2. Programación en Python 3](#__RefHeading___Toc2632_3123840034)

[2.1. Introducción 3](#__RefHeading___Toc2634_3123840034)

[2.2. Actividad guiada 2 4](#__RefHeading___Toc2636_3123840034)

[2.3. Elementos básicos de Python 6](#__RefHeading___Toc2638_3123840034)

[2.4. Gráficos con Matplotlib 12](#__RefHeading___Toc2640_3123840034)

[2.5. Colecciones de objetos 14](#__RefHeading___Toc2642_3123840034)

[2.5.1. Definición de los objetos 14](#__RefHeading___Toc2644_3123840034)

[2.6. Numpy 14](#__RefHeading___Toc2646_3123840034)

[2.7. Pandas 15](#__RefHeading___Toc2648_3123840034)

[2.8. Control de flujo 16](#__RefHeading___Toc2650_3123840034)

[2.8.1. Funciones 17](#__RefHeading___Toc2652_3123840034)

[2.8.2. Condicionales 17](#__RefHeading___Toc2654_3123840034)

[2.8.3. Bucles 18](#__RefHeading___Toc2656_3123840034)

[2.9. Gestión de datos 20](#__RefHeading___Toc2658_3123840034)

[Ideas clave 21](#__RefHeading___Toc2660_3123840034)

[Anexo: Readme de Python 22](#__RefHeading___Toc2662_3123840034)

[Bibliografía 25](#__RefHeading___Toc2664_3123840034)

[Recursos en internet 25](#__RefHeading___Toc2666_3123840034)

# Objetivos específicos

* Realizar operaciones de lectura y escritura de datos con Python.
* Saber escoger la estructura de datos de Python adecuada para cada problema.
* Tener las bases para realizar análisis descriptivo mediante tablas y gráficos en Python.
* Desarrollar pequeñas piezas de código en Python.

# 2. Programación en Python

## 2.1. Introducción

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py.ipynb)

Python es una herramienta para la programación de propósito general.

En los últimos años, se ha convertido en uno de los lenguajes de referencia para el *Data Science*. El motivo es que ha sabido rodearse de un gran ecosistema como [SciPy](https://www.scipy.org/) con librerías (numpy, pandas, scipy…) y herramientas (Jupyter, Spyder,…) orientadas al análisis y a la programación matemática.

Los desarrolladores de Python buscan hacer un lenguaje vivo y atractivo para el programador. Prueba de ello es que su nombre es un tributo a la compañia de humor británica *Monthy Python*.

El núcleo de su filosofía de programación se resume en el “*Zen de Python*”, formado por 20 aforismos escritos por Tim Peters, 19 de los cuales se pueden leer más abajo importando this. El veinte, dijo Tim Peters que lo diría el creador de Python, Guido van Rossum, pero parece que aún no se ha pronunciado…

import this

## The Zen of Python, by Tim Peters  
##   
## Beautiful is better than ugly.  
## Explicit is better than implicit.  
## Simple is better than complex.  
## Complex is better than complicated.  
## Flat is better than nested.  
## Sparse is better than dense.  
## Readability counts.  
## Special cases aren't special enough to break the rules.  
## Although practicality beats purity.  
## Errors should never pass silently.  
## Unless explicitly silenced.  
## In the face of ambiguity, refuse the temptation to guess.  
## There should be one-- and preferably only one --obvious way to do it.  
## Although that way may not be obvious at first unless you're Dutch.  
## Now is better than never.  
## Although never is often better than \*right\* now.  
## If the implementation is hard to explain, it's a bad idea.  
## If the implementation is easy to explain, it may be a good idea.  
## Namespaces are one honking great idea -- let's do more of those!

Ser *Pythoniano* o *Pythónico* significa utilizar correctamente el código, es decir, programar con un lenguaje simple y fácil de leer.

## 2.2. Actividad guiada 2

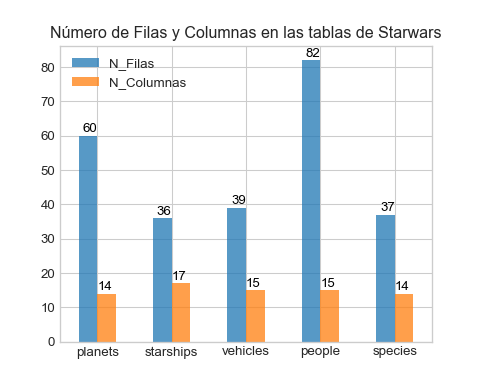
La actividad guiada, que te proponemos, en consonancia con los Pythonianos, es que te diviertas analizando datos relacionados con la saga de Star Wars: Planetas, Naves, Vehículos, Personajes y Especies.

Para ello, contamos con los datos de [SWAPI](https://swapi.dev/), acrónimo de STAR WARS API, que nos da acceso libre a una colección de datos de la saga.

Estos datos se han descargado y preparado expresamente para este curso. Puedes encontrar el código utilizado en el [Anexo: README de Python](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_60_anexo.ipynb), capítulo “IMPORTAR DATOS DE STARWARS SWAPI”.

Para cargar los datos y visualizar su contenido.

import pandas as pd  
import matplotlib.pyplot as plt  
#import seaborn as sns  
plt.style.use('seaborn-whitegrid')  
  
entidades = ['planets','starships','vehicles','people','species']  
entidades\_df = {x: pd.read\_pickle('www/' + x + '\_df.pkl') for x in entidades}  
entidades\_count = {k: v.shape for k,v in entidades\_df.items()}  
entidades\_count\_df = pd.DataFrame.from\_dict(entidades\_count, orient='index', columns=['N\_Filas', "N\_Columnas"])  
  
g = entidades\_count\_df.plot.bar(alpha=0.75, rot=0)  
plt.title("Número de Filas y Columnas en las tablas de Starwars")  
#g.title = 'Entidad de Starwars'  
for i, (k, row) in enumerate(entidades\_count\_df.iterrows()):  
 g.text(i-0.1, row['N\_Filas']+1, row['N\_Filas'], color='black', ha="center")  
 g.text(i+0.1, row['N\_Columnas']+1, row['N\_Columnas'], color='black', ha="center")  
plt.show()



Vemos que los datos están formados por 5 conjuntos. Por ejemplo, people, contienen 82 personajes descritos mediante 16 variables.

Una muestra de los datos de los personajes.

entidades\_df['people'].head()

## height ... url  
## name ...   
## Luke Skywalker 172.0 ... http://swapi.dev/api/people/1/  
## C-3PO 167.0 ... http://swapi.dev/api/people/2/  
## R2-D2 96.0 ... http://swapi.dev/api/people/3/  
## Darth Vader 202.0 ... http://swapi.dev/api/people/4/  
## Leia Organa 150.0 ... http://swapi.dev/api/people/5/  
##   
## [5 rows x 15 columns]

Observa que hay columnas numéricas como la altura, peso, edad (en años ABY Antes de la Batalla de Yavin). Otras categóricas, como el género del personaje. Incluso columnas en formato de lista, como las películas en las que salió el personaje y los vehículos y naves que condujo…

"Yoda habló de otra."   
"La otra de quién habló es tu hermana gemela.»   
― Luke Skywalker y Obi-Wan Kenobi

Te acordabas que Luke y Leia eran gemelos? ¿Sabías que su padre (Anakin, posteriormente Darth Vader tenía 22 años cuando los tuvo?

Podrás analizar todo esto y mucho más, en los datos y así convertirte en el auténtico Jedi que sabemos que llevas dentro.

Consulta la [Documentación de SWAPI](https://swapi.dev/documentation) sobre sus tablas y campos.

*Y que la fuerza te acompañe…*

## 2.3. Elementos básicos de Python

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_10_elem_bas.ipynb)

Aunque se presuponen unos conocimientos iniciales de Python, a continuación, se hace un repaso de Python, Anaconda y notebooks de Jupyter-Colab. El objetivo es reforzar aquellos elementos más orientados al análisis de datos.

#### INTERFAZ DE USUARIO

Python se puede ejecutar de forma interactiva des de consola de texto. Para abrir la consola, ejecutar el comando Python desde un terminal Windows o Linux. El temario utiliza notebooks ejecutados des de Jupyter y/o Colab.

Prueba a ejecutar la siguiente línea directamente en este notebook.

6 / (4 - 1.5)

## 2.4

**Recuerda**: Para editar una celda del notebook, aprieta Enter. Para ejecutarla, Ctrl+Enter (sólo ejecuta), o bien, Mayús+Enter (ejecuta y avanza celda).

#### FICHEROS \*.py

Una forma habitual de programar en Python es mediante ficheros de texto con extensión .py. Existen múltiples entornos para estos ficheros: Spyder (parecido a RStudio), PyCharm, Visual Studio Code o cualquier editor de texto como Notepad++.

Este tipo de ficheros se utilizan habitualmente para generar librerías de funciones y ejecutar desarrollo productivo de aplicaciones en Python.

#### NOTEBOOKS JUPYTER (\*.ipynb)

Los notebooks, como éste que estás leyendo, son la solución de la comunidad de SciPy para la programación interactiva. El proyecto nació con el nombre [IPython](https://ipython.org/) (o Python interactivo) y actualmente se orquestra en el paraguas de la plataforma [Jupyter](https://jupyter.org/). Jupyter permite ejecutar de forma interactiva docenas de lenguajes como Python, R…

Los notebooks, al igual que Rmardown, permiten combinar texto en formato Markdown (lenguaje de marcas ligero), con código con resultados en forma de texto o gráficos. Esta forma de funcionar resulta óptima para el análisis de datos y la formación.

Su uso se ha extendido de la mano del proyecto [Anaconda](https://www.anaconda.com/), la mayor plataforma de gestión de librerías open source enfocadas a la computación científica.

Para iniciar el entorno Jupyter sigue la guía del [Anexo: README de Python](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_60_anexo.ipynb) apartado “PREPARACIÓN ENTORNO LOCAL-JUPYTER (OPCIONAL)”.

##### COLABORATORY o COLAB

Actualmente la comunidad de Python cuenta con la plataforma [Colaboratory o Colab](https://colab.research.google.com/) que es un entorno libre para ejecutar notebooks Jupyter y almacenarlos en Google Drive. Este entorno sólo permite abrir ficheros .ipynb (no .py), pero facilita en gran medida su uso para desarrollar formación.

Puedes abrir los notebooks de este tema con Colab clicando en: [| Abre en Colab |](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_10_elem_bas.ipynb)

En la parte superior de los notebooks, encontrarás un script que inicializa el entorno Colab con las versiones de los paquetes de Python correctas y descarga el repositorio Github con el material del curso.

##### ACTIVIDAD GUIADA 2.1.

Crea tu primer notebook. En Jupyter/Colab, accede al menú File > New Notebook.

En Jupyter, selecciona el Kernel mbdds\_rpy20 que has creado como se indica en el [Anexo: README de Python](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_60_anexo.ipynb).

Inserta el siguiente código Python y ejecútalo:

print("Hello Wold!")

Guarda el documento con el nombre 21\_actividad.

##### EDITAR NOTEBOOKS JUPYTER

Un notebook está compuesto por un conjunto de casillas que pueden ser de distinto tipo:

* *Code*: Código Python que se puede ejecutar interactivamente y puede tener un salida o output.
* *Markdown*: Inserta texto con este formato. No tiene output.

Una casilla, puede tener dos modos:

* Edición: Aprieta Enter o doble clic. Puedes modificar su contenido Python o Markdown
* Visión: Tecla Esc o clic barra izquierda (en Jupyter). Selecciona una o varias casillas con el ratón o con las teclas Mayús + Cursos.

Para mostrar una casilla Markdown en formato impresión (sin marcas), ejecútala del mismo modo que si fuera código: - Sólo ejecutar: Ctrl+Enter. - Ejecutar y avanzar casilla: Botón Run, o bien, Mayús+Enter.

###### ATAJOS DE TECLADO

Puedes gestionar las celdas des del teclado en el modo de Visión (**teclaEsc**):

* Cambiar tipo celda:
  + Modo Markdown: Tecla M, Ctr+M+M (Colab).
  + Modo Code: Tecla Y, Ctr+M+Y (Colab).
* Insertar celdas:
  + Antes: Tecla A, Ctr+M+A (Colab).
  + Después: Tecla B, Ctr+M+B (Colab).
* Copiar, pegar, desplazar arriba, abajo…

###### MARKDOWN

* **Parágrafo de texto**: Deja una línea en blanco (Enter) antes y después de un parágrafo de texto.
* **Encabezados**: Utiliza # seguido de espacio en blanco para los títulos de las secciones:

# TITULO NIVEL 1  
## TITULO NIVEL 2  
...

Deja también una línea en blanco (Enter) antes y después del título.

* **Emphasis**: Negrita con \*\*string\*\* . Cursiva con \*string\*.
* **Código látex**: $x^2$.
* **Fuente ancho fijo**: `texto` (usa tilde abierta o con orientación izquierda).
* **Bloque de código (sin que se ejecute)**: Tres abiertas, código (nueva línea), tres tildes abiertas. Ejemplo:

```

Soy codigo1  
Soy código2

```

* **Línea en blanco**: Usa una líena nueva con dos espacios o <br>.
* **Lista de puntos**: Al inicio de línea, usa guion - seguido de espacio y texto. Para una sublista de puntos pon tres espacios o tabulado y guion. Puedes usar también asterisco.
* **Lista numerada**: Al inicio, pon 1. seguido de espacio y texto. No es necesario cambiar el 1. en cada nueva línea.
* **Link externo**: [texto enlace](http://url)
* **Link interno**: Crea un ID con html <a id="seccion\_ID"></a> antes de la sección. Ahora, para enlazar con la sección, utiliza [titulo sección](#seccion\_ID).
* **Imagen**: Arrastrar y soltar en la celda markdown. También con ![texto alternativo](url imagen o ruta del a imagen), o también, <img src="url imagen o ruta del a imagen" alt="texto alternativo" title="texto descriptivo" /> .
* **Sangrado (Indented)**: Al incio de línea, usa símbolo > seguido de espacio y texto. Ejemplo:

Texto sangrado

* **Línea horizontal**: Tres asteriscos al inicio de línea, \*\*\* .
* **Colores**: Usa <font color=blue|red|green|pink|yellow>Texto</font> .

###### AYUDA DES DEL NOTEBOOK

En Python se pide ayuda de un comando con:

help(len)

## Help on built-in function len in module builtins:  
##   
## len(obj, /)  
## Return the number of items in a container.

En Jupyter, ejecuta ?comando para que se abra un menú contextual de ayuda.

?len

También con el tabulador tea ayuda a autocompletar código, ver opciones, parámetros…

###### COMANDOS MÁGICOS

Los comandos mágicos, aumentan las capacidades interactivas de los notebooks. Son de uso común:

Ejecutar script .py

%run holaMundo.py

## Hello World!

Historial de comandos ejecutados.

# vemos los primerso 3:  
%history 1-3

## if 'google.colab' in str(get\_ipython()):  
## !git clone https://github.com/griu/mbdds\_fc20.git /content/mbdds\_fc20  
## !git -C /content/mbdds\_fc20 pull  
## %cd /content/mbdds\_fc20/Python  
## !python -m pip install -r requirementsColab.txt  
6 / (4 - 1.5)  
## help(len)

Cambiar de directorio de trabajo.

%cd nueva\_ruta

Tiempo de ejecución de una línea.

%time L = [n \*\* 2 for n in range(1000)]

## CPU times: user 566 µs, sys: 43 µs, total: 609 µs  
## Wall time: 616 µs

De toda una celda

%%time  
L1 = [n \*\* 2 for n in range(1000)]  
L2 = [n \*\* 2 for n in range(1000)]  
L3 = [n \*\* 2 for n in range(1000)]

## CPU times: user 1.99 ms, sys: 0 ns, total: 1.99 ms  
## Wall time: 2 ms

###### COMANDOS SHELL

Los comandos de sistema se pueden ejecutar con !comando

!dir

## holaMundo.py modulo1\_tema4\_Py\_40\_contr\_flujo.ipynb  
## modulo1\_tema4\_Py\_00\_indice.ipynb modulo1\_tema4\_Py\_50\_gest\_dat.ipynb  
## modulo1\_tema4\_Py\_10\_elem\_bas.ipynb modulo1\_tema4\_Py\_60\_anexo.ipynb  
## modulo1\_tema4\_Py\_20\_matplotlib.ipynb modulo1\_tema4\_Py.ipynb  
## modulo1\_tema4\_Py\_30\_colec\_obj.ipynb README.md  
## modulo1\_tema4\_Py\_31\_numpy.ipynb requirementsColab.txt  
## modulo1\_tema4\_Py\_32\_pandas.ipynb www

#### FUNCIONES

Para definir la función valor absoluto:

def valor\_absoluto(num):  
 """devuelve el valor absoluto"""  
  
 if num >= 0:  
 return num  
 else:  
 return -num

* Se inicia la definción con def, seguido del nombre de la función
* A continuación los parámetros entre paréntesis y :
  + Admite definir valores por defecto
* El texto sirve como ayuda cuando se llama a ?valor\_absoluto
* El cuerpo del código con sangrado
* Puede devolver un objeto añadiendo return objecto
* Si se quiere devolver valores por pantalla hay que usar print

Para llamar a la función.

valor\_absoluto(-6)

## 6

#### PAQUETES

El indexador de paquetes de Python es **PyPI**. igual que **CRAN** en R. Actualmente indexa más de 235.000 librearías.

Para instalarlos hay distintos métodos. El más general es mediante pip desde Shell.

pip install nombre\_paquete

**Sabías que**: Anaconda tiene su propio gestor de paquetes llamado conda. Funciona de forma similar a pip. Además conda, aporta la capacidad de gestionar *Environments*, que a su vez, permiten fijar un conjunto de versiones de paquetes de Python. Los *Environments* facilitan reproducir en otras máquinas todo el entorno de ejecución y la replicabilidad de los desarrollos. Revisa un ejemplo en [Anexo: README de Python](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_60_anexo.ipynb).

Desde Jupyter/Colab también se pueden instalar paquetes con !pip y !conda (fíjate que ahora llevan ! delante.

!pip install nombre\_paquete

Una vez instalados, en cada nueva sesión de Jupyter, hay que importarlos para que sean accesibles.

import nombre\_paquete

## 2.4. Gráficos con Matplotlib

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_20_matplotlib.ipynb)

La librería gráfica [Matplotlib](https://matplotlib.org/), concebida por John Hunter en 2002, fué construida sobre objetos numpy (arrays N-dimensionales) y posteriormente adaptada a objetos Pandas (matrices con vectores datos de distintos tipos).

Actualmente, se apoya en otras librerías, más simples y de aspecto gráfico modernizado com Seaborn y los mismos Pandas.

Los objetos numpy y pandas se expondrán ampliamente en el siguiente apartado, no obstante, dado que el curso presupone ciertos conocimientos de programación, se muestran ahora los gráficos para poder utilizarlos en los siguientes apartados como herramienta.

Para importar la librería, habitualmente se importa el submódulo pyplot.

import matplotlib.pyplot as plt

##### ACTIVIDAD GUIADA 2.2

Consiste en conocer mejor a las especies mediante gráficos. En concreto se puede analizar su altura, años de vida, clase de especie (mamífero, reptil,…) y en cuantas películas ha salido esa especie.

«¡No puedes llevar a Su Alteza Real allí! Los Hutts son gangsters ...»

―Quarsh Panaka.

El primer paso consiste en cargar los datos de las especies y extraer las columnas.

import numpy as np  
import pandas as pd  
  
entidades = ['planets','starships','vehicles','people','species']  
entidades\_df = {x: pd.read\_pickle('www/' + x + '\_df.pkl') for x in entidades}  
entidades\_df['species']["num\_peliculas"] = entidades\_df['species'].films.apply(lambda x: len(x)) # numero de peliculas en las que aparece  
species\_df = entidades\_df['species'][["classification","average\_height","average\_lifespan","num\_peliculas"]].dropna()  
  
  
nombre = species\_df.index.values  
clasificacion = species\_df.classification.values  
altura\_media = species\_df.average\_height.values  
vida\_media = species\_df.average\_lifespan.values  
num\_peliculas = species\_df.num\_peliculas.values  
  
species\_df.head()

## classification average\_height average\_lifespan num\_peliculas  
## name   
## Human mammal 180.0 120.0 6  
## Wookie mammal 210.0 400.0 4  
## Hutt gastropod 300.0 1000.0 2  
## Yoda's species mammal 66.0 900.0 5  
## Toydarian mammal 120.0 91.0 2

Observa como cada fila representa una especie que viene caracterizada por 5 vectores numpy. Para saber más de los numpy ve al capítulo [Colecciones de objetos: numpy](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_31_numpy.ipynb):

* **nombre**: Nombre de la especie.
* **clasificacion**: Clasificación de la especie.
* **altura\_media**: Altura media (en cm).
* **vida\_media**: Vida media en años.
* **num\_peliculas**: Número de películas en las que aparece esa especie.

[**Abre 2.4. Gráficos con Matplotlib en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_20_matplotlib.ipynb)

## 2.5. Colecciones de objetos

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_30_colec_obj.ipynb)

### 2.5.1. Definición de los objetos

#### TIPO BÁSICOS

En Python, el tipo de dato se fija en la creación del objeto.

Los principales tipo básicos son:

|  |  |  |
| --- | --- | --- |
| Ejemplo | Constructores | Tipo de dato |
| x = “Hola Mundo” | x = str(“Hello World”) | str |
| x = 20 | x = int(20) | int |
| x = 20.5 | x = float(20.5) | float |
| x = True | x = bool(5) | bool |

Para saber el tipo de un objeto, utiliza type().

**Recuerda**: Para saber el tipo del objeto utiliza type().

a = 3 > 4  
print("El tipo de 'a' es:",type(a),"y vale",a)

## El tipo de 'a' es: <class 'bool'> y vale False

[**Abre 2.5. Colecciones de objetos en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_30_colec_obj.ipynb)

## 2.6. Numpy

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_31_numpy.ipynb)

El package [numpy](https://numpy.org/) es la solución más popular dentro de Python para realizar computación científica.

Recoge las mejores prácticas introducidas en las Listas y organizadas para realizar cálculos de forma eficiente. Se estructuran como vectores o arrays de N, dimensiones de un mismo tipo de dato.

El primer paso, consiste en cargar la librería, habitualmente con *namespace* np.

import numpy as np

[**Abre 2.6. Numpy en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_31_numpy.ipynb)

## 2.7. Pandas

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_32_pandas.ipynb)

Abreviación de Panel Data, los objetos pandas dan soporte al análisi de datos con columnas de distinta tipología: categóricas, binarias, numéricas…

Respecto a los numpy, permiten indexar las filas y columnas. Este tipo de indexación, facilita, por ejemplo, realizar análisis de series temporales, no necesariamente con una frecuencia fija.

Como siempre el primer paso es cargar la librería con *namespace* pd

import pandas as pd

##### ACTIVIDAD GUIADA 2.4

Esta vez, se trata de seleccionar el mejor plantea posible para ubicar la academia Jedi. Los parámetros de la búsqueda son:

* Días largos para entrenar mucho
* Mucha agua para poder refrescarse
* Poca densidad de población para no ser molestado
* Buen clima (temperado o tropical)

«Si existe un auténtico centro del universo, ahora estás en el planeta más alejado de él»

―Luke Skywalker sobre Tatooine el planeta desértico dónde nacieron Anakin y Luke Skywalker.

El primer paso consiste en cargar los datos de los planetas.

import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns; sns.set() # para el estilo de graficos  
  
entidades = ['planets','starships','vehicles','people','species']  
entidades\_df = {x: pd.read\_pickle('www/' + x + '\_df.pkl') for x in entidades}  
planets\_df = entidades\_df['planets'][["rotation\_period","diameter","climate"  
 ,"temperate\_tropical","gravity"  
 ,"surface\_water","population"]].dropna()  
planets\_df = planets\_df[planets\_df.diameter>0]  
  
  
planets\_df.head()

## rotation\_period diameter ... surface\_water population  
## name ...   
## Tatooine 23.0 10465.0 ... 1.0 2.000000e+05  
## Alderaan 24.0 12500.0 ... 40.0 2.000000e+09  
## Yavin IV 24.0 10200.0 ... 8.0 1.000000e+03  
## Bespin 12.0 118000.0 ... 0.0 6.000000e+06  
## Endor 18.0 4900.0 ... 8.0 3.000000e+07  
##   
## [5 rows x 7 columns]

[**Abre 2.7. Pandas en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_32_pandas.ipynb)

## 2.8. Control de flujo

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_40_contr_flujo.ipynb)

Las herramientas de control de flujo permiten automatizar tareas. En este capítulo se trabajarán las funciones condicionales y bucles.

##### DATOS DE EJEMPLO

Este capítulo, utilizara ejemplos de vehículos de Star Wars.

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns; sns.set() # para el estilo de graficos  
  
entidades = ['planets','starships','vehicles','people','species']  
entidades\_df = {x: pd.read\_pickle('www/' + x + '\_df.pkl') for x in entidades}  
vehicles\_df = entidades\_df['vehicles'][["cost\_in\_credits","length","max\_atmosphering\_speed"  
 ,"crew","cargo\_capacity","pilots","films"]].dropna()  
  
vehicles\_df.head()

## cost\_in\_credits ... films  
## name ...   
## Sand Crawler 150000.0 ... [http://swapi.dev/api/films/1/, http://swapi.d...  
## T-16 skyhopper 14500.0 ... [http://swapi.dev/api/films/1/]  
## X-34 landspeeder 10550.0 ... [http://swapi.dev/api/films/1/]  
## Storm IV Twin-Pod cloud car 75000.0 ... [http://swapi.dev/api/films/2/]  
## Sail barge 285000.0 ... [http://swapi.dev/api/films/3/]  
##   
## [5 rows x 7 columns]

### 2.8.1. Funciones

Hagamos un breve repaso sobre lo que se ha tratado hasta el momento respecto a las funciones en Python.

Se ha introducido el concepto de función en Python en el capítulo de [Elementos básicos de Python](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_10_elem_bas.ipynb).

A continuación, en el capítulo de [Listas, tuplas y diccionarios](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_30_colec_obj.ipynb), se ha explicado la posibilidad de devolver varios elementos a la vez en forma de tuplas.

En [numpy](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_31_numpy.ipynb) se han explicado las funciones universales (ufunc), como una grupo de funciones que vectorizan al aplicarse sobre los elementos de los array numpy.

Finalmente, en [pandas](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_32_pandas.ipynb), se ha comentado la posibilidad de aplicar las funciones universales de numpy a objetos pandas (Series y DataFrame) y como estas funciones respetan los índices fila y columna. También se ha comentado el alineamiento de los índices cuando se realizan operaciones aritméticas entre dos objetos pandas.

En el capítulo de gestión de datos, se tratará 2 conceptos adicionales sobre las funciones.

Por un lado, se verá como vectorizar cualquier tipo de función, sobre los elementos de los objetos pandas, a través de la función apply.

**Recuerda**: En el tema de R las funciones apply ya se definieron con la misma finalidad.

El segundo concepto, van a ser las funciones de agregación (suma, media, cuantiles…). Éstas se definirán primero sobre objetos numpy y posteriormente sobre pandas.

La combinación de ambos conceptos, funciones vectorizadas y funciones de agregación sobre objetos pandas, van a terminar de rellenar tu caja de herramientas para poder iniciarte en data science.

### 2.8.2. Condicionales

Hagamos también aquí un breve repaso de los aspectos que ya se han comentado respecto a estructuras condicionales.

En el capítulo de los [Elementos básicos de Python](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_10_elem_bas.ipynb) se ha introducido la estructura básica *if-else*.

En el capítulo [Listas, tuplas y diccionarios](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_30_colec_obj.ipynb) se ha utilizado la escritura if-else en las *list comprehension*.

[x for x in range(5) if x%2==0] # lista de pares menores a 5

## [0, 2, 4]

Los capítulos de [Numpy](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_31_numpy.ipynb) y [Pandas](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_32_pandas.ipynb) también han servido para tratar los filtros, como un forma de operaciones condicionales sobre vectores y data frames.

Una nueva forma de realizar transformaciones condicionales con arrays numpy y series pandas es np.where().

Por ejemplo, veamos la siguiente variable *crew* que nos informa del número de tripulantes que puede llevar como máximo un vehículo.

vehicles\_df[['crew']].head()

## crew  
## name   
## Sand Crawler 46  
## T-16 skyhopper 1  
## X-34 landspeeder 1  
## Storm IV Twin-Pod cloud car 2  
## Sail barge 26

Para transformar esta columna en una nueva que indique si un vehículo puede llevar 1, 2 o 3 o más tripulantes.

vehicles\_df['crew\_r'] = np.where(vehicles\_df.crew<3, vehicles\_df.crew, 3)  
vehicles\_df[['crew','crew\_r']].head()

## crew crew\_r  
## name   
## Sand Crawler 46 3  
## T-16 skyhopper 1 1  
## X-34 landspeeder 1 1  
## Storm IV Twin-Pod cloud car 2 2  
## Sail barge 26 3

**Observa**: esta función de numpy aplica elemento a elemento. Observa también que se ha aplicado a una serie de un DataFrame, obteniendo una nueva serie.

### 2.8.3. Bucles

Los bucles for, se han tratado de forma básica en las *list comprehension* del capítulo de [Listas, tuplas y diccionarios](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/Rmd/modulo1_tema4_Py_30_colec_obj.ipynb).

Una forma de definir bucles con for es a través de listas y diccionarios.

crew\_list = [1,2,3]  
for n\_crew in crew\_list:  
   
 n\_vehi = vehicles\_df[vehicles\_df.crew\_r == n\_crew].shape[0]  
   
 if n\_vehi == 1:  
 txt\_vehicles = 'vehículo'  
 else:  
 txt\_vehicles = 'vehiculos'  
   
 txt\_crew = 'tripulante'  
 if n\_crew != 1:  
 txt\_crew = txt\_crew + 's'  
   
 txt\_n\_crew = str(n\_crew)  
 if n\_crew==3:  
 txt\_n\_crew = txt\_n\_crew + ' o más'  
   
 print("Hay {} {} con {} {}".format(n\_vehi, txt\_vehicles, txt\_n\_crew, txt\_crew))

## Hay 8 vehiculos con 1 tripulante  
## Hay 3 vehiculos con 2 tripulantes  
## Hay 6 vehiculos con 3 o más tripulantes

Este bucle recorre la lista n\_crew y nos indica el número de vehículos con esa cantidad de tripulantes.

Cuando se recorre un diccionario con un for, por defecto, éste lo hace por sus claves.

vehiculos\_tripulantes = {'un': 8, 'dos': 3, '3 o más': 6}  
  
for x in vehiculos\_tripulantes:  
 print(x)

## un  
## dos  
## 3 o más

Se puede iterar sobre la tupla (clave,valor) con .items()

for txt\_n\_crew, n\_vehi in vehiculos\_tripulantes.items():  
 print("Hay",n\_vehi,"vehículo/s con",txt\_n\_crew, "tripulante/s")

## Hay 8 vehículo/s con un tripulante/s  
## Hay 3 vehículo/s con dos tripulante/s  
## Hay 6 vehículo/s con 3 o más tripulante/s

Los bucles con numpy, tienen una lógica simétrica a las listas.

Para poder recorrer 2 colecciones de objetos (listas, diccionarios, numpy, series…) con el mismo número de elementos de forma correlativa se utiliza la función zip().

a = [1,2,3]  
b = [8,3,6]  
  
for x,y in zip(a,b):  
 print("Hay",y,"vehículo/s con",[x if x<3 else '3 o más'][0], "tripulante/s")

## Hay 8 vehículo/s con 1 tripulante/s  
## Hay 3 vehículo/s con 2 tripulante/s  
## Hay 6 vehículo/s con 3 o más tripulante/s

Para iterar **por columnas** de un DataFrame se utiliza .iteritems(). En cada iteración se obtiene la tupla(nombre columna, valor serie).

for nombreCol,variable in vehicles\_df.iteritems():  
 print(nombreCol,", tiene tipo:",variable.dtype)

## cost\_in\_credits , tiene tipo: float64  
## length , tiene tipo: float64  
## max\_atmosphering\_speed , tiene tipo: float64  
## crew , tiene tipo: int64  
## cargo\_capacity , tiene tipo: float64  
## pilots , tiene tipo: object  
## films , tiene tipo: object  
## crew\_r , tiene tipo: int64

De la misma forma, pero ahora iterando un DataFrame **por filas** con .iterrows().

for nombre,fila in vehicles\_df.iloc[0:2,0:2].iterrows():  
 print(nombre,", cuesta:",fila.cost\_in\_credits," y mide:",fila.length , "metros")

## Sand Crawler , cuesta: 150000.0 y mide: 36.8 metros  
## T-16 skyhopper , cuesta: 14500.0 y mide: 10.4 metros

## 2.9. Gestión de datos

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_50_gest_dat.ipynb)

A continuación, se presentan las funciones para la lectura/escritura de datos, cruce y construcción de tablas resumen.

Al final del capítulo se presenta la forma de manejar datos temporales.

##### ACTIVIDAD GUIADA 2.5

Se trata de analizar los personajes de la serie:

«Preferiría ser un monstruo que cree en algo, que sacrificaría todo para mejorar la galaxia, que ser alguien que se quede al margen y mire como si no tuviera repercusión en ellos.»

―Princesa Leia Organa

Esta actividad consiste en cruzar datos de personajes y planetas para construir descriptivos resumen de los datos de personajes.

El primer paso consiste en cargar los datos de los personajes y planetas.

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns; sns.set() # para el estilo de graficos  
  
entidades = ['planets','starships','vehicles','people','species']  
entidades\_df = {x: pd.read\_pickle('www/' + x + '\_df.pkl') for x in entidades}  
  
# planetas  
planets\_df = entidades\_df['planets'][["climate","temperate\_tropical","population","url"]].dropna()  
  
# Datos principales  
people\_df = entidades\_df['people'][["height","mass","eye\_color","birth\_year","gender","homeworld"]].dropna()  
  
people\_df.head()

## height mass ... gender homeworld  
## name ...   
## Luke Skywalker 172.0 77.0 ... male http://swapi.dev/api/planets/1/  
## C-3PO 167.0 75.0 ... none http://swapi.dev/api/planets/1/  
## R2-D2 96.0 32.0 ... none http://swapi.dev/api/planets/8/  
## Darth Vader 202.0 136.0 ... male http://swapi.dev/api/planets/1/  
## Leia Organa 150.0 49.0 ... female http://swapi.dev/api/planets/2/  
##   
## [5 rows x 6 columns]

planets\_df.head()

## climate ... url  
## name ...   
## Tatooine arid ... http://swapi.dev/api/planets/1/  
## Alderaan temperate ... http://swapi.dev/api/planets/2/  
## Yavin IV temperate, tropical ... http://swapi.dev/api/planets/3/  
## Bespin temperate ... http://swapi.dev/api/planets/6/  
## Endor temperate ... http://swapi.dev/api/planets/7/  
##   
## [5 rows x 4 columns]

[**Abre 2.9. Gestión de datos en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_50_gest_dat.ipynb)

# Ideas clave

* Python es un lenguje de propósito general con un conjunto de paquetes que facilitan la computacion científica en el ámbito de la ciencia de datos.
* Python permite la visualización de datos.
* Python es un lenguaje perfectamente válido para entornos productivos de tratamientos de datos.
* Python interactua de forma simple con distintas bases de datos.
* El entorno Jupyter/Colab aportan capacidades interactivas a Python. Permite tanto desarrollar programas, como análisis de datos.
* Se utiliza en todas las fases de análisis de datos:
  + Adquisición y preparación de los datos (bases de datos, ficheros planos, etc.).
  + Análisis de los datos: construcción de modelos predictivos, de agrupamiento o clasificación, utilidades gráficas para la visualización de datos, etc.
  + Manejo y almacenamiento efectivo de los datos como son los cálculos con matrices. Y un lenguaje de programación bien desarrollado que incluye saltos condicionales, bucles, funciones recursivas, utilidades para la entrada y salida de datos, etc.
  + Comunicación de los resultados: realización de informes para presentación de los resultados y conclusiones. Utiliza Jupyter/Colab combinado con Markdown.
  + Aplicación de los resultados obtenidos: por ejemplo, utilización de modelos predictivos desarrollados para en función de una serie de datos históricos (datos de entrenamiento y test del modelo) predecir ciertas salidas.

# Anexo: Readme de Python

[**Abre en Colab**](https://colab.research.google.com/github/griu/mbdds_fc20/blob/master/Python/modulo1_tema4_Py_60_anexo.ipynb)

#### PREPARACIÓN DEL ENTORNO COLAB

Des de [Colab](http://colab.research.google.com/), hay que clonar el repositorio cada vez que inicias un nuevo libro. En los libros se incluye el código necesario para ello.

#### PREPARACIÓN ENTORNO LOCAL-JUPYTER (OPCIONAL)

##### CLONAR REPOSITORIO

En local puedes utilizar el mismo proyecto que has clonado en el [README DE R](../../../../../C:/Users/usuari/Documents/GitHub/mbdds_fc20/Python/R/README.md).

Para actualizarlo de nuevo, des de consola.

cd mbdds\_fc20  
git pull  
cd Python

##### CREAR EN LOCAL UN NUEVO ENVIRONMENT DE ANACONDA

Abrimos una línea de comandos (con *Anaconda 3.0* ya disponible).

* Windows: Escribimos Anaconda en el menú Inicio y aparecerá la consola MS-DOS de Anaconda.
* Linux: Abrimos Terminal

conda deactivate  
conda create -n mbdds\_rpy20 python=3.6.9  
conda activate mbdds\_rpy20

Verifica que se ha creado y está activo.

conda info --envs

##### INSTALA LAS LIBRERIAS DE PYTHON

cd mbdds\_fc20/Python  
conda activate mbdds\_rpy20  
python -m pip install -r requirementsColab.txt

##### PUBLICAR EL KERNEL

Para acceder al nuevo environment desde Jupyter necesitas publicar el kernel.

python -m ipykernel install --user --name mbdds\_rpy20 --display-name "mbdds\_rpy20"

Puede tardar unos minutos en publicarse.

##### LANZAR ENTORNO JUPYTER NOTEBOOK

Para acceder al servidor Jupyter.

conda activate mbdds\_rpy20  
jupyter notebook

Debería abrirse un navegador con acceso a Jupyter desde donde podrás acceder a los notebooks. Habitualmente el servidor Jupyter se abre en <http://localhost:8888/> .

#### IMPORTAR DATOS DE STARWARS SWAPI

Datos procesados a partir de <https://swapi.dev/documentation#root>.

import requests  
import pandas as pd  
import numpy as np

# exec(open('get\_entity.py').read())  
def get\_entity(entity = None, url\_base = 'http://swapi.dev/api/'):  
# https://swapi.dev/documentation#root  
   
 page\_num = 1  
 results\_acum = []  
 entity\_list = ["films","people","planets","species","starships","vehicles"]  
   
 if(entity not in entity\_list):  
 page\_num = None  
 print('Call a valid entity: ' + ','.join(entity\_list))  
   
 while(page\_num is not None):  
 url = "".join([url\_base, entity, '/?page=', str(page\_num)])  
 rjson = requests.get(url).json()  
 results\_acum = results\_acum + rjson['results']  
 if(rjson['next'] is not None):  
 page\_num = page\_num + 1  
 else:  
 page\_num = None  
   
 if (len(results\_acum) > 0):  
 results\_df = pd.DataFrame(results\_acum)  
 else:  
 results\_df = entity\_list  
   
 return(results\_df)  
  
def str2num(df,num\_list):  
 num\_list = [lab for lab in num\_list if lab in df.columns]  
 num\_list = [lab for lab in num\_list if df[lab].dtype==object]  
 for x in num\_list:  
 if (x=="crew"):  
 df.loc[df[x] == "30-165",x] = "165"  
 df[x] = pd.to\_numeric(df[x].replace('[^0-9\.-]', '', regex=True), errors='coerce')  
 return(df)

res = get\_entity()

res

films\_df = get\_entity("films")  
people\_df = get\_entity("people")  
planets\_df = get\_entity("planets")  
species\_df = get\_entity("species")  
starships\_df = get\_entity("starships")  
vehicles\_df = get\_entity("vehicles")  
  
people\_df.index = people\_df.name  
planets\_df.index = planets\_df.name  
species\_df.index = species\_df.name  
starships\_df.index = starships\_df.name  
vehicles\_df.index = vehicles\_df.name  
  
people\_df.drop("name",inplace=True, axis=1)  
planets\_df.drop("name",inplace=True, axis=1)  
species\_df.drop("name",inplace=True, axis=1)  
starships\_df.drop("name",inplace=True, axis=1)  
vehicles\_df.drop("name",inplace=True, axis=1)

people\_num = ["height","mass","birth\_year"]  
planets\_num = ["diameter","rotation\_period","orbital\_period","population","surface\_water", "gravity"]  
species\_num = ["average\_height","average\_lifespan"]  
starships\_num = ["passengers","length","crew","cost\_in\_credits","cargo\_capacity"  
 ,"hyperdrive\_rating","MGLT","max\_atmosphering\_speed"]  
vehicles\_num = ["cargo\_capacity","cost\_in\_credits","crew","length","max\_atmosphering\_speed","passengers"]

people\_df.loc[(people\_df["gender"].isin(['n/a', 'none'])) | people\_df["gender"].isna(),"gender"] = 'none'  
planets\_df["temperate\_tropical"] = planets\_df.climate.apply(lambda x: ("temperate" in x) or ("tropical" in x)).astype(int)

people\_df2 = str2num(people\_df,people\_num)  
planets\_df2 = str2num(planets\_df,planets\_num)  
species\_df2 = str2num(species\_df,species\_num)  
starships\_df2 = str2num(starships\_df,starships\_num)  
vehicles\_df2 = str2num(vehicles\_df,vehicles\_num)

planets\_df2.loc["Bespin","gravity"] = 1.5

people\_df2.to\_pickle("www/people\_df.pkl")  
planets\_df2.to\_pickle("www/planets\_df.pkl")  
species\_df2.to\_pickle("www/species\_df.pkl")  
starships\_df2.to\_pickle("www/starships\_df.pkl")  
vehicles\_df2.to\_pickle("www/vehicles\_df.pkl")

vehicles\_df2.describe()

# Bibliografía

* W. McKinney. Python for Data Anlysis. O’Reilly Media, Inc. 2nd ed.; 2017

Profundizar en el uso de Pandas, NumPy y Matplotlib para el análisis y preparación de datos.

* J. VanderPlas. Python Data Science Handbook. O’Reilly Media, Inc.; 2016. Disponible en: <http://faculty.marshall.usc.edu/gareth-james/ISL/>

Profundizar en el uso de Pandas, Numpy y Machine Learning.

* C. R. Severance. Python para todos. Elliott Hauser, Sue Blumenberg; 2009. Disponible en: <https://www.py4e.com/book.php>

Cubre todo lo básico de Python más la interacción con bases de datos, visualización y modelado.

# Recursos en internet

* [Cheatsheet de Python](https://www.pythoncheatsheet.org/)