

# Inteligencia Artificial Moderna

## Contents

Introducción a la Programación con Python

- [1. Introducción a Python](#)
- [2. Introducción a la programación con Python](#)
- [3. IDEs](#)
- [4. Hola Jupyter!](#)
- [5. Declarando variables](#)
- [6. Ejercicios](#)
- [7. Ejercicios I](#)
- [8. Listas, Funciones, Errores](#)
- [9. Control de flujo](#)
- [10. Tuplas](#)
- [11. Tipos de errores](#)
- [12. Manejo de errores](#)
- [13. Listas por comprensión \(List Comprehension\)](#)
- [14. I/O](#)
- [15. Funciones](#)
- [16. Ejercicios II](#)
- [17. Ejercicios Adicionales II](#)
- [18. Programación orientada a objetos](#)
- [19. Ejercicios POO IV](#)
- [20. Aplicación de POO: Experimento de Tiempo de Reacción](#)
- [21. Aplicación de POO: PACMAN](#)

Automatización y Minería Web

- [22. Automatización y Minería Web](#)
- [23. Automatización:](#)
- [24. Bash y OS](#)
- [25. Automatización II: outputs, envío de mail, volcado a GSheets, scheduling](#)
- [26. Ejercicios Automatización II](#)
- [27. Expresiones Regulares](#)
- [28. Práctica: GSheet y PyTrends](#)
- [29. Ejercicio: Organizando PDFs](#)
- [30. Edición de Imágenes con Python](#)
- [31. Color](#)
- [32. Métodos de manipulación](#)

APIs

- [33. APIs](#)
- [34. APIs geográficas](#)
- [35. APIs Geográficas: Ejercicios](#)
- [36. APIs de Series de Tiempo](#)

- [37. APIs Series de Tiempo: Ejercicios](#)

Colaboración

- [38. Generando contenido](#)

Primera versión de la formación educativa de Humai en formato libro.

## Tabla de contenidos:

### 1. Introducción a Python

En este curso se introduce el lenguaje de programación Python. Cada carpeta contiene una clase que, a su vez, incluye una Jupyter Notebook con la clase en sí y una carpeta con ejercicios, exceptuando la última clase que es un proyecto integrador.

Cada Jupyter se puede ejecutar de manera local usando el programa Jupyter Notebook que viene con la distribución de Python conocida como Anaconda. Recomendamos que intenten instalarse esta distribución ya que es estándar su uso en la industria. Para hacerlo puede hacer clic en downloads acá: <https://www.anaconda.com/products/individual>

Alternativamente, cada notebook incluye un botón para abrir en Google Colab, el cual es un producto gratuito de Google, es una Jupyter Notebook en la nube. El botón lo pueden encontrar al comienzo de cada clase. Finalmente, también incluimos acá las rutas a las que redirige ese botón, que es <https://colab.research.google.com> + la ruta en github.

También te dejamos los links de los videos que acompañan a cada una de las clases dadas.

1. Tipos de Datos: aquí introducimos los tipos de datos básicos, condicionales y bucles. Clase elaborada por Matías Grinberg, María Gaska y Leonardo Ignacio Córdoba.
  - [Clase](#)
  - [Ejercicio](#)
  - [Video](#) - Clase dictada por María Gaska.
1. Listas y Funciones: en esta clase ahondamos en listas y nos centramos en entender errores, cómo manejarlos, I/O y funciones. Clase elaborada por Matías Grinberg, María Gaska, Leonardo Ignacio Córdoba y Gastón Bujía.
  - [Clase](#)
  - [Ejercicio](#)
  - [Ejercicios Adicionales](#)
  - [Video](#) - Clase dictada por Leonardo Córdoba
1. Módulos y Funciones: en la tercera clase del curso proponemos continuar estudiando funciones, más orientados a algoritmos. Además, vemos cómo importar módulos. Clase elaborada por María Gaska, Leonardo Ignacio Córdoba y Gastón Bujía.
  - [Clase](#)
  - [Ejercicio](#)
  - [Video](#) - Clase dictada por Gaston Bujia
1. Introducción a la Programación Orientada a Objetos: en esta carpeta van a encontrar una introducción a la Programación Orientada a Objetos. Clase elaborada por María Gaska, Leonardo Ignacio Córdoba y Matías Grinberg.
  - [Clase](#)
  - [Ejercicio](#)
  - [Video](#) - Clase dictada por María Gaska
1. Proyecto de Aplicación de POO: Experimento Psicométrico: en esta clase usaremos lo visto para hacer un pequeño experimento de tiempo de reacción que involucra elementos iniciales de Programación Orientada a Objetos, manejo y análisis de datos, y automatización para generar un reporte de los resultados. Clase elaborada por Matías Grinberg.
  - [Clase](#)
  - [Video](#) - Clase dictada por Matias Grinberg

1. Proyecto de Aplicación de POO: PACMAN: finalmente, como cierre proponemos seguir profundizando con POO al mismo tiempo que armamos una versión simple del popular juego Pacman. Clase elaborada por María Gaska y Leonardo Ignacio Córdoba.

- [Clase](#)

 Open in Colab

Recordá abrir en una nueva pestaña

## 2. Introducción a la programación con Python

### I. Introducción a la programación con Python

### II. Tabla de Contenidos

#### I. Python

#### III. IDEs

- I. VS Code
- II. Pycharm
- III. Spyder

#### IV. Hola Jupyter!

#### V. Declarando variables

- I. Numéricos
- II. Operaciones numéricas - Divisiones
- III. Booleanos
  - I. Operaciones lógicas

#### VI. Ejercicios

- I. Strings
  - I. Ejercicios
- II. Listas
  - I. Indexing y slicing en listas
  - II. Otras operaciones y métodos
- III. Diccionarios
- IV. Condicionales
- V. Bucles o Loops
  - I. For loop
  - II. While loop
- VI. Recursos y tips
- VII. Tips en la práctica

### 2.1. Python

- Elegante, sencillo
- Comunidad y accesibilidad
- Desarrollos de estado del arte
- Flexibilidad “full stack”
- Multi-paradigma

## 3. IDEs

### 3.1. VS Code

The screenshot shows the VS Code interface with the following details:

- File Explorer (Left):** Shows the project structure. The `src` folder contains `components`, `images`, `pages`, `index.js`, `utils.js`, `.gitignore`, `gatsby-config.js`, and `gatsby-node.js`. The `components` folder contains `blog-post.js`.
- Editor (Top Right):** The file `blog-post.js` is open. A code completion dropdown is visible at the cursor position, showing suggestions like `data`, `dateFormat`, `debug`, etc.
- Terminal (Bottom):** Shows the output of a build command:

```
info i [wdm]: Compiled successfully.  
info changed file at  
WAIT Compiling...  
9:51:57 AM  
  
info i [wdm]: Compiling...  
DONE Compiled successfully in 63ms  
9:51:58 AM  
  
info i [wdm]:  
info i [wdm]: Compiled successfully.
```

## 3.2. Pycharm

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The project is named "cdf" and is located at "/Users/.../Documents/Programming/Python/cdf". It contains files like `cdf.py`, `setup.py`, `README.rst`, `gitignore`, and `speed_acccpy`.
- Code Editor:** The main file `cdf.py` is open. The code defines a class `CDF` with methods `cdf` and `cdf_p_plot`.
  - Method `cdf`:** Calculates the cumulative distribution function (CDF). It takes parameters: `df` (pandas DataFrame), `conditions` (string or list of strings), and `conditions` (list of strings). It returns a pandas DataFrame containing CDFs for each condition.
  - Method `cdf_p_plot`:** Plots the cumulative distribution functions. It takes parameters: `cdf_data` (pandas DataFrame), `save_file` (boolean), `legend` (boolean), and `im_size` (integer).
- Toolbars and Status Bar:** The top bar includes standard options like File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help. The status bar at the bottom indicates "2 processes running" and "File: CDF-master.py".

### 3.3. Spyder

```

6
7 import pylab
8 from numpy import cos, linspace, pi, sin, random
9 from scipy.interpolate import splprep, splev
10
11 # %% Generate data for analysis
12
13 # Make ascending spiral in 3-space
14 t = linspace(0, 1.75 * 2 * pi, 100)
15
16 x = sin(t)
17 y = cos(t)
18 z = t
19
20 # Add noise
21 x += random.normal(scale=0.1, size=x.shape)
22 y += random.normal(scale=0.1, size=y.shape)
23 z += random.normal(scale=0.1, size=z.shape)
24
25
26 # %% Perform calculations
27
28 # Spline parameters
29 smoothness = 3.0 # Smoothness parameter
30 k_param = 2 # Spline order
31 nests = -1 # Estimate of number of knots needed (-1 = maximal)
32
33 # Find the knot points
34 knot_points, u = splprep([x, y, z], s=smoothness, k=k_param, nests=-1)
35
36 # Evaluate spline, including interpolated points
37 xnew, ynew, znew = splev(linspace(0, 1, 400), knot_points)
38
39
40 # %% Plot results
41
42 # TODO: Rewrite to avoid code smell
43 pylab.subplot(2, 1)
44 data = pylab.plot(x, y, 'bo-', label='Data with X-Y Cross Section')
45 fit, = pylab.plot(xnew, ynew, 'r-', label='Fit with X-Y Cross Section')
46 pylab.legend()
47 pylab.xlabel('x')
48 pylab.ylabel('y')
49
50 pylab.subplot(2, 2)
51 data = pylab.plot(x, z, 'bo-', label='Data with X-Z Cross Section')
52 fit, = pylab.plot(xnew, znew, 'r-', label='Fit with X-Z Cross Section')
53 pylab.legend()
54 pylab.xlabel('x')

```

The Variable explorer shows the following data structures:

Name	Type	Size	Value
array_int8	int8	(2, 3)	Min: -7 Max: 6
array_uint32	uint32	(2, 2, 3)	Min: 1 Max: 7
bars	container.BarContainer	20	BarContainer object of matplotlib.container
df	DataFrame	(3, 2)	Column names: bools, ints
filename	str	1	C:\ProgramData\Anaconda3\lib\site-pacak...
list_test	list	2	[Dataframe, Numpy array]
nrows	int	1	344
r	float64	1	7.611082589334796
radii	float64	(20,)	Min: 0.4983036538535687 Max: 9.856846974942551
region	tuple	2	(slice, slice)
rgb	float64	(45, 45, 4)	Min: 0.0 Max: 1.0
series	Series	(1,)	Series object of pandas.core.series.mod...
test_none	NoneType	1	NoneType object of builtins module

The IPython console shows the following code and output:

```

...:
...:     ...: ls = LightSource(270, 45)
...:     ...: # To use a custom hillshading mode, override the built-in shading
...:     ...: # in the rgba colors of the shaded surface calculated from "shade".
...:     ...: rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_mode='soft')
...:     ...: surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, facecolors=rgb,
...:     ...:                         linewidth=0, antialiased=False, shade=False)
...:     ...:
...:     ...: plt.show()

```

## 4. Hola Jupyter!

En esta *notebook* conoceremos el entorno Jupyter (o Google Colaboratory, una versión online gratuita), interfaz incluida en la instalación de Python ampliamente usada para Data Science llamada [Anaconda](#). Para seguir el curso offline, luego de descargar e instalar Anaconda, si usan Windows deberán buscar el programa en su computadora y ejecutar Jupyter.

Si están en una distribución de linux o Mac pueden ejecutar en la terminal:

```
jupyter notebook
```

Cada una de estas *celdas* funciona como un bloque donde podemos escribir texto plano, Latex, HTML, además de ejecutar código Python, R, bash y otros.

```
\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i
```

## 5. Declarando variables

Existen distintos tipos de variables:

- int: entero
- float: punto flotante
- string: cadena de caracteres
- bool: booleano, 0, 1, True o False

Y estructuras de datos, como:

- list: lista de elementos (de cualquier tipo, incluida otra lista)
- dict: “diccionario”, conjunto de pares llave:valor

Veamos un primer ejemplo

```
primera_variable = 'Hola mundo!'
print(primera_variable)
```

Declaramos una variable llamada `primera_variable` asignándole el valor de "Hola mundo!". La función `print` toma entre paréntesis esa variable y muestra el valor en pantalla.

En Jupyter la última línea de una celda se imprime al ejecutar...

```
primera_variable
```

## 5.1. Numéricos

Declaramos dos variables de tipo numérico:

```
x = 5
y = 17
```

Podemos hacer las operaciones numéricas usuales:

```
print(x+y) # suma
print(x*2) # multiplicacion
print(x**(1/2)) # elevado a
```

## 5.2. Operaciones numéricas - Divisiones

Devuelve la división:

```
print(y / x)
```

Devuelve la parte entera de la división:

```
print(y // x)
```

Devuelve el resto:

```
print(y % x)
```

Otros ejemplos:

```
un_string = 'variable de ejemplo'
un_bool = True
```

La función `type` recibe de argumento una variable y devuelve su tipo

```
type(x), type(y)
```

Intentemos la siguiente suma...

```
5 + "10"
```

¿Qué paso?

Si convertimos el string "10" a tipo int

```
5 + int('10')
```

Cada tipo de variable responde a ciertos **métodos**. Veamos las operaciones lógicas, que utilizan **booleanos**

## 5.3. Booleanos

### 5.3.1. Operaciones lógicas

```
print(10 >= 9)

print("palabra" == "palabra")
print("otras palabras" != "otras palabras")

print("test" in "testing")
a = 'testing'
print('test' in a)
```

Los operadores disponibles son:

```
Relacionales:
- >= , <=, <, > : Mayor o igual, menor o igual, mayor o menor
- != , == : Distinto, Igual
- in : Contenido por
Lógicos:
- not o ~ : Negación
- and o & : Ambas verdaderas
- or o | : Una u otra es verdadera
```

Como booleanos, se pueden utilizar tanto 0 y 1 como True y False.

## 6. Ejercicios

1- Definan a = 5, b = 7 y c = 8

Prueben las siguientes operaciones booleanas:

2- a > b

3- a < b

4- a + b < c

5- (a + b > c) | (a + b < c) → a + b o es mayor a c o es menor

6- (a + b > c) & (a + b < c) → a + b es mayor a c y es menor a c

## 6.1. Strings

```
"dos strings se pueden " + "sumar"
```

Podemos acceder a ellos mediante **índices**. Los mismos funcionan para cualquier **iterable**, que es un objeto con muchos elementos que son accesibles secuencialmente. Podemos usar los índices para acceder a una posición determinada, pasándola entre corchetes [posición]. Es IMPORTANTE mencionar que en Python la primera posición tiene el índice 0 (y no 1).

```
texto = "programación en python"
```

```
texto[0]
```

También podemos usar los índices para traer más de un elemento al mismo tiempo, usando el **slicing**. El slicing lleva tres parámetros: comienzo (start), final (stop) e intervalo o paso (step).

- El parámetro comienzo (start) indica la primera posición incluida en la selección, por default es 0.
- El parámetro final (stop) es la primera posición que NO va a estar incluída, por default se incluyen todos los elementos.

- El parámetro intervalo o paso (step), es optativo e indica el tamaño del paso entre seleccionar un elemento y el siguiente, por default el paso es 1.  
El paso también puede ser negativo, en este caso contamos desde el final hacia el comienzo, esto es muy útil para dar vuelta los strings...  
[comienzo : final : intervalo] (en inglés es [start : stop : step])

El intervalo que se recibe es semi-aberto (indice\_primer, indice\_segundo], es decir se incluye el primero y no el segundo valor. Se admiten números negativos, que contabilizan desde el final.

```
texto[:]
```

```
texto[:1]
```

```
texto[1:4]
```

```
texto[::-1]
```

### Ejercicio

1- Dado ese string devuelvan únicamente la palabra python usando slicing

```
texto
```

```
texto[16:23]
```

```
texto[16:23]
```

```
texto[-6:]
```

```
texto[-6:23:1]
```

Veamos algunos otros métodos de los strings

```
texto.upper()
```

```
texto.title()
```

```
texto.split() # Devuelve una lista, de ésto vamos a hablar a continuación
```

Otras formas de combinar variables y texto...

Rellenamos un string con `.format(query, modales)`:

```
tema = 'data science'
modales = 'por favor'

'Dame información de {0} si esta disponible, {1}'.format(tema, modales)
```

```
#Otra manera
print(f'Dame información de {tema.upper()} si esta disponible, {modales}')
```

Otros métodos a mencionar son:

```
"1000".isdigit()
```

```
texto.replace(' ', '...')
```

```
" quitando espacios ".strip()
```

```
texto = "Me gusta el 37" # Piso el contenido de la variable para hacer otras pruebas
texto.isalpha()
```

```
texto = "Me" # Piso el contenido de la variable
texto.isalpha()
```

### 6.1.1. Ejercicios

1- Concatenar los string "hola" y "qué tal", separando ambos strings con un espacio (Tip: un espacio es: " ")

2- Guardar el resultado anterior en una variable y pasar el texto a mayúsculas.

```
# Ejercicio 1
```

```
# Ejercicio 2
```

## 6.2. Listas

Las listas son un conjunto de elementos ordenados. Estos elementos pueden ser de cualquier tipo, incluyendo otras listas. Veamos algunas operaciones con ellas.

```
amigos = ['Mateo', 'Nico', 'Claudia', 'Ernestina', 'Paola']
```

### 6.2.1. Indexing y slicing en listas

De la misma forma que con los strings en el contexto de las listas generalmente hablamos de **indexación** o acceso por índice a la acción de encontrar un valor según su posición en la lista.

Importantísimo (sí, ¡una vez más!): En Python el primer elemento se indexa con el valor 0. Es decir, si queremos el primer valor de una lista tenemos que llamar a la posición 0.

```
amigos[0]
```

Además, el último elemento se indexa como -1, el siguiente -2 y así sucesivamente. Entonces para acceder al último elemento:

```
amigos[-1]
```

Además, podemos acceder a varios elementos en simultáneo, usando el **slicing** de la misma manera que con strings.

El slicing aplicado a una lista nos devuelve una **lista**.

Veamos algunos ejemplos:

```
amigos[:] # arranco en 0 hasta el final, con el paso default (1)
```

```
amigos[:1] # arranco en 0 y el primero sin incluir es el 1, con el paso default (1)
```

```
amigos[:-1] # excluimos el último elemento
```

```
amigos[::-2] # salteamos un elemento a la vez
```

Ejercicios:

1- ¿Es el resultado de amigos[:1] igual al de acceder directamente a la primera posición?

2- Recorrer la lista de elementos de atrás para adelante.

3- Seleccionar del 3er elemento al 4to elemento.

### 6.2.2. Otras operaciones y métodos

Agregar un nuevo elemento:

```
amigos.append('Chicharito')
print(amigos)

amigos = amigos + ['Chicharito']

x = 10

x = x + 3

x
```

Sumamos otra lista:

```
amigos = amigos + ['Pipi', 'Toto']
print(amigos)
```

Unir una lista con un separador dado:

```
ejemplo = ['valor1', 'valor2', 'valor3']
';'.join(ejemplo)
```

El ejercicio 1 se podría haber resuelto usando .join así:

```
" ".join(["hola", "qué tal"])
```

Borrado por valor:

```
amigos.remove('Mateo')
print(amigos)
```

Borrado por índice:

```
del amigos[0]
print(amigos)
```

Devuelve un elemento y lo borra de la lista:

```
valor = amigos.pop(0)
```

```
edades = [30, 40, 38, 30, 37]
```

Cantidad de apariciones:

```
print(edades.count(30))
```

Largo de la lista:

```
print(len(edades))
```

Ordenar la lista:

```
sorted(edades)
```

Sumar el total:

```
print(sum(edades))
```

Mínimo:

```
print(min(edades))
```

Máximo:

```
print(max(edades))
```

Ejercicio:

1- Calcular el promedio de edad de la lista "edades"

```
# Ejercicio 1
```

## 6.3. Diccionarios

Los diccionarios consisten en estructuras que contienen pares de una **llave** y un **valor**. Los elementos NO están ordenados, con lo cual no se puede acceder por posición ni slicing.

Veamos un ejemplo

```
dnis = {'Herrera':32676585, 'Guzmán':9564787,  
'Pérez':5676898, 'Hernández':40565999,  
'Abraham':28375814,  
"soy_una_llave":"soy_un_valor"}
```

Sin embargo, sí podemos acceder a un elemento por su llave. Accedemos al valor de "Abraham"

```
dnis['Abraham'] # noten que se usa la misma notación que con las listas
```

¿Qué pasa si tratamos un diccionario como una lista?

```
dnis[0]
```

Tenemos un error, tratemos de interpretarlo: KeyError se refiere a que no existe una llave (Key) a la que se trató de acceder. En este caso la llave que se trató de acceder es 0.

Podemos ver todas las llaves:

```
dnis.keys()
```

Traer todos los pares de elementos:

```
dnis.items()
```

Y utilizar los mismos métodos para borrar y extraer como en las listas:

```
dnis.pop('Herrera')
```

Los diccionarios tienen longitud, de la misma forma que las listas y string

```
len(dnis)
```

## 6.4. Condicionales

El condicional tiene la siguiente sintaxis:

```
if CONDICIÓN:  
    código1  
elif CONDICIÓN2:  
    código2  
else:  
    código3
```

Donde la condición es un operador que devuelve un objeto de tipo booleano. La **indentación** del código define qué parte se incluye como condicional.

El término “elif” viene de “else if”. La condición sólo se evaluará si la condición del “if” no se cumple.

```
precio_dolar = 62  
  
if precio_dolar >= 90:  
    print("El dólar se fue por las nubes")  
elif (precio_dolar < 90) and (precio_dolar >= 70):  
    print("El dólar subió")  
else:  
    print("El dólar es menor a 70")  
  
print(precio_dolar)
```

## 6.5. Bucles o Loops

Los bucles son un tipo de sentencia donde se realiza el código contenido repetidamente. Existen dos tipos. En el bucle **for**, se **itera** o recorre un conjunto de elementos actuando por cada uno de ellos. En el bucle **while** se itera hasta que se cumple una condición de corte.

### 6.5.1. For loop

```
for n in [1,2,'3']:  
    print(f'EL tipo es {n*2} {type(n*2)}')  
  
lista = [1,2,3,4,5]  
  
for n in lista:  
    print(n * 2)
```

También agreguemos la función **range** a nuestro repertorio. La función range consta de tres parámetros importantes: start, stop, step. Si pasamos un sólo parámetro, estamos pasando el “stop” y tomamos 0 como valor default de “start”. Del mismo modo que cuando vimos listas, el stop no está incluido y el step por default es 1.

Veamos cómo funciona:

```
for n in range(5):  
    print(n)
```

Si pasamos dos parámetros, estamos pasando el start y el stop, el primer valor es el “start” y el segundo el “stop”.

```
for n in range(0,5):  
    print(n)
```

Noten que el valor de start se incluye y el de stop no:

```
for n in range(1, 6): # esto nos da el mismo resultado que un par de celdas más arriba  
    print(n * 2)
```

Por último, el tercer parámetro es el step...

Ejercicios combinando lo visto hasta el momento:

1- Imprimir los valores de 1 a 50, saltando de a un valor por vez...

2- Dada la siguiente lista: medios = [“cheques”, “bonos”, “acciones”, 1000, “transferencia”]. Acceder al 2do elemento, y luego al 3er elemento (de ese 2do elemento) y responder qué devuelve.

3- De manera similar, ahora accedan al 4to elemento, y luego al 3er elemento de éste, ¿qué ocurre?

4- Agreguen una lista vacía al final de esa lista.

5- Recorran *medios* (iteren) y si el tipo del elemento es string, hagan un print de su primer elemento, si es int (entero) pásenlo a string y hagan un print de su primer elemento y si no es nada de eso hagan un print que diga "Es otro tipo". Tip1: para pasar un elemento a tipo string pueden usar la función str(variable), donde variable es el elemento que quieren transformar. Tip2: para arrancar este ejercicio pueden copiar lo siguiente:

```
for medio in medios:  
    pass # pass no hace nada, sólo pasa sin ejecutar nada... Reemplácenlo y  
completén con las condiciones.
```

### 6.5.2. While loop

Los bucles **while** se definen con una condición, y el código contenido se ejecuta mientras la misma evalúe como True. Es importante definir correctamente cuándo la condición pasa de True a False, si no lo hacemos podemos dejar corriendo un programa infinitamente sin que corte. Si eso llega a suceder tienen un botón de stop arriba en la notebook.

```
contador = 0  
  
while contador < 20:  
    contador += 1 # esto equivale a count = count + 1  
    print(contador)
```

## 6.6. Recursos y tips

- [Google!](#)
- [StackOverflow](#)
- [Google Colab](#)
- [Slicing](#)

## 6.7. Tips en la práctica

- Ver *docstring* con Shift + TAB o help()
- Leer Errores
- Print

 Open in Colab

Recordá abrir en una nueva pestaña

# 7. Ejercicios I

## 7.1. 1- Cambiar un texto

Necesitamos mostrar en nuestra web la sinopsis de las películas. El problema es que tenemos los textos separados por pipes ("|") en lugar de saltos de línea.

```
string = 'Sinopsis | Marty McFly, un típico adolescente americano de los años  
ochenta, '|  
'es accidentalmente enviado de vuelta a 1955 en una "máquina del tiempo" realizada  
con'|  
'un DeLorean inventada por un científico un poco loco. | En este viaje, Marty debe'|  
'\|'  
'asegurarse de que sus padres se encuentren y se enamoren, para que pueda volver a  
su tiempo. '|  
  
print(string)
```

¿Cómo podríamos hacer para que la función "print" muestre saltos de línea en lugar de pipes?

## 7.2. 2 - Crear un acrónimo

Ahora nos piden crear un acrónimo (una palabra compuesta por la primera letra de cada palabra) de cada título. Pero antes de eso es necesario que transformemos los títulos a "title case" porque no todos van a llegar proljos. Entonces, el acrónimo de "Volver al futuro" debería ser "VAF".

```
titulo = 'Volver al futuro'  
  
# 1. Transformar el string a "title case"  
  
# 2. Crear una lista con todas las palabras del string  
  
# 3. Recorrer la lista y agregar al acrónimo la primera letra de cada palabra  
acronimo = ''
```

 Open in Colab

Recordá abrir en una nueva pestaña

## 8. Listas, Funciones, Errores

- [Control de flujo](#)
- [Tuplas](#)
- [Errores](#)
- [List Comprehension](#)
- [Funciones](#)

### 8.1. Pequeño Repaso

```
for i in range(1, 10):  
    if i % 2 == 0:  
        print(f"El número {i} es par")
```

```
El número 2 es par  
El número 4 es par  
El número 6 es par  
El número 8 es par
```

## 9. Control de flujo

Contamos con 3 keywords que modifican el orden de ejecución **dentro de un bucle**

- **continue**: interrumpe el flujo del bucle y retoma la ejecución en la siguiente iteración
- **break**: termina el bucle
- **pass**: no tiene efecto, se usa para evitar error cuando lo exige la sintaxis

Vamos a simular una base de datos de usuarios empleando un diccionario. Este diccionario tiene como llave un id representando por un entero. Como valor tiene un diccionario con nombre, apellido y dni.

```
base = {  
    1: {"nombre": "Fernanda",  
         "apellido": "Fernandez",  
         "dni": 33333331},  
    2: {"nombre": "Gonzalo",  
         "apellido": "Gonzalez",  
         "dni": 33333332},  
    3: {"nombre": "Rodrigo",  
         "apellido": "Rodriguez",  
         "dni": 33333333}  
}
```

Supongamos que en una página web tenemos un formulario en el cual las personas que acceden completan con su dni y queremos saber el nombre y apellido si contamos con el mismo en la base, y si no lo tenemos sólo queremos saber cuál es el dni.

```
dnis = [333333331, 333333336, 333333339, 333333332, None, 333333333]

n_encontrados = 0

for dni in dnis: # itero por todos los dnis

    if type(dni) == int: # si el tipo es entero
        dni_encontrado = False # inicializo una variable con valor False, ya van a ver para qué :-)

        # Ahora viene la parte complicada, ¿cómo sé si ese dni ya está en mi base?
        # 1- Recordemos que base es un diccionario y como tal tiene el método
        # items(), pruébenlo afuera de esta celda
        # me devuelve una tupla, con la llave en el primer elemento y el valor en el segundo

        # itero por todos los elementos
        for i in base.items():
            valor = i[1] # guardo el valor en una variable
            if dni == valor["dni"]:
                dni_encontrado = True
                nombre_completo = valor["nombre"] + " " + valor["apellido"]
                n_encontrados += 1 # esto equivale a encontrados = encontrados +
                1, agrega uno
                break # freno la búsqueda, ésto evita que siga buscando

        if dni_encontrado: # entra acá si es True
            print(f'{nombre_completo.title()} ingreso a nuestra web')

        elif not dni_encontrado: # noten el not
            print(f'El dni {dni} no se encuentra en la base...')
            continue # sigo con la búsqueda, NO paso a la siguiente linea

        print(f'Hasta el momento se encontraron {n_encontrados} casos')

    else:
        pass # si dni no es un entero entonces no hacemos nada, suponemos que hubo
        # algún tipo de error
```

```
Fernanda Fernandez ingreso a nuestra web
Hasta el momento se encontraron 1 casos
El dni 333333336 no se encuentra en la base...
El dni 333333339 no se encuentra en la base...
Gonzalo Gonzalez ingreso a nuestra web
Hasta el momento se encontraron 2 casos
Rodrigo Rodriguez ingreso a nuestra web
Hasta el momento se encontraron 3 casos
```

## 10. Tuplas

Una **tupla** es una secuencia **inmutable** de longitud fija. Las tuplas generalmente se utilizan para representar secuencias de elementos que tienen longitud fija.

```
registro = ('Argentina', 112000, 'Bariloche')
```

Podemos acceder a un elemento por un índice como con una lista

```
registro[0]
```

```
'Argentina'
```

### 10.1. Unpacking

Con una tupla, o una lista podemos separar una secuencia en variables de la siguiente manera

```
país, población, ciudad = registro

print("País:", país)
print("Población:", población)
print("Ciudad:", ciudad)
```

```
País: Argentina
Población: 112000
Ciudad: Bariloche
```

```
registros = [registro, ('Brasil', 477000, 'Florianópolis')]
registros[1]
```

```
('Brasil', 477000, 'Florianópolis')
```

Pero como las tuplas son **inmutables** no se pueden modificar

```
# Una lista sí se puede modificar
lista = [1, 2, 3]

lista[0] = 4

lista
```

```
[4, 2, 3]
```

En cambio una tupla no se puede modificar o, como dice el cartel de error, no permite asignación:

```
print(registro)
registro[0] = 'Chile'
```

```
('Argentina', 112000, 'Bariloche')
```

```
-----  
TypeError                                                 Traceback (most recent call last)
<ipython-input-10-943f179ec8dc> in <module>
      1 print(registro)
----> 2 registro[0] = 'Chile'

TypeError: 'tuple' object does not support item assignment
```

```
# tampoco se puede borrar ninguno de sus elementos
del registro[0]
```

```
-----  
TypeError                                                 Traceback (most recent call last)
<ipython-input-11-de3b5b100261> in <module>
      1 # tampoco se puede borrar ninguno de sus elementos
----> 2 del registro[0]

TypeError: 'tuple' object doesn't support item deletion
```

## 11. Tipos de errores

En el ejemplo de arriba vimos una pantalla de error. El **traceback** o **stack trace** nos muestra el camino del error: veremos más adelante que cuando tenemos funciones anidadas, vemos aquí como el error se propaga.

Las cosas a las que en principio les debemos prestar atención son:

1. El nombre del error, en este caso **TypeError**
2. La explicación dada en el último renglón: “tuple’ object doesn’t support item deletion”
3. La flecha que nos indica la línea en la que ocurrió el error

Existen distintos tipos de errores en Python, pueden consultar la lista de todas las excepciones básicas acá:

<https://docs.python.org/3/library/exceptions.html#bltin-exceptions>

Sin embargo, las excepciones más frecuentes son:

**AttributeError**: cuando tratamos de llamar a una referencia que no existe.

**NameError**: cuando se llama a una variable u otro nombre que no está definido en el ambiente en que estamos trabajando.

**KeyError**: cuando queremos llamar a una llave que no existe, por ejemplo, en un diccionario.

**SyntaxError**: cuando hay un problema de sintaxis, errores muy comunes al comienzo pueden ser que no se cerró una llave o corchete, que falta alguna coma o dos puntos.

**TypeError**: cuando el tipo de soporta una determinada operación.

**IndexError**: cuando el índice al que se quiere acceder no existe, generalmente debido a que se llama a un índice mayor al largo de la lista.

Veamos algunos ejemplos:

```
print(no_definido)
```

```
NameError Traceback (most recent call last)
<ipython-input-12-7eac29aaea1e> in <module>
----> 1 print(no_definido)

NameError: name 'no_definido' is not defined
```

```
print(base) # recordamos base
```

```
{1: {'nombre': 'Fernanda', 'apellido': 'Fernandez', 'dni': 333333331}, 2:
{'nombre': 'Gonzalo', 'apellido': 'Gonzalez', 'dni': 333333332}, 3: {'nombre':
'Rodrigo', 'apellido': 'Rodriguez', 'dni': 333333333}}
```

```
base['Gertrudis']
```

```
KeyError Traceback (most recent call last)
<ipython-input-14-dfebd799e0> in <module>
----> 1 base['Gertrudis']

KeyError: 'Gertrudis'
```

## 11.1. Ejercicios

Prueben y arreglen los siguientes errores de código:

1. ¿Qué está pasando en este código?

```
if 10 > 5
    print('Qué está pasando?')
```

```
File "<ipython-input-15-c797e0e0635f>", line 1
  if 10 > 5
  ^
SyntaxError: invalid syntax
```

1. ¿Y en este? Traten de arreglarlo

```
a = 5
b = "3"
a + b == 8
```

```
-----  
TypeError                                                 Traceback (most recent call last)  
<ipython-input-16-715f0e881b0b> in <module>  
      1 a = 5  
      2 b = "3"  
----> 3 a + b == 8  
  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

1. ¿Y continuación?

```
a = ["the beatles", "the rolling stones", "queen", "led zepellin"]  
a[4]
```

```
-----  
IndexError                                              Traceback (most recent call last)  
<ipython-input-17-5956792cce49> in <module>  
      1 a = ["the beatles", "the rolling stones", "queen", "led zepellin"]  
      2  
----> 3 a[4]  
  
IndexError: list index out of range
```

4- ¿Y acá? ¿qué pasó?

```
a.appended(["cream"])
```

```
-----  
AttributeError                                           Traceback (most recent call last)  
<ipython-input-18-fb0e2f01d423> in <module>  
----> 1 a.appended(["cream"])  
  
AttributeError: 'list' object has no attribute 'appended'
```

## 12. Manejo de errores

Para anticipar y manejar los errores, podemos usar la cláusula **try** y **except**

```
a = '20'  
b = 5  
  
try:  
    print(a+b)  
except:  
    print(int(a)+int(b)) # convierto a int ambas variables
```

```
25
```

Si queremos ver qué excepción ocurrió, podemos usar la siguiente sintaxis:

```
try:  
    print(a+b)  
except Exception as e:  
    print(f"El error fue {e}")
```

```
El error fue can only concatenate str (not "int") to str
```

También podemos especificar qué hacer para distintos tipos de error:

```

lista_tuplas = [('3', 8),
                (5, 0),
                (3, ),
                (4, 6)]

for t in lista_tuplas:
    print(f"la tupla es {t}")

    try:
        print("intento dividir...")
        print(t[0] / t[1])
        print("éxito!")
    except IndexError:
        print('El largo está mal')
    except TypeError:
        print('Error en el tipo de datos')
    except ZeroDivisionError:
        print("No se puede dividir por cero")

```

```

la tupla es ('3', 8)
intento dividir...
Error en el tipo de datos
la tupla es (5, 0)
intento dividir...
No se puede dividir por cero
la tupla es (3, )
intento dividir...
El largo está mal
la tupla es (4, 6)
intento dividir...
0.6666666666666666
éxito!

```

## 13. Listas por comprensión (List Comprehension)

Las **listas por comprensión** son una funcionalidad muy flexible de Python que permite crear listas de un modo más “descriptivo”, basandose en la notación de definición de conjuntos.

Supongamos que necesitamos obtener una lista con los elementos de una lista original elevados al cuadrado.

Sin listas por comprensión haríamos...

```

lista = [1, 2, 3, 4, 5]
cuadrados = []
for x in lista:
    cuadrados.append(x**2)
cuadrados

```

```
[1, 4, 9, 16, 25]
```

En cambio, con listas por comprensión usamos esta expresión:

```
cuadrados = [x**2 for x in lista]
```

En las listas por comprensión también podemos incluir condicionales en una sola línea, vean la siguiente expresión:

```
x = 10
print(x if x > 15 else 0)
```

```
0
```

```
x = 16
print(x if x > 15 else 0)
```

```
16
```

Ahora vamos a generar una lista por comprensión sólo con los números donde el cuadrado sea menor a 15

```
cuadrados = [x**2 for x in lista if x**2 < 15]
cuadrados
```

```
[1, 4, 9]
```

La sintaxis para **filtrar** con condicionales es

```
[ (elemento) ( for x in (iterable) ) ( if condicion ) ]
```

Donde "elemento" es lo que vayamos a guardar en la lista. Incluyendo un **else**:

```
[ (elemento) (if condicion else otro_elemento) ( for x in (iterable) ) ]
```

Pueden hacerse loops anidados:

```
[i for i in range(x) for j in range(y)]
```

Otra forma de pensar la sintaxis es a partir de teoría de conjuntos. Por ejemplo: Un conjunto S definido por todos los números  $X / 4$  que pertenecen a los Naturales y cumplen que su cuadrado es menor a 20

$\{S = \{x / 4 \mid x \in \mathbb{N}, x^2 < 60\}\}$

Con un loop for:

```
S = []
for x in range(1000):
    if x ** 2 < 60:
        S.append(x/4)
S
```

```
[0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75]
```

Con listas por comprensión:

```
S = [x/4 for x in range(1000) if x**2 < 60]
S
```

```
[0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75]
```

```
for x in range(3):
    for y in 'abc':
        print(x,y)
```

```
0 a
0 b
0 c
1 a
1 b
1 c
2 a
2 b
2 c
```

```
[(x, y) for x in range(3) for y in 'abc']
```

```
[(0, 'a'),
 (0, 'b'),
 (0, 'c'),
 (1, 'a'),
 (1, 'b'),
 (1, 'c'),
 (2, 'a'),
 (2, 'b'),
 (2, 'c')]
```

O comprensiones anidadas:

```
[ [i for i in range(x)] for j in range(y) ]
```

```
[[l*n for l in 'abc'] for n in range(3)]
```

```
[[], [], []], [['a', 'b', 'c'], ['aa', 'bb', 'cc']]
```

Además, el elemento que se genera puede ser de otro tipo, en este caso una tupla:

```
animales = ['mantarraya', 'pandas', 'narval', 'unicornio']  
[(a, len(a)) for a in animales]
```

```
[('mantarraya', 10), ('pandas', 6), ('narval', 6), ('unicornio', 9)]
```

### 13.1. Ejercicios

1- Dado el siguiente grupo de palabras, crear otra lista que contenga sólo la primera letra de cada una

```
lista = ['a', 'ante', 'bajo', 'cabe', 'con']
```

2- Dada la siguiente frase, crear una lista con el largo de cada palabra. Tip: Antes de aplicar listas por comprensión pueden usar la función split que vimos la clase pasada y la función replace para remover la puntuación.

```
frase = 'Cambiar el mundo, amigo Sancho, no es ni utopía ni locura, es justicia'
```

## 14. I/O

Ahora veremos como abrir archivos usando Python base. Para hacerlo usaremos la sentencia **with** para definir un **contexto** del siguiente modo:

```
with open('corpus.txt', 'r') as inp:  
    string_contenido = inp.read()
```

Lo que significa: con el archivo “corpus.txt” abierto en **modo** lectura (“r” de read) con el **alias** *inp*, definimos la variable contenido usando el método **.read()** para leer el archivo. Algunas aclaraciones:

- El método **.read()** es propio del objeto de input, que **en esta ocasión** llamamos *inp*. Otro método útil es **.readlines()** que nos permite iterar por renglones.
- La ruta al archivo puede ser **relativa** como en este caso, donde el archivo se encontraría en la misma carpeta que la notebook. También se puede dar el path completo, como podría ser “C:/Usuarios/Matías/Documentos/corpus.txt”

Existen varios modos de abrir un archivo, incluyendo:

```
- r: read, lectura  
- w: write, escritura  
- a: append, agregar al final del archivo
```

Por ejemplo, para escribir en un archivo, haríamos:

```
with open(outpath, 'w') as f:  
    f.write(string)
```

```
with open('nuevo.txt', 'w') as f:  
    f.write('ejemplo de escritura')
```

```
with open('nuevo.txt', 'r') as f:  
    contenido = f.read()  
  
print(contenido)
```

## ejemplo de escritura

En Python se puede pedir un input del usuario de la siguiente manera:

```
while True:  
    usuario_dijo = input('Ingrese un numero')  
  
    try:  
        num = int(usuario_dijo)  
        break  
    except:  
        print('No anduvo, intente de nuevo')  
  
    print(f'Su numero fue {num}! :D')
```

```
Ingrese un numero  
No anduvo, intente de nuevo  
Ingrese un numero1  
Su numero fue 1! :D
```

## 15. Funciones

Las funciones nos permiten estandarizar y reutilizar un proceso en múltiples ocasiones.

Como patrón de diseño, las funciones tienen que ser lo más atómicas posibles, es decir, resolver un problema lo más pequeño posible y bien definido.

Los nombres también son importantes, el nombre de la función tiene que reflejar lo mejor posible lo que hace.

Las funciones se definen de la siguiente manera:

```
def nombre_de_funcion(argumentos):  
    """docstring opcional"""  
    resultado = procedimiento(argumentos)  
    return resultado
```

Las variables definidas en el contexto de una función son locales, es decir, sólo viven dentro de esa función. Por otra parte, si desde una función se llama a una variable que no está definida en la función tratará de buscarla afuera. En general, es buena práctica usar sólo variables definidas dentro de la función o argumentos.

Los argumentos son variables que le pasamos a la función para que *haga algo*.

Las funciones emplean dos términos reservados:

- def: que indica el comienzo de la definición de la función
- return: establece la variable o variables que devuelve la función

En este punto vale la pena mencionar que en Python los nombres (de variables, funciones, etc.) se almacenan y ordenan en NameSpaces. En este punto, la información definida dentro de la función no es compartida con el exterior de la función (salvo algunas excepciones que no vienen al caso), por lo tanto, para compartir la, la función debe enviar desde el NameSpace local (el de la función) al NameSpace global (donde está el resto del programa). Para ello, el término return indica qué se devolverá al ambiente global. Para que efectivamente esa/s variable/s que la función devuelve se guarden una variable en el ambiente global debemos asignar el resultado a una variable. Esto va a quedar más claro con el código.

Por último, vale la pena mencionar que print sólo nos muestra el contenido de una variable pero no envía la información desde la función hacia afuera.

```
def promedio(lista):  
    return sum(lista) / len(lista)  
  
resultado = promedio([0,1,2,3]) # asigno el resultado  
print(resultado)
```

1.5

```
def suma(a,b):  
    """Esta función recibe dos numeros y devuelve la suma"""  
    return a+b
```

```
r = suma(3,5) # asigno el resultado  
print(r)
```

```
8
```

```
def rango(lista):  
    return max(lista) - min(lista)
```

```
lista = [89, -24, 9, 2]  
rango(lista)
```

```
113
```

## 15.1. Ejercicios:

1- Escribir una función que reciba una lista con números y devuelva la suma de todos los pares

2- Escribir una función que tome una lista con distintos elementos (de cualquier tipo) y devuelva una lista sólo con los últimos tres números que aparecen.

En las funciones existen argumentos se pueden pasar por posición o por su nombre. Algunos de los argumentos tienen un valor por default.

```
def unir_lista(lista, conector=' '):  
    """Esta función recibe dos argumentos, una lista y un conector  
    y devuelve la lista unida usando el conector."""  
  
    unida = conector.join([str(e) for e in lista])  
  
    return unida
```

```
unir_lista(['unir',3,'cosas'])
```

```
'unir 3 cosas'
```

```
unir_lista(['probando', 'unir', 'lista', 123], conector = ',')
```

```
'probando,unir,lista,123'
```

Lo que no podemos hacer es pasar el argumento nombrado antes de el o los posicionales

```
unir_lista(conector = ',',['probando', 'unir', 'lista', 123])
```

```
File "<ipython-input-47-81791f827422>", line 1  
    unir_lista(conector = ',',['probando', 'unir', 'lista', 123])  
          ^  
SyntaxError: positional argument follows keyword argument
```

Cuando uso los argumentos por su nombre, puedo pasarlo en cualquier orden

```
unir_lista(conector = ',',lista = ['probando', 'unir', 'lista', 123])
```

```
'probando,unir,lista,123'
```

Existen distintos tipos de argumentos, principalmente los llamados "args" y "kwargs", o argumentos y "keyword arguments" es decir argumentos nombrados.

También podemos tener un número variable de argumentos, usando el símbolo \*, por convención al parámetro se le llama 'args'

Internamente, los elementos de \*args se tratan como una tupla

```
def suma_todos(*args):
    print(type(args))
    return sum(args)
```

```
suma_todos(9,1,4,2)
```

```
<class 'tuple'>
```

```
16
```

En cambio los elementos de \*\*kwargs se tratan como un diccionario

```
def consulta(**kwargs):
    print(type(kwargs))
    texto = f"Su consulta es {kwargs['fecha']} a las {kwargs['hora']}"
    return texto
```

```
consulta(fecha='hoy', hora='4PM')
```

```
<class 'dict'>
```

```
'Su consulta es hoy a las 4PM'
```

Finalmente, podemos juntar todo lo visto en una misma función, es decir, tener argumentos sin nombre sin valor default, argumentos con valores default, args y kwargs. Esto es especialmente útil cuando necesitamos enviar args o kwargs a otras funciones usadas dentro de una función.

```
valor_consulta = {"consulta":10, "arreglos_caries":20, "flúor":15}
```

```
costos = list(valor_consulta.values())
costos
```

```
[10, 20, 15]
```

```
def emitir_factura(nombre, dni, tipo="DNI", *args, **kwargs):
    factura = ""
    factura += f"Gracias por su visita Nombre:{nombre} \n"
    factura += f"{tipo} {dni} \n"
    costo = suma_todos(*args) # hago unpacking de args
    factura += f"El costo es {costo} \n"
    factura += consulta(**kwargs) # hago unpacking de kwargs
    return factura
```

```
factura = emitir_factura("Obi Wan", 370310455, "DNI", *costos, fecha="hoy",
hora="5PM")
```

```
<class 'tuple'>
<class 'dict'>
```

```
print(factura)
```

```
Gracias por su visita Nombre:Obi Wan
DNI 370310455
El costo es 45
Su consulta es hoy a las 5PM
```

 Open in Colab

Recordá abrir en una nueva pestaña

## 16. Ejercicios II

## 16.1. El contador de palabras

Una revista científica quiere publicar los abstracts de los trabajos que aprobó recientemente pero primero tiene que asegurarse de que ninguno de los abstracts tenga más de 200 palabras.

Para interactuar con los archivos que tenemos en nuestro “file system” vamos a utilizar el módulo os. No se preoculen por entender todos los detalles ahora, vamos a ir profundizando en la utilización de módulos.

```
!wget https://datasets-humai.s3.amazonaws.com/datasets/publicaciones.zip
```

```
!unzip publicaciones.zip
```

```
import os
```

```
archivos_directorio = os.listdir('publicaciones')
```

```
print(archivos_directorio)
```

La función listdir nos devuelve una lista con todos los archivos que están en la carpeta publicaciones. Noten que solamente nos devuelve los nombres de los archivos, no la ruta completa que necesitamos para acceder a los mismos desde la ubicación en el filesystem donde se encuentra esta notebook.

Las rutas hasta los archivos cambian con el sistema operativo, por eso si están en Windows, la forma de acceder al archivo Yukon Delta Salmon Management.txt es ejercicios\Yukon Delta Salmon Management.txt mientras que si están en Linux o Unix la forma de acceder es ejercicios/Yukon Delta Salmon Management.txt . Para evitar problemas y que el código sea ejecutable desde cualquier sistema operativo, el módulo os tiene la función os.join.

Entonces para crear las rutas vamos a usar la función os.path.join y para esto es ideal una lista por comprensión

```
rutas_archivos = [os.path.join('publicaciones',archivo) for archivo in archivos_directorio]
```

```
rutas_archivos
```

Ahora vamos a unir estas dos listas del mismo tamaño en una lista de tuplas utilizando la función “zip” de Python nativo. Como el zip de Python devuelve un objeto iterable, vamos a convertirlo en lista para trabajar mejor

```
tuplas_archivos = list(zip(rutas_archivos,archivos_directorio))
```

```
for tupla in tuplas_archivos:  
    print(tupla)
```

Ahora sí, vamos a pedirles que crean una función que reciba una tupla con la ruta y el nombre del archivo. Necesitamos que esta función cuente las palabras que hay en el txt que se encuentra en esa ruta y luego imprima el nombre del archivo y la cantidad. Después vamos a escribir un for loop que recorra la lista tuplas\_archivos y devuelva una tupla con el nombre del archivo y la cantidad de palabras. Desde el loop for vamos a imprimir esa tupla.

```
# 1. Escribir la función
```

```
# 2. Recorrer en un loop tuplas_archivos invocando a la función
```

Entonces ¿Cuáles superan las 250 palabras? Si quieren ir una milla extra modifiquen la función para que devuelva True si supera y False si no supera en lugar de devolver la cantidad.

```
# 3. Modifiquen la función
```

```
#4. Vuelvan a llamarla
```

## 17. Ejercicios Adicionales II

Estos ejercicios tienen un nivel de dificultad un poco mas elevado. Cada ejercicio tiene una función de test para chequear si lo que hicieron esta bien.

```
!wget https://datasets-humai.s3.amazonaws.com/datasets/test_intro_clase2.zip  
!unzip test_intro_clase2.zip
```

```
from test import *
```

### 17.1. Juego de espías (Fácil)

El espía Ramsay debe codificar los mensajes que le mandan otros espías sobre la cantidad de tropas que tiene el enemigo en distintos cuarteles. Para esto, otro espía le manda una tira de números con un pequeño truco. Esta tira de números estan separados por -, pero para que no sea tan fácil saber que esta informando, la cantidad de tropas esta levemente escondida y tambien esta escondido el número del cuartel. El cuartel estará escondido en el último lugar de la tira y para obtener la cantidad de tropas aproximadas se deben sumar todos los números que son divisibles por el número del cuartel de la tira. Crear una función que reciba el string de la tira de números y devuelva la cantidad de tropas que hay en el cuartel enemigo como una tupla. Adicionalmente, podria imprimir un mensaje con la información requerida.

Ej:

```
INPUT:  
tira_numeros = '29-32-1-5-65-12345-0-12-2'  
OUTPUT:  
(2, 44)  
"En el cuartel número 2 hay 44 soldados"
```

```
# Corre para ver si tu código esta bien  
def informe_espia(tira_numeros):  
    # Hacer la magia  
  
    return (cuartel,informe)
```

```
test1(informe_espia)
```

### 17.2. Codificador César (Intermedio)

Una de las formas mas antiguas de crear un código encriptado es lo que se conoce como el encriptado César

[https://es.wikipedia.org/wiki/Cifrado\\_C%C3%A9sar](https://es.wikipedia.org/wiki/Cifrado_C%C3%A9sar). En este tipo de encriptado lo que se hace es "girar" el abecedario una determinada cantidad de pasos según una clave numérica (ver ejemplo). Crear una función que lea un string dentro de un txt en la misma ruta que esta notebook, tome una clave y devuelva el string encriptado con la clave César en minúsculas(asumir que el texto esta en castellano).

Ej: Clave = 2

Letra	Letra encriptada
A	C
B	D
C	E
...	...
Y	A
Z	B

```
INPUT:  
'mi_archivo.txt' ("Hola estudiante"), clave = 1  
OUTPUT:  
"Jqnc guvwfkcovg"
```

AYUDA

El método `mi_lista.index(elemento)` búsca el `elemento` en la lista `mi_lista` y devuelve la posición del elemento si lo encontró. Si no lo encontró devuelve un `ValueError`.

```
def codificador_cesar(mensaje_path, clave):
    # Un ayudin
    abecedario = 'abcdefghijklmnñopqrstuvwxyz'
    # Ahora hagan su magia (ojo con las mayúsculas)

    return cifrado
```

```
# Corre para ver si tu código esta bien
test2(codificador_cesar)
```

## OPCIONALES

- Hacer que la función guarde la salida como un archivo de texto.
- Podemos encriptar respetando mayúsculas. Adaptar la función para que lo haga. Se puede usar la función test2\_mayusculas para probarla.  
Sugerencia: Mirar el método `isupper()` para los strings.
- Adaptar la función anterior pero para desencriptar, una función que cree una lista con todas las posibles rotaciones del texto.

### 17.3. La calesita (Rompecoco)

El señor Jacinto es dueño de una antigua calesita con animalitos que no funciona hace varios años y quiere volver a ponerla en funcionamiento. Para eso va a probarla prenriendola y viendo cuanto rota segun la cantidad de movimientos.

Crear una función que reciba una lista de strings (con la primera en mayúscula) con los animales que componen la calesita, una cantidad de ciclos(`n_ciclos`) y devuelva la misma lista pero rotada hacia la derecha esa cantidad de movimientos, donde un movimiento es cambiar todos los animales una posición hacia la derecha:

Ej:

```
INPUT:
['Unicornio', 'Oso', 'Jirafa', 'Pato', 'Elefante'], movimiento = 1
OUTPUT:
['Elefante', 'Unicornio', 'Oso', 'Jirafa', 'Pato']
```

```
def probar_calesita(calesita, n_movimientos):
    # Proba la calesita

    return calesita_girada
```

```
# Corre para ver si tu código esta bien
test3(probar_calesita)
```

Cuando prueba la calesita se da cuenta que es muy lenta. Debe sacar uno de los animales para que pueda funcionar correctamente. Para eso los manda a pesar y le dicen cual es el que hay que sacar para que funcione perfectamente.

Modificar la función anterior para que reciba un string, que es un animal en MAYÚSCULAS (`animal_quitar`) para sacar y pruebe la función nuevamente.

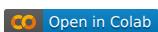
Ej:

```
INPUT:
['Unicornio', 'Oso', 'Jirafa', 'Pato', 'Elefante'], animal_quitar = 'JIRAF', movimientos = 1
OUTPUT:
['Elefante', 'Unicornio', 'Oso', 'Pato']
```

```
def probar_calesita_arreglada(calesita, n_mov, animal_quitar):
    # Arregla la calesita

    return calesita_arreglada
```

```
# Corre para ver si tu código esta bien
test4(probar_calesita_arreglada)
```



# 18. Programación orientada a objetos

- I. Terminología de clases y objetos
- II. Creando la primera clase
  - I. Herencia
  - II. Protección de acceso
- III. Métodos Especiales
- IV. Métodos Estáticos
- V. Duck typing y monkey patching
  - I. Duck typing
  - II. Monkey Patching

En el paradigma de programación orientada a objetos los programas se estructuran organizando el código en entidades llamadas objetos. Estos nos permiten encapsular data, funciones y variables dentro de una misma clase. Veamos de qué se trata.

## 18.1. Terminología de clases y objetos

1. Una **clase** es un prototipo de objeto, que engloba atributos que poseen todos los objetos de esa clase. Los atributos pueden ser datos como variables de clase y de instancia, y métodos (funciones). Se acceden con un punto.
2. Una **instancia** es un objeto en particular que pertenece a una clase.
3. Los **atributos** son variables asociadas a los objetos. Los atributos pueden ser de clase (toman el mismo valor para toda la clase) o de instancia (toman un valor diferente para cada instancia). Lo más común de utilizar son los atributos de instancia que reflejan estados de los objetos. Los atributos de clase se utilizan para que todos los objetos compartan por ejemplo, las mismas configuraciones o la misma conexión a una base de datos.
4. La **herencia** es la transferencia de atributos de una clase a otra clase
5. Un **método** es una función contenida dentro de un objeto.
6. Un **objeto** es una instancia única de una estructura definida por su clase. Posee de atributos variables de clase, de instancia y métodos.

## 18.2. Creando la primera clase

```
import math

#La sintaxis es:
class Ejemplo:
    pass

# Instancio la clase
x = Ejemplo()

print(type(x))
```

Por convención, las clases se nombran empleando “upper camel case”. Es decir, con mayúscula para cada término que sea parte del nombre.

Una librería famosa en Python por sus clases es “requests”. Esta librería se usa para acceder a información web por HTTP. Algunas de sus clases son:

- Session
- Request
- ConnectionError
- ConnectTimeout

Las últimas dos clases son para especificar errores, noten que se repiten las mayúsculas.

Podemos pensar a una clase como un molde, el cual usamos para generar objetos o instancias que tienen ciertos atributos o métodos (funciones) que deseamos mantener.

Aquellos atributos y métodos que queremos que los objetos conserven son definidos como parte del constructor. El constructor en Python es el método reservado `__init__()`. Este método se llama cuando se instancia la clase y en ese momento se inicializan los atributos de la clase, para lo cual podemos pasar parámetros.

Además, vamos a emplear el término reservado `self` para indicar aquellos atributos y métodos que van a ser propios de la instancia. Veámoslo con un ejemplo.

### 18.3. Init y Self

```
class Persona():
    def __init__(self, nombre, apellido, edad, contacto):
        # Este método puede tomar parámetros que asignamos a los atributos, que
        # luego podemos acceder
        self.edad = edad # este es un atributo
        self.contacto = contacto # este es otro atributo
        self.nombre = nombre
        self.apellido = apellido

    def nombre_completo(self):
        # este método muestra el nombre completo a partir del nombre y apellido
        nombre_completo = ', '.join([self.apellido, self.nombre])
        return nombre_completo

    def saludar(self):
        print(f'Hola mi nombre es {self.nombre_completo()},'
              f'y te dejo mi mail por si necesitás algo: {self.contacto}'')
```

```
instancia_ejemplo = Persona('Matías Andrés', 'Ripley', 24, 'mati@rip.com')
instancia_ejemplo.saludar()
```

```
Hola mi nombre es Ripley, Matías Andrés y te dejo mi mail por si necesitás algo:
mati@rip.com
```

Ahora veamos una clase menú que administra los platos y los precios

```
class Menu():
    def __init__(self, items):
        self.items = items

    def precio(self, lista_items):
        precio = 0
        for nombre_item in lista_items:
            precio = precio + self.items[nombre_item]
        return precio

    def tamaño(self):
        return len(self.items)
```

```
mi_menu = Menu({'latte':25, 'medialuna':15})
```

¿Cuánto salen un latte y dos medialunas? ¿Cuántos ítems tenemos?

```
mi_menu.precio(['latte', 'medialuna', 'medialuna'])
```

```
55
```

```
mi_menu.tamaño()
```

Ejercicio: Vamos a mejorar la clase anterior... En lugar de que el método precio reciba una lista de strings, hagamos que reciba una lista de diccionarios cada uno con dos claves nombre y cantidad que queríamos ordenar ¿Cuántos cuestan 10 lattes y 30 medialunas?

Los atributos también son conocidos como variables de instancia, en contraposición a las variables de clase. Las variables de instancia toman un valor específico a una instancia en particular (por eso se emplea el término **self**), por su parte, las variables de clase tienen un valor común para todas las instancias de una clase. Por convención las variables de clase se definen antes del constructor y no llevan **self** en su definición pero sí cuando se la quiere llamar.

```

class Curso:
    max_alumnos = 35 # definimos variable de clase

    def __init__(self, nombre, duracion, alumnos = None, costo=10):
        self.nombre = nombre
        self.duracion = duracion
        if alumnos is None:
            self.alumnos = []
        else:
            self.alumnos = alumnos
        self.costo = costo # costo tiene un valor por default
    """Por qué ese if? Las variables por default sólo se evalúan a la hora de
ejecutar la sentencia def.
En nuestro caso necesitamos que self.alumnos sea una lista y las listas
son objetos mutables.
Esto quiere decir que podemos modificarla sin volver a asignarla. Si en
vez de 'alumnos = None' usáramos
alumnos = [], entonces con cada nueva instancia del objeto estaríamos
compartiendo los alumnos.
Para evitar eso, en general la forma pythónica de hacerlo es usando None
por default y asignando el valor
deseado dentro de la función y no en el 'def' """
    def inscribir_alumno(self, nombre):
        self.alumnos.append(nombre) # para poder llamar a alumnos tengo que usar
        self.
        print(f'Se agregó al alumno/a {nombre}')

    def tomar_lista(self):
        for a in self.alumnos:
            print(f'Alumno: {a}')

    def resumen(self):
        print(f'Curso {self.nombre}, {self.duracion} clases pensadas para
{len(self.alumnos)} alumnos\n'
              f'Por el muy módico precio de {self.costo} rupias.',
              # llamo variable de clase:
              f'La ocupación actual es del
{round(len(self.alumnos)/self.max_alumnos,2)*100}%')

```

```
curso_python = Curso('Python', 6)
```

```
curso_python.alumnos
```

```
[]
```

```
# Llamamos métodos de la instancia
curso_python.inscribir_alumno('Diótima')
curso_python.inscribir_alumno('Aritófanés')
```

```
Se agregó al alumno/a Diótima
Se agregó al alumno/a Aritófanés
```

```
curso_python.tomar_lista()
```

```
Alumno: Diótima
Alumno: Aritófanés
```

```
curso_python.resumen()
```

```
Curso Python, 6 clases pensadas para 2 alumnos
Por el muy módico precio de 10 rupias. La ocupación actual es del 6.0%
```

```
curso_ml = Curso('Machine Learning', 8)
```

```
curso_ml.alumnos # vean que el curso está vacío!
```

```
curso_ml = Curso('Machine Learning', 8)
curso_ml.inscribir_alumno('Agatón')
curso_ml.inscribir_alumno('Erixímaco')
curso_ml.inscribir_alumno('Sócrates')
```

```
curso_ml.resumen()
```

```
curso_ml.alumnos
```

Ejercicios:

1- Defina una clase Punto que tome como parámetros x e y (las coordenadas) y constante que se puede instanciar correctamente.

2- En Python existen los llamados métodos mágicos (magic methods) o dunder (Double Underscores). Estos métodos se caracterizan, justamente, por comenzar y terminar con “\_\_”. Uno de los más comunes es el que permite darle estilo a la función **print**. Para que nuestro objeto entonces tenga un lindo print tenemos que definir una función “**\_\_str\_\_**” que sólo toma “self” como parámetro y que torne un string. Eso que retorna es el string que queremos que muestra cuando hagamos “print” del objeto. Dicho ésto, te invitamos a que lo intentes de la siguiente manera:

a. Definí una función “**\_\_str\_\_**” que sólo toma self como parámetro.

b. La función debe retornar el string que querés mostrar, recordá que podés usar los valores de “x” y de “y”

### 18.3.1. Herencia

La herencia se emplea cuando queremos que una clase tome los atributos y características de otra clase. En este caso, la clase derivada (Alumno) **hereda** atributos y métodos de la clase base (Persona). Para acceder a los métodos de la clase previa vamos a emplear el método reservado **super()**. Con este método podemos invocar el constructor y así acceder a los atributos de esa clase.

```
# Clase derivada
class Alumno(Persona):
    def __init__(self, curso, *args):
        """
            Alumno pertenece a un Curso (una instancia de la clase Curso) y, además,
            tiene otros atributos que pasaremos
            a la clase previa
        """
        self.curso = curso
        super().__init__(*args) # inicializamos la clase 'madre'. La llamamos
        usando super() y ejecutamos el constructor
        # Nótese también que desempacamos args

    def saludar(self): # Sobrecarga de métodos, ver abajo
        super().saludar() # ejecutamos el método de Persona .saludar() y agregamos
        más cosas a este método
        print('Estoy cursando:')
        self.curso.resumen()

    def estudiar(self, dato): # También podemos definir nuevos métodos
        self.conocimiento = dato
```

La clase Persona cuenta con un método **saludar()** y para Alumno también definimos un método **saludar()**. Cuando instanciemos un Alumno y ejecutemos el método **saludar()** lo que va a ejecutarse es el método **saludar()** de Alumno, no de Persona. Esto no quita que el método **saludar()** de Alumno llame al de Persona. Además, vale la pena mencionar que los dos tienen los mismos parámetros (ninguno en este caso). Este patrón de diseño es lo que se llama sobrecarga de métodos o overriding.

```
scott = Alumno(curso_python, 'Scott', 'Henderson', 49, 'sh@mail.com')
scott.saludar()
```

```
scott.estudiar('Se puede heredar de otra clase y extender sus métodos')
```

```
scott.conocimiento
```

Ejercicio:

1- Listar cuáles son los atributos y los métodos de scott y especificar cuáles provienen de Persona y cuáles están definidos por ser Alumno.

### 18.3.2. Protección de acceso

Podemos cambiar el acceso (publico, no publico, protejido) de los métodos y variables.

Dos formas distintas de encapsulamiento:

- \_nópublico
- \_protegido

Los atributos o método no públicos pueden ser accedidos desde el objeto y llevan el prefijo “\_”. La utilidad de este es indicarle al usuario que es una variable o método privado, de uso interno en el código de la clase y que no está pensando que sea usado desde afuera, por el usuario.

Por otra parte, en el caso de usar como prefijo “\_\_” (doble “\_”) directamente vamos a ocultar la variable o método de la lista de sugerencias para el usuario y tampoco va a poder invocarlo desde el objeto. Por este motivo, decimos que el atributo o método está protegido.

```
class Auto():

    def __init__(self, color, marca, velocidad_maxima):
        self.color = color
        self.marca = marca
        self.__velocidad_maxima = 200
        self.velocidad = 0
        self.__contador = 0 # kilómetros recorridos

    def avanzar(self, horas=1, velocidad=10):
        if self.__chequear_velocidad(velocidad):
            self.velocidad = velocidad
            print(f'avanzando durante {horas} horas')
            self.__contador += horas*self.velocidad
        else:
            print(f'Tu auto no puede llegar a tanta velocidad, el máximo es {self.__velocidad_maxima}')

    def __chequear_velocidad(self, velocidad):
        es_valida = False
        if velocidad < self.__velocidad_maxima:
            es_valida = True
            if self.velocidad < velocidad:
                print("Vas a acelerar!")
            else:
                print("Vas a desacelerar!")
        else:
            print("Tu motor no permite ir tan rápido")
            es_valida = False
        return es_valida

    def status(self):
        print(f'Vas a una velocidad de {self.velocidad} y llevás {self.__contador} km. recorridos')
```

```
superauto = Auto('rojo', 'Ferraudi', 200)
```

```
# Atributo no publ
superauto.avanzar(10)
```

```
superauto.status()
```

```
# No se puede acceder a un atributo protegido
superauto.__contador
```

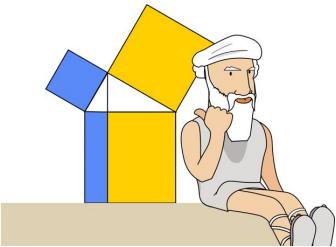
```
# Pero sí se puede acceder a un método no público:
superauto.__chequear_velocidad(10)
```

Ejercicio:

A continuación se define una clase Linea. Esta clase toma como parámetros dos objetos Punto() (instancias de la clase que definieron antes).

1- Agregar un método ‘largo’ que permita calcular el largo de la línea. Para ello vale la pena recordar que ésta se puede calcular como la hipotenusa del triángulo rectángulo que se forma con los dos puntos.

$\sqrt{a^2 + b^2}$



2- Agregar un método 'pendiente' que permita calcular la pendiente de la línea. Recordar que ésta se puede calcular como el cociente entre las diferencias de 'y' y de 'x'.

La fórmula es :  $m = (y_2 - y_1)/(x_2 - x_1)$

## 18.4. Métodos Especiales

Las clases en Python cuentan con múltiples métodos especiales, los cuales se encuentran entre dobles guiones bajos () .

Los métodos especiales más utilizados son `__init__()`, `__str__()` y `__del__()` .

`__init__` sirve para inicializar la clase y `__del__` sirve para eliminar completamente el objeto del compilador.

Veamos un ejemplo de uso de `__str__`. Una vez que definimos este método, responde a la sintaxis reservada de Python `str()`.

```
# Clase derivada
class Alumno(Persona):
    def __init__(self, curso, *args):
        """
        Alumno pertenece a un Curso (una instancia de la clase Curso) y, además,
        tiene otros atributos que pasaremos
        a la clase previa
        """
        self.curso = curso
        super().__init__(*args) # inicializamos la clase 'madre'. La llamamos
        usando super() y ejecutamos el constructor
        # Nótese también que desempacamos args

    def saludar(self): # Sobrecarga de métodos, ver abajo
        super().saludar() # ejecutamos el método de Persona .saludar() y agregamos
        más cosas a este método
        print('Estoy cursando:')
        self.curso.resumen()

    def estudiar(self, dato): # También podemos definir nuevos métodos
        self.conocimiento = dato

    def __str__(self):
        #Devuelve un string representativo del alumno
        return f'Alumno {self.nombre_completo()}'
```

```
un_alumno = Alumno(curso_python, 'Scott', 'Henderson', 49, 'sh@mail.com')
```

```
str(un_alumno)
```

## 18.5. Métodos Estáticos

¿Qué pasa si no queremos instanciar los objetos a la hora de usarlos? En algunos diseños, tiene sentido utilizar las clases como simples repositorios de métodos. Por ejemplo, si necesito resolver varias operaciones geométricas puedo crear una clase Geometria que contenga todos los métodos necesarios.

Para crear este tipo de métodos en una clase utilizamos el decorador `@staticmethod`.

```

class Geomtria():
    """Resuelve operaciones geométricas"""

    @staticmethod
    def pendiente(x1,y1,x2,y2):
        return ((y2 - y1)/(x2-x1))

    @staticmethod
    def area_circulo(radio):
        return math.pi * (radio**2)

```

```
Geomtria.area_circulo(3)
```

## 18.6. Duck typing y monkey patching

Dos características de la programación orientada a objetos con Python son el duck typing y el monkey patching. Este tipo de flexibilidad es el que le permitió a Python crecer tanto en su adopción porque reducen la cantidad de palabras que es necesario escribir para desarrollar código, lo cual ahorra tiempo y también disminuyen la complejidad.

### 18.6.1. Duck typing

```

class ElHobbit:

    def __init__(self,nombre):
        self.nombre = nombre

    def __len__(self):
        return 95022

    def saludar(self):
        return f'Hola soy {self.nombre}'

el_hobbit = ElHobbit('Frodo')

```

```
len(el_hobbit)
```

```

mi_str = "Hello World"
mi_list = [34, 54, 65, 78]
mi_dict = {"a": 123, "b": 456, "c": 789}

```

*"If it walks like a duck, and it quacks like a duck, then it must be a duck."*

Duck typing significa que a diferencia de otros lenguajes, las funciones especiales no están definidas para una lista específica de clases y tipos, si no que se pueden usar para cualquier objeto que las implemente. Esto no es así para la mayoría de los lenguajes.

```

len(mi_str)
len(mi_list)
len(mi_dict)
len(el_hobbit)

```

```

mi_int = 7
mi_float = 42.3

```

```

len(mi_int)
len(mi_float)

```

## 18.7. Monkey Patching

Guerrilla, gorilla, ¿monkey?... Este término viene de uno anterior, "guerrilla patching", que hace referencia a emparchar el código rápido y cuando es necesario.

Se refiere a la posibilidad en Python de sobreescibir clases después de haberlas instanciado y por qué no también la funcionalidad de los módulos.

```
el_hobbit.saludar()
```

```
def saludo_largo(self):
    return f'Hola mi nombre es {self.nombre}'
```

```
ElHobbit.saludar = saludo_largo
```

```
el_hobbit.saludar()
```

Esto es especialmente útil cuando queremos sobre-escribir ligeramente módulos hechos por terceros (¡o por nosotros mismos en otro momento!)

```
import math
```

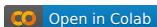
```
math.pi
```

```
math.pi = 2
```

```
math.pi
```

Los cambios se sobre-escriben si REINICIAMOS EL KERNEL y volvemos a importar el módulo

```
import math
math.pi
```

 Open in Colab

Recordá abrir en una nueva pestaña

## 19. Ejercicios POO IV

### 19.1. Programación orientada a objetos

En este ejercicio vamos a programar un carrito de compras.

Para esto vamos a escribir dos clases: Carrito e Item.

El ítem va a tener como propiedades un nombre, un precio y una url de la imagen que lo representa. Por default, la url se inicializa en blanco.

El Carrito tiene como propiedad una lista de diccionarios, la variable \_lineas.

Los carritos se inicializan vacíos y luego se agregan líneas utilizando el método agregar\_línea(). Cada línea es un diccionario con dos claves: "ítem" que contiene un objeto de tipo ítem y "cantidad" según la cantidad que queremos agregar al carrito.

Por último los carritos tienen un método get\_total() que devuelve la suma de los precios de los ítems, multiplicados por las cantidades que hay en cada línea.

```
# Clase Item
```

```
# Crear los ítems banana de $49.5 y yoghurt de $32.5
```

```
# Crear la clase Carrito
```

Ahora vamos a instanciar el carrito y agregarle una "línea" con dos bananas y otra con tres yoghures.

```
# Instancias el carrito
```

```
# Agregar bananas
```

```
# Agregar yoghures
```

```
# obtener el total
```

 Open in Colab

Recordá abrir en una nueva pestaña

## 20. Aplicación de POO: Experimento de Tiempo de Reacción

En esta clase vamos a utilizar un diseño de POO para crear una batería de experimentos:

- Tiempo de Reacción
- Cuestionario

Para eso, tendremos una clase **Sujeto** que contendrá un id de sujeto, una clase abstracta **Experimento** de la cual heredarán las de **Tiempo de Reacción** y **Cuestionario**, y una clase para generar un **Reporte** de los resultados automáticamente.

```
!pip install python-docx
```

```
Requirement already satisfied: python-docx in
/home/matias/anaconda3/lib/python3.7/site-packages (0.8.10)
Requirement already satisfied: lxml>=2.3.2 in
/home/matias/anaconda3/lib/python3.7/site-packages (from python-docx) (4.5.0)
```

Empezamos creando una clase para definir el *sujeto* del experimento.

```
class Sujeto():
    '''Clase para un Sujeto experimental.
    info contiene datos por ejemplo sociodemográficos previos
    en resultados guardaremos los resultados de las pruebas'''
    def __init__(self, sujeto_id, info: dict):
        self.id = sujeto_id
        self.info = info
        self.resultados = dict()

    def __repr__(self):
        # Función mágica para mostrar los datos del sujeto
        # el :<20 hace un relleno de máximo 20 espacios
        s = f'Datos de ID {self.id}<20>\n\n'
        for k, v in self.info.items():
            s += f'{k:<20} {v:<20}\n'
        for k, v in self.resultados.items():
            s += f'{k:<20} {v:<20}\n'
        return s
```

Veamos como funciona nuestra clase.

```
# Instanciamos
info = {'edad':22, 'fecha_nacimiento':'01-01-1998', 'años_educacion':6}
sujeto = Sujeto('123456789', info)
```

```
print(sujeto)
```

```
Datos de ID 123456789
edad          22
fecha_nacimiento 01-01-1998
años_educacion   6
```

Una vez que ya contamos con sujeto. Definimos una clase para la realización del experimento.

La clase tendrá tres métodos:

- instrucion: dará una pauta sobre como realizar el experimento.
- corregir\_datos: toma los datos crudos y calcula un resultado
- tomar\_experimento: realizará el experimento.

```

class Experimento():
    def __init__(self, sujeto: Sujeto, consigna='atencion!', nombre='Experimento'):
        self.nombre = nombre
        self.sujeto = sujeto
        self.consigna = consigna
        self.datos = []

    def instrucion(self):
        print(self.consigna)

    def corregir_datos(self):
        pass

    def tomar_experimento(self):
        pass

```

```
exp = Experimento(sujeto)
```

```
exp.nombre
```

```
'Experimento'
```

## 20.1. Tiempo de Reacción

Para tomar el experimento se utilizarán dos modulos/librerías: **time** y **random**.

- **time**, nos va a ayudar a calcular el tiempo durante nuestro experimento.
- **random**, nos va a permitir generar números aleatorios.

## 20.2. Ejercicio:

Crear una función *tomar\_experimento* que ejecute una prueba de tiempo de reacción. Para esto, recibe como argumento “rango\_pausa”, que es el rango de segundos máximo que puede demorar entre un estímulo y el siguiente.

Usando:

- La función **time.time()** para devolver el tiempo actual en segundos
- La función **random.random()** para generar un número aleatorio entre 0 y 1 y variar el tiempo de pausa
- La función **time.sleep(s)** para hacer una pausa de s segundos
- Y la función **input()** que aguarda una entrada del usuario, bloqueando la ejecución hasta que ingresa la tecla *ENTER*.

Pista en pseudocódigo:

```

Dado un entero "n_trials"

Hacer "n_trials" veces:
    Tomar un tiempo 0
    Aguardar input
    Tomar un tiempo 1
    Calcular diferencia entre tiempos
    Agregar diferencia a la lista
    Hacer una pausa random de tiempo

```

```
import time
import random
```

```
def tomar_experimento(n_trials, rango_pausa=10):
    pass
```

```

class TiempoDeReaccion(Experimento):
    '''Hereda la clase Experimento'''
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def corregir_datos(self):
        mu = sum(self.datos) / len(self.datos)
        var = sum([(x - mu)**2 for x in self.datos]) / len(self.datos)
        self.sujeto.resultados['MediaReaccion'] = mu
        self.sujeto.resultados['VarianzaReaccion'] = var

    # COMPLETAR
    # Usar la función tomar_experimento, esta vez con self en los argumentos
    # y guardar el resultado en self.datos
    # Al final del experimento, llamar a la función "corregir_datos"
    def tomar_experimento(self, n_trials, rango_pausa=10):
        pass

```

### 20.3. Utilizando nuestra clase

```
# Generar un str random como ID
import random
```

```
'9422781775311632'
```

```
# Instanciar Sujeto
```

```
# Instanciar TiempoDeReaccion
```

```
# Llamar metodo tomar_experimento
```

¿Que datos contiene?

```
# Llamamos un atributo
```

```
[0.6562693119049072,
 3.1354660987854004,
 0.6210577487945557,
 2.6110854148864746,
 0.8488471508026123]
```

### 20.4. Cuestionario

Generamos un cuestionario de ejemplo. Cada pregunta debe estar separada en un renglón (es decir por un caracter de *newline*, generalmente '\n')

```

with open('cuestionario.txt', 'w') as out:
    out.write(
        '''1. Me siento calmado.
2. Me siento seguro.
3. Estoy tenso.
4. Estoy contrariado.
5. Me siento a gusto.'''
    )

```

```

class Cuestionario(Experimento):
    '''Otro caso de prueba que hereda de Experimento. Esta vez la inicialización
    incluye cargar las preguntas en el atributo self.preguntas'''
    def __init__(self, path_cuestionario, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.cargar_preguntas(path_cuestionario)

    def cargar_preguntas(self, path):
        # COMPLETAR
        # Función que lee el txt y guarda cada renglón en self.preguntas

    def corregir_datos(self):
        self.sujeto.resultados[f'Total{self.nombre}'] = sum(self.datos)

    def tomar_experimento(self):
        # COMPLETAR
        # Dar la instrucción
        # Tomar cada pregunta con input()
        # Convertir la entrada a entero
        # Guardar la respuesta en self.datos
        # Al finalizar, ejecutar corregir_datos

    # Extra: agregar un chequeo de respuesta correcta y que sea int

```

Usamos la clase anterior para tomar el cuestionario

```

consigna = '''Aparecerán abajo algunas expresiones que las personas usan para
describirse.
Lea cada frase y presione el número que indique cómo se siente ahora mismo, siendo
0 = NO, 1 = POCO, 2 = BASTANTE, 3 = MUCHO.
'''

stai = Cuestionario('cuestionario.txt', sujeto, consigna = consigna,
nombre='Cuestionario')

```

```
# Tomar el experimento
```

```
sujeto
```

```

Datos de ID 9422781775311632

edad          32
fecha_nacimiento 01-01-1998
años_educacion   6
MediaReaccion    1.57454514503479
VarianzaReaccion 1.1579792508427773
TotalCuestionario 11

```

## 20.5. Generando Reporte

Una vez realizado el ejercicio vamos a generar una clase para realizar el reporte del experimento.

La clase Reporte va a generar un documento que contendrá:

- Un método para analizar los datos resultado del experimento;
- Un método para crear gráficos a partir de los resultados.
- Un método para generar el informe final

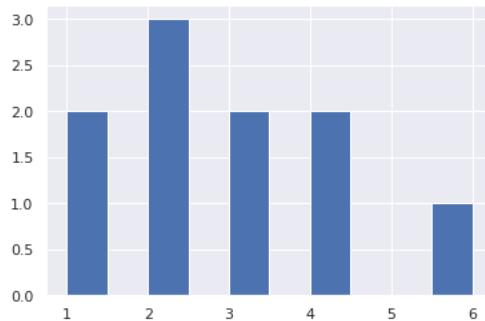
Para los gráficos vamos a usar matplotlib, que tiene métodos como "plot" donde recibe listas de números y genera un objeto "Figura". En este tema de visualización se ahonda en el curso de Análisis de Datos. La función `sns.set()` simplemente aplica un estilo.

```

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# Pequeño ejemplo de matplotlib
plt.hist([1,2,3,6,4,1,2,3,2,4]);

```



```
from docx import Document
from docx.shared import Cm
```

```
class Reporte():
    '''Esta clase va a generar un reporte para un Sujeto a partir de un Experimento'''
    def __init__(self, sujeto: Sujeto, experimento: Experimento, img_path = 'tmp.png'):
        self.suj = sujeto
        self.exp = experimento
        self.img_path = img_path

    def crear_graficos(self):
        '''Genera un gráfico de líneas y un histograma para los datos de los experimentos'''
        fig, ax = plt.subplots(1,2, figsize=(11,4))
        ax[0].plot(self.exp.datos)
        ax[0].set_ylabel('Error (ms)')
        ax[0].set_xlabel('Trials')
        sns.distplot(self.exp.datos, ax=ax[1])
        ax[1].set_ylabel('Frecuencia')
        ax[1].set_xlabel('Error (ms)')
        return ax

    def generar_reporte(self):
        '''Función que genera un documento de Word en base a los datos de Sujeto y Experimento. Ver más en:
        https://python-docx.readthedocs.io/en/latest/
        '''
        # Instanciamos un Documento
        documento = Document()

        # Agregamos un Titulo
        documento.add_heading(f'Informe de Resultados', 0)
        # Info del sujeto
        p = documento.add_paragraph(f'Sujeto ID: {self.suj.id}')
        p.add_run(f'{self.suj.id}').bold = True

        # Agregamos una lista de items iterando
        # el diccionario de info del Sujeto
        for k, v in self.suj.info.items():
            documento.add_paragraph(f'{k}= {v}', style='List Bullet')

        # Agregamos resultados, hay distintos estilos disponibles
        p = documento.add_paragraph(f'Resultados de {self.exp.nombre}', style='Intense Quote')

        # COMPLETAR
        # iterar los resultados en self.suj.resultados
        # agregando otra lista de items

        # Agregamos los gráficos
        ax = self.crear_graficos()
        plt.savefig(self.img_path) # usamos esta ruta provisoriamente
        plt.close() # para no mostrar el plot y solo guardar lo
        documento.add_picture(self.img_path, width=Cm(14))

        # Guardamos el archivo docx
        fp = f'reporte_{self.suj.id}.docx'
        documento.save(fp)
        return fp
```

```

# Instanciamos
rep = Reporte(sujeto, tr)

# Generar reporte
fp = rep.generar_reporte()

# Para descargar el doc en Colab
from google.colab import files

files.download(fp)

```

 Open in Colab

Recordá abrir en una nueva pestaña

## 21. Aplicación de POO: PACMAN

En esta guía vamos a empezar a diseñar una versión básica del tradicional juego PACMAN, organizando el código a partir de algunos conceptos de POO.

Qué elementos van a ser clases? Pensemos en cómo es el juego... En primer lugar, existe un juego, que surge de la interacción entre el PACMAN, los FANTASMAS, el MAPA... Antes de pensar cómo modelar el juego pensemos en el mapa.

¿Cómo describirían un mapa?

Una versión simple de un mapa podría considerar:

- Ser como una matriz, lo cual lo podríamos hacer con una lista de listas
- Ser accesible en sus posiciones, ya que vamos a querer posicionar a los fantasmas, a pacman y posiblemente otros objetos
- Ser printeable/printable
- Poder preguntarle si una posición existe en el mapa, o qué hay en una determinada posición

Escribamos primero la clase "Mapa".

```

import random
import sys

class Mapa:
    def __init__(self, n_rows, n_cols):
        # primero guardo la cantidad de filas y la cantidad de columnas
        self.n_rows = n_rows
        self.n_cols = n_cols
        # luego, genero un mapa vacío. Nota: ésto se puede hacer mejor usando
        # setters, dejémoslo para otra oportunidad
        self._map = [[CeldaVacia() for i in range(self.n_cols)] for x in
range(self.n_rows)]

    def __getitem__(self, row):
        # esta es una función mágica, nos va a permitir acceder de una forma muy
        # pythonizada
        return self._map[row] # pueden adivinar qué nos va a devolver?

    def __str__(self):
        # esta es otra función mágica, nos va a dar un print muy canchero, y luego
        # la vamos a modificar más
        str_map = ""
        for row in self._map:
            for col_idx in range(len(row)):
                str_map += str(row[col_idx])
                if col_idx == len(row) - 1:
                    str_map += "\n"
        return str_map

```

Y ahora vamos a generar el primer objeto que puede pertenecer a un mapa: una celda vacía. Queremos que esta celda vacía se muestre desde una clase que no es la original, usamos el método `repr` en lugar de `str`

```
class CeldaVacia:  
    def __repr__(self):  
        return " . "
```

Primero, veamos para qué sirve...

```
# instanciamos un mapa de 5 filas por 3 columnas  
mapa = Mapa(5,3)
```

```
mapa[4]
```

```
print(mapa)
```

```
mapa.n_cols
```

¿Qué va a pasar a continuación...?

```
mapa[4][4]
```

¿Qué más podemos imprimir dentro del mapa? A continuación vamos a crear las clases Pacman y Fantasma

```
class Fantasma:  
    def __repr__(self):  
        return " G "
```

```
class Pacman:  
    def __repr__(self):  
        return " P "
```

Ahora que ya creamos estos elementos, necesitamos crear métodos en el mapa que nos permitan agregarlos y también consultar para cada posición del mapa qué elemento la está ocupando.

```
class Mapa:  
    def __init__(self, n_filas:int, n_columnas:int):  
        # primero guardo la cantidad de filas y la cantidad de columnas  
        self.n_filas = n_filas  
        self.n_columnas = n_columnas  
        # luego, genero un mapa vacío. Nota: ésto se puede hacer mejor usando  
        # setters, dejémoslo para otra oportunidad  
        self._elementos = [[CeldaVacia() for i in range(self.n_columnas)] for x in  
                           range(self.n_filas)]  
  
    def __str__(self):  
        str_map = ""  
        for row in self._elementos:  
            for col_idx in range(len(row)):  
                element = row[col_idx]  
                str_map += str(element)  
                if col_idx == len(row) - 1:  
                    str_map += "\n"  
  
        return str_map  
  
    def modificar_elemento(self, elemento, pos_x, pos_y):  
        # método para modificar el tablero  
        self._elementos[pos_x][pos_y] = elemento  
  
    def get_elemento(self, x, y):  
        # método para tomar un elemento  
        return self._elementos[x][y]
```

Creamos un nuevo mapa

```
mapa = Mapa(7,8)
```

```
print(mapa)
```

Y vamos a agregar a Pacman en la posición (2,2)

```
mapa.modificar_elemento(Pacman(), 2, 2)

print(mapa)
```

Ahora vamos a crear la clase Juego. A la hora de inicializar el juego, creamos un mapa con los personajes ubicados en sus posiciones iniciales.

Ejercicio: Creen una clase “Juego” que tenga entre sus propiedades un mapa de 9x9 donde vamos a colocar un Pacman en la posición (2,3) y fantasmas en las posiciones (4,4) y (5,7).

## 21.1. Poblando el tablero e inicializando el juego

Ahora vamos a crear una clase Juego que se inicialice con un mapa de 7x8 y los personajes en distintas posiciones. También vamos a inicializar el score en 0 y crear un método start() que permita tomar el input del usuario y nos muestre el tablero.

```
class Juego:

    def __init__(self):
        # cuando instanciamos el juego creamos personajes. Por ahora las
        # posiciones son fijas pero se podrían hacer al azar
        mapa_inicial = Mapa(7,8)
        mapa_inicial.modificar_elemento(Pacman(), 4, 3)
        mapa_inicial.modificar_elemento(Fantasma(), 1, 3)
        mapa_inicial.modificar_elemento(Fantasma(), 0, 0)
        self.mapa = mapa_inicial
        self.puntaje = 0

    def start(self):
        fin = False
        while not fin:
            print(f"score:{self.puntaje}")
            print(self.mapa)
            input_movimiento = input()
```

```
juego = Juego()
juego.start()
```

## 21.2. Validar el input

Por ahora el juego nos permite ingresar cualquier valor, no reacciona a ese input y tenemos que terminar el juego manualmente.

Agreguemos, entonces un poco de funcionalidad a la clase Juego. Ahora vamos a validar que el input sea alguna de las teclas “a”, “w”, “s” o “d” que permiten mover a Pacman. Para eso vamos a crear un método privado en la clase \_input\_valido que valida si la tecla es correcta

```
class Juego:

    def __init__(self):
        # cuando instanciamos el juego creamos personajes. Por ahora las
        # posiciones son fijas pero se podrían hacer al azar
        mapa_inicial = Mapa(7,8)
        mapa_inicial.modificar_elemento(Pacman(), 4, 3)
        mapa_inicial.modificar_elemento(Fantasma(), 1, 3)
        mapa_inicial.modificar_elemento(Fantasma(), 0, 0)
        self.mapa = mapa_inicial
        self.puntaje = 0

    def start(self):
        # Loop con la lógica del juego
        fin = False
        while not fin:
            print(f"score:{self.puntaje}")
            print(self.mapa)
            input_movimiento = input()
            if not self._input_valido(input_movimiento):
                print("El input tiene que ser a, w, s o d")

    def _input_valido(self, move_input):
        # Valida si el input es correcto
        if move_input in ["a", "w", "s", "d"]:
            return True
        else:
            return False
```

```
juego = Juego()
juego.start()
```

### 21.3. Mover a Pacman

Ahora queremos reaccionar al input del usuario. Para esto vamos a invocar al método `_modificar_elemento`.

- Para poder mover a Pacman tenemos que consultar su posición. Esto lo hacemos con el método `_get_posicion_pacman`.
- Una vez que tenemos la posición vamos a calcular la nueva posición en función del movimiento del usuario. Noten que éste puede ser un método estático porque no modifica ni consulta ninguna de las propiedades de la clase `Juego`.
- Finalmente actualizamos el mapa del juego.

```
class Juego:

    def __init__(self):
        # cuando instanciamos el juego creamos personajes. Por ahora las
        # posiciones son fijas pero se podrían hacer al azar
        mapa_inicial = Mapa(7,8)
        mapa_inicial.modificar_elemento(Pacman(),4,3)
        mapa_inicial.modificar_elemento(Fantasma(),1,3)
        mapa_inicial.modificar_elemento(Fantasma(),0,0)
        self.mapa = mapa_inicial
        self.puntaje = 0

    def start(self):
        # Loop con la lógica del juego
        fin = False
        while not fin:
            print(f"score:{self.puntaje}")
            print(self.mapa)
            input_movimiento = input()
            if not self._input_valido(input_movimiento):
                print("El input tiene que ser a, w, s o d")
            else:
                self._actualizar_mapa(input_movimiento)

    def _input_valido(self, input_movimiento):
        # Valida si el input es correcto
        if input_movimiento in ["a","w","s","d"]:
            return True
        else:
            return False

    def _actualizar_mapa(self, input_movimiento):
        # Actualiza la posición del pacman
        pacman_x, pacman_y = self._get_posicion_pacman()
        nueva_x, nueva_y =
self._avanzar_posicion(pacman_x,pacman_y,input_movimiento)
        self.mapa.modificar_elemento(CeldaVacia(), pacman_x, pacman_y)
        self.mapa.modificar_elemento(Pacman(), nueva_x, nueva_y)
        return 0

    def _get_posicion_pacman(self):
        # Iteramos por todos los elementos del mapa del juego hasta encontrar a
        # Pacman
        for fila in range(self.mapa.n_filas):
            for columna in range(self.mapa.n_columnas):
                elemento = self.mapa.get_elemento(fila,columna)
                if isinstance(elemento,Pacman):
                    # Hay un único elemento de la clase Pacman. Los demás son
                    # Fantasma o Celdavacia
                    return fila, columna

    @staticmethod
    def _avanzar_posicion(x,y,move):
        if move == "a":
            y = y - 1
        elif move == "w":
            x = x - 1
        elif move == "s":
            x = x + 1
        elif move == "d":
            y = y + 1
        else:
            raise NotImplementedError(f"{move}")
        return x,y
```

Ahora movamos un poco a Pacman por el mapa del juego.

```
juego = Juego()  
juego.start()
```

## 21.4. Validar los movimientos de Pacman

Ahora nos queda agregar un poco más de funcionalidad:

- Validar que se pierde cuando Pacman coincide en el casillero con un Fantasma
- Aumentar el score en cada jugada
- Evitar que los Pacman se salga de los límites del mapa.

Como el resultado de mover a Pacman puede ser tanto chocarse con la pared, como coincidir con un fantasma y perder, como moverse exitosamente, vamos a ampliar la funcionalidad del método `_actualizar_mapa`.

Ahora este método devuelve un status, de acuerdo al resultado de la movida y vamos a renombrarlo como `_actualizar_mapa_y_devolver_status`. Noten lo importante que es que los métodos tengan nombres detallados que señalen todo lo que hacen. De esa forma el código es más legible y eso mejora la experiencia de trabajar colaborando con otros.

No tengan miedo de usar nombres largos para los métodos. Poner buenos nombres para las variables y funciones hace que el código cuente una historia que otros también puedan leer.

```

class Juego:

    def __init__(self):
        # cuando instanciamos el juego creamos personajes. Por ahora las
        # posiciones son fijas pero se podrían hacer al azar
        mapa_inicial = Mapa(7,8)
        mapa_inicial.modificar_elemento(Pacman(),4,3)
        mapa_inicial.modificar_elemento(Fantasma(),1,3)
        mapa_inicial.modificar_elemento(Fantasma(),0,0)
        self.mapa = mapa_inicial
        self.puntaje = 0

    def start(self):
        # Loop con la lógica del juego
        fin = False
        while not fin:
            print(f"score:{self.puntaje}")
            print(self.mapa)
            input_movimiento = input()
            if not self._input_valido(input_movimiento):
                print("El input tiene que ser a, w, s o d")
            else:
                status = self._actualizar_mapa_y_devolver_status(input_movimiento)
                if status == 1:
                    print("Cuidado con la pared")
                if status == 2:
                    print("Perdiste")
                    return
                elif status == 0:
                    self.puntaje += 1

    def _input_valido(self, input_movimiento):
        # Valida si el input es correcto
        if input_movimiento in ["a","w","s","d"]:
            return True
        else:
            return False

    def _actualizar_mapa_y_devolver_status(self, input_movimiento):
        # Actualiza la posición del pacman
        pacman_x,pacman_y = self._get_posicion_pacman()
        nueva_x, nueva_y =
self._avanzar_posicion(pacman_x,pacman_y,input_movimiento)
        print(nueva_x, nueva_y)
        if self._choca_la_pared(nueva_x, nueva_y):
            return 1
        elif isinstance(self.mapa.get_elemento(nueva_x, nueva_y),Fantasma):
            return 2
        else:
            self.mapa.modificar_elemento(CeldaVacia(), pacman_x, pacman_y)
            self.mapa.modificar_elemento(Pacman(), nueva_x, nueva_y)
            return 0

    def _get_posicion_pacman(self):
        # Iteramos por todos los elementos del mapa del juego hasta encontrar a
        Pacman
        for fila in range(self.mapa.n_filas):
            for columna in range(self.mapa.n_columnas):
                elemento = self.mapa.get_elemento(fila,columna)
                if isinstance(elemento,Pacman):
                    # Hay un único elemento de la clase Pacman. Los demás son
                    Fantasma o CeldaVacia
                    return fila, columna

    def _choca_la_pared(self, x, y):
        if x >= 0 and x < self.mapa.n_filas and y >= 0 and y <
self.mapa.n_columnas:
            return False
        return True

    @staticmethod
    def _avanzar_posicion(x,y,move):
        if move == "a":
            y = y - 1
        elif move == "w":
            x = x - 1
        elif move == "s":
            x = x + 1
        elif move == "d":
            y = y + 1
        else:
            raise NotImplementedError(f"{move}")
        return x,y

```

```
juego = Juego()  
juego.start()
```

## 21.5. Agregar el movimiento de los fantasmas

Ahora vamos a agregar movimiento al azar para los fantasmas al principio de la jugada. Estos pueden moverse en cuatro posibles direcciones. Para decidir adónde irán, vamos a ordenar estas cuatro posiciones al azar y elegir la primera que sea “válida”, es decir, que no se choca contra una pared ni trata de ocupar un casillero ya ocupado.

Noten cómo reutilizamos los métodos que habíamos usado para mover a Pacman “\_choca\_la\_pared” y “\_avanzar\_posicion” vuelven a usarse para mover a los fantasmas

```

class Juego:

    def __init__(self):
        # cuando instanciamos el juego creamos personajes. Por ahora las
        # posiciones son fijas pero se podrían hacer al azar
        mapa_inicial = Mapa(7,8)
        mapa_inicial.modificar_elemento(Pacman(), 4, 3)
        mapa_inicial.modificar_elemento(Fantasma(), 1, 3)
        mapa_inicial.modificar_elemento(Fantasma(), 0, 0)
        self.mapa = mapa_inicial
        self.puntaje = 0

    def start(self):
        # Loop con la lógica del juego
        fin = False
        while not fin:
            print(f"score:{self.puntaje}")
            print(self.mapa)
            status = self._mover_fantasmas_y_devolver_status()
            if status == 1:
                print("Perdiste")
                return
            input_movimiento = input()
            if not self._input_valido(input_movimiento):
                print("El input tiene que ser a, w, s o d")
            else:
                status = self._actualizar_mapa_y_devolver_status(input_movimiento)
                if status == 1:
                    print("Cuidado con la pared")
                if status == 2:
                    print("Perdiste")
                    return
                elif status == 0:
                    self.puntaje += 1

    def _input_valido(self, input_movimiento):
        # Valida si el input es correcto
        if input_movimiento in ["a", "w", "s", "d"]:
            return True
        else:
            return False

    def _actualizar_mapa_y_devolver_status(self, input_movimiento):
        # Actualiza la posición del pacman
        pacman_x, pacman_y = self._get_posicion_pacman()
        nueva_x, nueva_y =
        self._avanzar_posicion(pacman_x, pacman_y, input_movimiento)
        if self._choca_la_pared(nueva_x, nueva_y):
            return 1
        elif isinstance(self.mapa.get_elemento(nueva_x, nueva_y), Fantasma):
            return 2
        else:
            self.mapa.modificar_elemento(CeldaVacia(), pacman_x, pacman_y)
            self.mapa.modificar_elemento(Pacman(), nueva_x, nueva_y)
        return 0

    def _get_posicion_pacman(self):
        # Iteramos por todos los elementos del mapa del juego hasta encontrar a
        Pacman
        for fila in range(self.mapa.n_filas):
            for columna in range(self.mapa.n_columnas):
                elemento = self.mapa.get_elemento(fila, columna)
                if isinstance(elemento, Pacman):
                    # Hay un único elemento de la clase Pacman. Los demás son
                    Fantasma o CeldaVacia
                    return fila, columna

    def _choca_la_pared(self, x, y):
        if x >= 0 and x < self.mapa.n_filas and y >= 0 and y <
        self.mapa.n_columnas:
            return False
        return True

    @staticmethod
    def _avanzar_posicion(x,y,move):
        if move == "a":
            y = y - 1
        elif move == "w":
            x = x - 1
        elif move == "s":
            x = x + 1
        elif move == "d":
            y = y + 1
        else:
            raise NotImplementedError(f"{move}")

```

```

        return x,y

    def _mover_fantasmas_y_devolver_status(self):
        posiciones_fantasmas = self._get_posiciones_fantasmas()
        for posicion_fantasma in posiciones_fantasmas:
            nueva_x, nueva_y = self._nueva_posicion_fantasma(posicion_fantasma)
            if isinstance(self.mapa.get_elemento(nueva_x, nueva_y), Pacman):
                return 1
            else:
                self.mapa.modificar_elemento(Fantasma(), nueva_x, nueva_y)
                self.mapa.modificar_elemento(CeldaVacia(), posicion_fantasma[0],
posicion_fantasma[1])
        return 0

    def _nueva_posicion_fantasma(self, posicion_fantasma):
        posibles_direcciones = ["a", "w", "s", "d"]
        random.shuffle(posibles_direcciones)
        # El método shuffle modifica la lista "inplace" y cambia el orden original
        # por un orden aleatorio
        for direccion in posibles_direcciones:
            nueva_x, nueva_y = self._avanzar_posicion(posicion_fantasma[0],
posicion_fantasma[1], direccion)
            if not self._choca_la_pared(nueva_x, nueva_y) and \
not isinstance(self.mapa.get_elemento(nueva_x, nueva_y),
Fantasma):
                return nueva_x, nueva_y

    def _get_posiciones_fantasmas(self):
        posiciones_fantasmas = []
        for fila in range(self.mapa.n_filas):
            for columna in range(self.mapa.n_columnas):
                elemento = self.mapa.get_elemento(fila, columna)
                if isinstance(elemento, Fantasma):
                    posiciones_fantasmas.append((fila, columna))
        return posiciones_fantasmas

```

```
juego = Juego()
```

```
juego.start()
```

## 21.6. Fantasmas perseguidores

Ahora sería bueno que los fantasmas sean inteligentes. Es decir que persigan a Pacman por el tablero en vez de moverse aleatoriamente.

Para eso vamos a programarles el siguiente comportamiento: van a acercarse a Pacman en la dirección en la cual se encuentran más lejos. Lo bueno de este comportamiento es que nunca los va a hacer chocarse contra una pared así que podemos borrar ese control.

Imaginen esta posición en el tablero.

.	.	.	G(0,4)	.
.	.	.	.	.
P(3,0)	.	.	.	.
.	.	.	.	.

Para acercarse a Pacman el fantasma se debería mover hacia la izquierda porque la distancia más grande se encuentra en el eje y. Cuando la diferencia más grande está en el eje x, el fantasma se debería mover para arriba o para abajo. El signo de la diferencia es el que determina concretamente para dónde se mueve el fantasma. En caso de empate, cualquiera de las direcciones es lo mismo.

Ejercicio Agranden el tamaño del tablero a 10x10 y modifiquen el método `_mover_fantasmas_y_devolver_status` para que los fantasmas sean perseguidores.

```
juego = Juego()
juego.start()
```

## 22. Automatización y Minería Web

En esta carpeta encontrarán las clases sobre automatización de procesos usando Python.

1. Automatización I: En esta clase conocemos Pyautogui para operar con mouse y teclado, la librería OS para interactuar con el sistema operativo local, y una breve intro a Bash
2. Expresiones Regulares: Vemos **regex**, la extendida herramienta para buscar en strings que es tan recurrente en el trabajo de ciencia de datos.

Título	Clase	Ejercicios	Video
Automatización I	<a href="#"> Open in Colab</a>	<-	<a href="#">Video</a>
Expresiones Regulares	<a href="#"> Open in Colab</a>	<-	<a href="#">Video</a>
			<a href="#">Google Sheets</a>
			<a href="#">Cron</a>
			<a href="#">Google Data Studio</a>
			<a href="#">Enviar mails</a>
			<a href="#"> Open in Colab</a>

Recordá abrir en una nueva pestaña

## 23. Automatización:

En esta clase:

- Aprenderemos sobre programar desde una carpeta, o **directorio de trabajo**, y las una rutas o *path* relativos, o absolutos.
- Veremos como buscar archivos, también listar, borrar, copiar, mover y otras operaciones usuales con el sistema operativo. Librerías os, glob, shutil
- Lectura de distintos tipos de datos: .sav, .dta, .csv, .zip

Actualmente cualquier tarea que uno se proponga automatizar, así lo es. Podemos encontrar un espectro de complejidad:

- Por un lado, con la librería *pyautogui* podemos automatizar movimientos de mouse y teclado. Esto da la posibilidad de automatizar tareas sencillas de una manera directa.

(El siguiente ejemplo anda solo corriendolo como local!)

```
# Devuelve las dimensiones de la pantalla
ancho, alto = pyautogui.size()
# Devuelve la posición actual
Xactual, Yactual = pyautogui.position()
# Mover a coordenadas (pixels)
pyautogui.moveTo(100, 150)
# Clickear
pyautogui.click()
# Tipear con delay
pyautogui.write('Hello world!', interval=0.25)
# Apretar una tecla
pyautogui.press('esc')
pyautogui.hotkey('ctrl', 'c')
pyautogui.locateOnScreen()
```

**Nota:** Lamentablemente *pyautogui* es lo único incluido en cualquiera de nuestros cursos que no funciona en Colab! Para ejecutar esta sección instalar Anaconda en la computadora local

```
!pip install pyautogui
```

```
Requirement already satisfied: pyautogui in c:\programdata\anaconda3\lib\site-packages (0.9.52)
Requirement already satisfied: pygetwindow>=0.0.5 in
c:\programdata\anaconda3\lib\site-packages (from pyautogui) (0.0.9)
Requirement already satisfied: pymsgbox in c:\programdata\anaconda3\lib\site-packages (from pyautogui) (1.0.9)
Requirement already satisfied: pyscreeze>=0.1.21 in
c:\programdata\anaconda3\lib\site-packages (from pyautogui) (0.1.27)
Requirement already satisfied: PyTweening>=1.0.1 in
c:\programdata\anaconda3\lib\site-packages (from pyautogui) (1.0.3)
Requirement already satisfied: mouseinfo in c:\programdata\anaconda3\lib\site-packages (from pyautogui) (0.1.3)
Requirement already satisfied: pyrect in c:\programdata\anaconda3\lib\site-packages
(from pygetwindow>=0.0.5->pyautogui) (0.1.4)
Requirement already satisfied: Pillow>=6.2.1; python_version == "3.8" in
c:\programdata\anaconda3\lib\site-packages (from pyscreeze>=0.1.21->pyautogui)
(8.0.1)
Requirement already satisfied: pyperclip in c:\programdata\anaconda3\lib\site-packages (from mouseinfo->pyautogui) (1.8.2)
```

```
from time import sleep
import os
import pyautogui as pygui
import numpy as np
```

```
def cuadrado(l):
    pygui.drag(l, 0, duration=1)
    pygui.drag(0, l, duration=1)
    pygui.drag(-l, 0, duration=1)
    pygui.drag(0, -l, duration=1)
```

```
cuadrado(200)
```

```
url = r'https://www.lanacion.com.ar/'
```

```
sleep(2)
pygui.hotkey('ctrl', 't')
sleep(2)
pygui.write(url)
sleep(2)
pygui.press('enter')
```

```
def abrir_paint():
    # Windows
    os.system('start mspaint.exe')
    sleep(1)
    S = pygui.size()
    pygui.moveTo(S[0]//2, S[1]//2)
```

```
def cuadrado_espiral(l, ratio=0.9):
    pygui.moveTo(-l//2, -l//2, duration=0.1)
    for i in range(50):
        l = l*ratio
        pygui.drag(l, 0, duration=0.1)
        l = l*ratio
        pygui.drag(0, l, duration=0.1)
        l = l*ratio
        pygui.drag(-l, 0, duration=0.1)
        l = l*ratio
        pygui.drag(0, -l, duration=0.1)
        if l < 5:
            break
```

```
sleep(2)
abrir_paint()
cuadrado_espiral(600)
```

Ocasionalmente hay algunos casos de uso, pero encontramos obstáculos:

- Frágil ante cambios
- Método bruto, ad hoc
- Solo tareas sencillas

Encontraremos el mismo dilema más adelante en *web scraping*, en un extremo el consumo de APIs y en otro la automatización del navegador.

## 24. Bash y OS

### 24.1. Interactuando con el Sistema Operativo

```
# Con ! enviamos un comando a la terminal del sistema operativo
!pwd
```

```
C:\Users\Matias\MyStuff\cursos-python\Automatizacion
```

```
!ls
```

```
automatizacion_pyguyi_bash_os.ipynb  
Expo_2021.zip  
expresiones_regulares.ipynb  
expresiones_regulares_sol.ipynb  
gsheets_pytrends.ipynb
```

Algunos comandos a saber:

- pwd : “print working directory”
- ls: listar directorio actual
- wget: para descargar archivos

```
# http://microdatos.dane.gov.co/index.php/catalog/472/get_microdata  
!wget -O Expo_2021.zip https://unket.s3.sa-east-1.amazonaws.com/data/Expo_2021.zip
```

```
!pwd
```

```
C:\Users\Matias\MyStuff\cursos-python\Automatizacion
```

```
carpeta_actual = os.getcwd()  
carpeta_actual
```

```
'C:\\\\Users\\\\Matias\\\\MyStuff\\\\cursos-python\\\\Automatizacion'
```

```
carpeta_actual.split('\\')[-1]
```

```
'Automatizacion'
```

Librería OS

```
import os
```

```
#Devuelve el "current working directory", o directorio actual de trabajo  
os.getcwd()  
  
#Es por "change directory", o sea cambiar el directorio actual de trabajo  
os.chdir(path)  
  
#recorre recursivamente el árbol de directorios, empezando por el path.  
#en cada iteracion devuelve carpeta, [subcarpetas], [archivos]  
os.walk(path)  
  
#crea un directorio  
os.makedirs(path)  
  
#chequea si existe un path  
os.path.exists(path)  
  
#borra un archivo o carpeta vacia  
os.remove(path)  
  
#enumera el contenido del path  
os.listdir(path)  
  
#permite cambiar paths, nombres y extensiones  
os.rename(path, new_path)  
  
# Devuelve la ruta absoluta de una ruta relativa  
os.path.abspath(path)
```

Cuatro maneras de listar la carpeta actual

```
# Con la terminal  
!ls
```

```
automatizacion_pyguyi_bash_os.ipynb  
Expo_2021.zip  
expresiones_regulares.ipynb  
expresiones_regulares_sol.ipynb  
gsheets_pytrends.ipynb
```

```
!dir
```

```
Volume in drive C is Windows  
Volume Serial Number is 760A-09D7  
  
Directory of C:\Users\Matias\MyStuff\cursos-python\Automatizacion  
  
10/04/2021 10:50 PM <DIR> .  
10/04/2021 10:50 PM <DIR> ..  
10/04/2021 10:45 PM <DIR> .ipynb_checkpoints  
10/04/2021 10:50 PM 67,444 automatizacion_pyguyi_bash_os.ipynb  
10/04/2021 09:57 PM 65,811,481 Expo_2021.zip  
10/05/2021 12:25 AM 18,752 expresiones_regulares.ipynb  
10/05/2021 12:25 AM 18,350 expresiones_regulares_sol.ipynb  
10/05/2021 12:25 AM 25,871 gsheets_pytrends.ipynb  
5 File(s) 65,941,898 bytes  
3 Dir(s) 184,259,207,168 bytes free
```

```
# Con Python  
# ruta relativa  
  
os.listdir('..')
```

```
['.ipynb_checkpoints',  
'automatizacion_pyguyi_bash_os.ipynb',  
'Expo_2021.zip',  
'expresiones_regulares.ipynb',  
'expresiones_regulares_sol.ipynb',  
'gsheets_pytrends.ipynb']
```

```
# Con Python  
# ruta absoluta  
  
os.listdir(os.getcwd())
```

```
['.ipynb_checkpoints',  
'automatizacion_pyguyi_bash_os.ipynb',  
'Expo_2021.zip',  
'expresiones_regulares.ipynb',  
'expresiones_regulares_sol.ipynb',  
'gsheets_pytrends.ipynb']
```

Tres maneras de ver el directorio actual de trabajo

```
os.getcwd()
```

```
'C:\\\\Users\\\\Matias\\\\MyStuff\\\\cursos-python\\\\Automatizacion'
```

```
!pwd
```

```
C:\\\\Users\\\\Matias\\\\MyStuff\\\\cursos-python\\\\Automatizacion
```

```
os.path.abspath('..')
```

```
'C:\\\\Users\\\\Matias\\\\MyStuff\\\\cursos-python\\\\Automatizacion'
```

Ejercicio

Crear una carpeta nueva con la función `os.makedirs()`. Recibe simplemente la ruta a la carpeta a crear

```
new_dir = 'nueva_carpeta'
```

Ejercicio:

Cambiar el directorio de trabajo a 'new\_dir' usando os.chdir

```
| !pwd
```

```
| C:\Users\Matias\MyStuff\cursos-python\Automatizacion\nueva_carpet
```

```
| import zipfile  
|  
| # Extraemos en la nueva carpeta  
| with zipfile.ZipFile('../Expo_2021.zip', "r") as zip_ref:  
|     zip_ref.extractall('.')  
|
```

Buscamos todos los .zip

```
| for elemento in os.listdir('./Expo_2021/'):   
|     if elemento[-4:] == '.zip':  
|         print(elemento)
```

```
| Abril.zip  
| Agosto.zip  
| Enero.zip  
| Febrero.zip  
| Julio.zip  
| Junio.zip  
| Marzo.zip  
| Mayo.zip
```

Ahora usando glob

```
| from glob import glob
```

```
| # ruta con todos los archivos  
| datos_path = './Expo_2021/'
```

```
| zip_files = glob(datos_path + '/*.zip')  
| zip_files
```

```
| ['./Expo_2021\\Abril.zip',  
|  './Expo_2021\\Agosto.zip',  
|  './Expo_2021\\Enero.zip',  
|  './Expo_2021\\Febrero.zip',  
|  './Expo_2021\\Julio.zip',  
|  './Expo_2021\\Junio.zip',  
|  './Expo_2021\\Marzo.zip',  
|  './Expo_2021\\Mayo.zip']
```

Hacemos un bucle para recorrer los archivos .zip y descomprimirlos en carpetas correspondientes

```
| import zipfile  
|  
| # Recorremos la lista de zips  
| for f in zip_files:  
|  
|     # Definimos el nombre de la nueva carpeta en una variable  
|     # Es la misma ruta "f" pero sin el ".zip"  
|  
|     new_dir = f.replace('.zip', '')  
|     print(new_dir)  
|  
|     # Extraemos en la nueva carpeta  
|     with zipfile.ZipFile(f, "r") as zip_ref:  
|         zip_ref.extractall(new_dir)  
|         sleep(0.5)
```

```
./Expo_2021\Abril  
./Expo_2021\Agosto  
./Expo_2021\Enero  
./Expo_2021\Febrero  
./Expo_2021\Julio  
./Expo_2021\Junio  
./Expo_2021\Marzo  
./Expo_2021\Mayo
```

```
# Vemos que ya estan los .zips, y carpetas correspondientes  
os.listdir('Expo_2021')
```

```
['Abril',  
'Abril.zip',  
'Agosto',  
'Agosto.zip',  
'Enero',  
'Enero.zip',  
'Febrero',  
'Febrero.zip',  
'Julio',  
'Julio.zip',  
'Junio',  
'Junio.zip',  
'Marzo',  
'Marzo.zip',  
'Mayo',  
'Mayo.zip']
```

La estructura queda así:

```
.  
└── Expo_2021  
    ├── Abril  
    │   ├── Abril.csv  
    │   ├── Abril.dta  
    │   └── Abril.sav  
    └── Abril2021.zip  
  
...  
  
    ├── Octubre  
    │   ├── Octubre.csv  
    │   ├── Octubre.dta  
    │   └── Octubre.sav  
    └── Octubre2021.zip  
    ├── Septiembre  
    │   ├── Septiembre.csv  
    │   ├── Septiembre.dta  
    │   └── Septiembre.sav  
    └── Septiembre2021.zip
```

```
# Buscamos los archivos del tipo que queremos usando glob  
# Podemos cambiar el tipo cambiando la siguiente variable
```

```
# tipo = 'dta'  
# tipo = 'csv'  
tipo = 'sav'
```

```
datos = glob(f'{datos_path}/*/*{tipo}')  
datos
```

```
['./Expo_2021\Abril\\Abril.sav',  
 './Expo_2021\\Agosto\\Agosto.sav',  
 './Expo_2021\Enero\\Enero.sav',  
 './Expo_2021\\Febrero\\Febrero.sav',  
 './Expo_2021\\Julio\\Julio.sav',  
 './Expo_2021\\Junio\\Junio.sav',  
 './Expo_2021\\Marzo\\Marzo.sav',  
 './Expo_2021\\Mayo\\Mayo.sav']
```

```
datos_path + tipo + '_2021'
```

```
'./Expo_2021\sav_2021'
```

```
# Chequeamos si la carpeta ya existe, y si no creamos una carpeta para guardar todos archivos los de ese tipo
```

```
if not os.path.exists(datos_path + '/' + tipo + '_2021'):
    os.mkdir(datos_path + tipo + '_2021')
```

```
# Vemos que se creó la carpeta "sav_2021"
```

```
!ls Expo_2021
```

```
Abril
Abril.zip
Agosto
Agosto.zip
Enero
Enero.zip
Febrero
Febrero.zip
Julio
Julio.zip
Junio
Junio.zip
Marzo
Marzo.zip
Mayo
Mayo.zip
sav_2021
```

```
datos
```

```
['./Expo_2021\\Abril\\Abril.sav',
 './Expo_2021\\Agosto\\Agosto.sav',
 './Expo_2021\\Enero\\Enero.sav',
 './Expo_2021\\Febrero\\Febrero.sav',
 './Expo_2021\\Julio\\Julio.sav',
 './Expo_2021\\Junio\\Junio.sav',
 './Expo_2021\\Marzo\\Marzo.sav',
 './Expo_2021\\Mayo\\Mayo.sav']
```

```
# Movemos todos los que los archivos de ese tipo a la nueva carpeta
# Podemos chequear si ya está en la carpeta con os.path.exists
```

```
for d in datos:
    # Definimos en una variable cuál va a ser la nueva ruta
    new_path = datos_path + tipo + '_2021/' + d.replace('\\', '/').split('/')[-1]
    if not os.path.exists(new_path):
        os.rename(d, new_path)
```

```
new_path
```

```
'./Expo_2021/sav_2021/Abril.sav'
```

```
# Listamos los archivos dentro de la nueva carpeta
```

```
!ls {datos_path}/{tipo}_2021
```

```
Abril.sav
Agosto.sav
Enero.sav
Febrero.sav
Julio.sav
Junio.sav
Marzo.sav
Mayo.sav
```

```

.
├── Expo_2021
│   ├── Abril2021
│   │   ├── Abril.csv
│   │   └── Abril.dta
│   ├── Abril2021.zip
│   └── Agosto2021
│       └── Agosto.csv
...
└── Octubre2021
    ├── Octubre 2021.csv
    └── Octubre 2021.dta
├── Octubre2021.zip
└── sav_2021
    ├── Abril.sav
    ├── Agosto.sav
    ├── Diciembre.sav
    ├── Enero.sav
    ├── Febrero.sav
    ├── Julio.sav
    ├── Junio.sav
    ├── Marzo.sav
    ├── Mayo.sav
    ├── Noviembre 2021.sav
    ├── Octubre 2021.sav
    └── Septiembre.sav
└── Septiembre2021
    ├── Septiembre.csv
    └── Septiembre.dta
└── Septiembre2021.zip

```

En dos casos puede ser necesario recurrir a otra librería, *shutil*

Estos casos son:

- `copyfile`: para copiar archivos
- `rmtree`: para borrar directorios

```
from shutil import copyfile
copyfile(src, dst)
```

Usaremos esta 2da función para limpiar los archivos que extrajimos y no necesitamos

`zip_files`

```
['./Expo_2021\\Abril.zip',
 './Expo_2021\\Agosto.zip',
 './Expo_2021\\Enero.zip',
 './Expo_2021\\Febrero.zip',
 './Expo_2021\\Julio.zip',
 './Expo_2021\\Junio.zip',
 './Expo_2021\\Marzo.zip',
 './Expo_2021\\Mayo.zip']
```

`zip_dir`

```
'C:\\\\Users\\\\Matias\\\\MyStuff\\\\cursos-
python\\\\Automatizacion\\\\nueva_carpeta\\\\Expo_2021\\\\Abril'
```

```

import shutil

for f in zip_files:
    # ¡CUIDADO!
    # Estas funciones de borrar y renombrar son potencialmente peligrosas
    # Antes de ejecutarlas con una ruta, pueden hacer un print para asegurarse de
    # qué se haría

    # Borramos las carpetas de más, para eso
    # buscamos la ruta absoluta del zip.
    # Sabemos que carpeta donde se extrajo se llama igual pero sin .zip
    zip_dir = os.path.abspath(f).split('.')[0]
    print('Borrando...', zip_dir)

    shutil.rmtree(zip_dir)

    # Borraremos también los .zip
    os.remove(f)

```

Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Abril  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Agosto  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Enero  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Febrero  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Julio  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Junio  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Marzo  
Borrando... C:\Users\Matias\MyStuff\cursos-  
python\Automatizacion\nueva\_carpeta\Expo\_2021\Mayo

```

# Nos queda solo la carpeta con todos los archivos del tipo elegido, ordenados
!ls {datos_path}

```

sav\_2021

Cargamos los datos a un DataFrame

```

from glob import glob

rutas_datos = glob(datos_path + '/*/*' + tipo)
rutas_datos

```

```

['./Expo_2021\\sav_2021\\Abril.sav',
'./Expo_2021\\sav_2021\\Agosto.sav',
'./Expo_2021\\sav_2021\\Enero.sav',
'./Expo_2021\\sav_2021\\Febrero.sav',
'./Expo_2021\\sav_2021\\Julio.sav',
'./Expo_2021\\sav_2021\\Junio.sav',
'./Expo_2021\\sav_2021\\Marzo.sav',
'./Expo_2021\\sav_2021\\Mayo.sav']

```

Leemos 3 tipos de datos:

- Stata
- SPSS
- CSV

```

# para stata o SPSS
!pip install pyreadstat

```

```

import pandas as pd

datos = []

for r in rutas_datos:

    if tipo == 'dta':
        # En el caso de stata necesitamos instalar la siguiente librería
        import pyreadstat

        df, metadata = pyreadstat.read_dta(r)

    elif tipo == 'sav':
        df = pd.read_spss(r)

    elif tipo == 'csv':
        df = pd.read_csv(r, delimiter = ',')

    else:
        print('Otro tipo!')

    datos.append(df)

dataframe_total = pd.concat(datos)
dataframe_total.to_csv('todos.csv')

```

```
dataframe_total.sample(3)
```

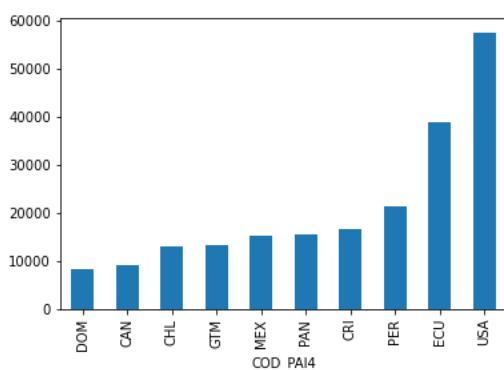
	FECH	ADUA	PAIS	COD_PA14	COD_SAL1	COD_SAL	DPTO2	VIA	BANDER
14447	2108.0	35.0	023		DEU	35.0	BUN	41.0	1.0
25769	2103.0	37.0	239		ECU	37.0	IPI	11.0	3.0
30939	2105.0	90.0	239		ECU	11.0	MDE	5.0	4.0

3 rows × 29 columns

```

dataframe_total.groupby('COD_PA14')\
    .count()['CANTI'] \
    .sort_values() \
    .iloc[-10:] \
    .plot.bar();

```



## 24.2. Ejercicio GSheets

1. Tomar la siguiente lista de temas
2. Crear una carpeta con su nombre en “title case”
3. Dentro de cada carpeta, guardar los datos de tendencias para el día de hoy

```
temas = ['Python', 'IA', 'Meditación', 'Jazz Rock', 'Sustentabilidad',
'Neurociencias']
```

La siguiente celda prepara y define una función `get_trends` para buscar tendencias de Google. La misma recibe una búsqueda y devuelve un diccionario con los datos de las tendencias.

```
!pip install pytrends

import pandas as pd
from pytrends.request import TrendReq

def get_trends(query):
    pytrend = TrendReq()
    pytrend.build_payload(kw_list=[query])
    df = pytrend.interest_by_region()
    return df.sort_values(query, ascending=False)[query].to_dict()
```

```
Requirement already satisfied: pytrends in c:\programdata\anaconda3\lib\site-packages (4.7.3)
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\site-packages (from pytrends) (2.24.0)
Requirement already satisfied: lxml in c:\programdata\anaconda3\lib\site-packages (from pytrends) (4.6.1)
Requirement already satisfied: pandas>=0.25 in c:\programdata\anaconda3\lib\site-packages (from pytrends) (1.1.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages (from requests->pytrends) (2020.6.20)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\programdata\anaconda3\lib\site-packages (from requests->pytrends) (1.25.11)
Requirement already satisfied: idna<3,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from requests->pytrends) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\programdata\anaconda3\lib\site-packages (from requests->pytrends) (3.0.4)
Requirement already satisfied: numpy>=1.15.4 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.25->pytrends) (1.19.2)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.25->pytrends) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.25->pytrends) (2020.1)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas>=0.25->pytrends) (1.15.0)
```

Ahora sí, recorrer la lista de temas y:

- Buscar las tendencias con la función `get_trends`
- Si no existe ya la carpeta para ese tema, crearla
- Crear un archivo .txt o .csv con los datos buscados dentro de esa carpeta

```
# Ejemplo
query = 'Python'
tendencias = get_trends(query)
tendencias
```

```
{'China': 100,  
 'St. Helena': 13,  
 'Singapore': 10,  
 'South Korea': 9,  
 'Israel': 9,  
 'Hong Kong': 7,  
 'Taiwan': 6,  
 'Switzerland': 5,  
 'Norway': 5,  
 'United States': 5,  
 'India': 5,  
 'Sweden': 4,  
 'Czechia': 4,  
 'Ireland': 4,  
 'Netherlands': 4,  
 'Canada': 4,  
 'Slovenia': 4,  
 'Tunisia': 4,  
 'New Zealand': 4,  
 'Finland': 4,  
 'Denmark': 4,  
 'Australia': 4,  
 'United Kingdom': 4,  
 'Germany': 4,  
 'United Arab Emirates': 3,  
 'Russia': 3,  
 'Sri Lanka': 3,  
 'Japan': 3,  
 'Croatia': 3,  
 'Pakistan': 3,
```

'Kenya': 3,  
'Portugal': 3,  
'Austria': 3,  
'Belgium': 3,  
'Bangladesh': 3,  
'France': 3,  
'Slovakia': 2,  
'Bulgaria': 2,  
'Romania': 2,  
'Serbia': 2,  
'Hungary': 2,  
'Belarus': 2,  
'Kazakhstan': 2,  
'Poland': 2,  
'Colombia': 2,  
'Philippines': 2,  
'Spain': 2,  
'Greece': 2,  
'Chile': 2,  
'Vietnam': 2,  
'Morocco': 2,  
'Ukraine': 2,  
'South Africa': 2,  
'Malaysia': 2,  
'Nigeria': 2,  
'Saudi Arabia': 1,  
'Egypt': 1,  
'Ecuador': 1,  
'Iran': 1,  
'Indonesia': 1,  
'Peru': 1,  
'Mexico': 1,  
'Thailand': 1,  
'Argentina': 1,  
'Italy': 1,  
'Brazil': 1,  
'Turkey': 1,  
'Namibia': 0,  
'Myanmar (Burma)': 0,  
'Mozambique': 0,  
'Panama': 0,  
'Nepal': 0,  
'Montserrat': 0,  
'Montenegro': 0,  
'Pitcairn Islands': 0,  
'Mongolia': 0,  
'Monaco': 0,  
'Moldova': 0,  
'Nauru': 0,  
'Nicaragua': 0,  
'Paraguay': 0,  
'New Caledonia': 0,  
'Niger': 0,  
'Niue': 0,  
'Norfolk Island': 0,  
'North Korea': 0,  
'Papua New Guinea': 0,  
'North Macedonia': 0,  
'Northern Mariana Islands': 0,  
'Puerto Rico': 0,  
'Palau': 0,  
'Palestine': 0,  
'Oman': 0,  
'Afghanistan': 0,  
'Qatar': 0,  
'Rwanda': 0,  
'Timor-Leste': 0,  
'Togo': 0,  
'Tokelau': 0,  
'Tonga': 0,  
'Trinidad & Tobago': 0,  
'Turkmenistan': 0,  
'Turks & Caicos Islands': 0,  
'Tuvalu': 0,  
'U.S. Outlying Islands': 0,  
'U.S. Virgin Islands': 0,  
'Uganda': 0,  
'Uruguay': 0,  
'Uzbekistan': 0,  
'Vanuatu': 0,  
'Vatican City': 0,  
'Venezuela': 0,  
'Wallis & Futuna': 0,  
'Western Sahara': 0,  
'Yemen': 0,  
'Zambia': 0,  
'Zimbabwe': 0,

'Tanzania': 0,  
'Tajikistan': 0,  
'São Tomé & Príncipe': 0,  
'South Georgia & South Sandwich Islands': 0,  
'Réunion': 0,  
'Samoa': 0,  
'San Marino': 0,  
'Senegal': 0,  
'Seychelles': 0,  
'Sierra Leone': 0,  
'Sint Maarten': 0,  
'Solomon Islands': 0,  
'Micronesia': 0,  
'South Sudan': 0,  
'Syria': 0,  
'St. Barthélemy': 0,  
'St. Kitts & Nevis': 0,  
'St. Lucia': 0,  
'St. Martin': 0,  
'St. Pierre & Miquelon': 0,  
'St. Vincent & Grenadines': 0,  
'Sudan': 0,  
'Suriname': 0,  
'Svalbard & Jan Mayen': 0,  
'Somalia': 0,  
'Libya': 0,  
'Mayotte': 0,  
'Burundi': 0,  
'Cameroon': 0,  
'Cape Verde': 0,  
'Caribbean Netherlands': 0,  
'Cayman Islands': 0,  
'Central African Republic': 0,  
'Chad': 0,  
'Christmas Island': 0,  
'Cocos (Keeling) Islands': 0,  
'Comoros': 0,  
'Congo - Brazzaville': 0,  
'Congo - Kinshasa': 0,  
'Cook Islands': 0,  
'Costa Rica': 0,  
'Cuba': 0,  
'Curaçao': 0,  
'Cyprus': 0,  
'Côte d'Ivoire': 0,  
'Djibouti': 0,  
'Dominica': 0,  
'Dominican Republic': 0,  
'El Salvador': 0,  
'Equatorial Guinea': 0,  
'Eritrea': 0,  
'Cambodia': 0,  
'Burkina Faso': 0,  
'Eswatini': 0,  
'Brunei': 0,  
'Algeria': 0,  
'American Samoa': 0,  
'Andorra': 0,  
'Angola': 0,  
'Anguilla': 0,  
'Antarctica': 0,  
'Antigua & Barbuda': 0,  
'Armenia': 0,  
'Aruba': 0,  
'Azerbaijan': 0,  
'Bahamas': 0,  
'Bahrain': 0,  
'Barbados': 0,  
'Belize': 0,  
'Benin': 0,  
'Bermuda': 0,  
'Bhutan': 0,  
'Bolivia': 0,  
'Bosnia & Herzegovina': 0,  
'Botswana': 0,  
'Bouvet Island': 0,  
'British Indian Ocean Territory': 0,  
'British Virgin Islands': 0,  
'Estonia': 0,  
'Ethiopia': 0,  
'Mauritius': 0,  
'Jamaica': 0,  
'Jordan': 0,  
'Kiribati': 0,  
'Kosovo': 0,  
'Kuwait': 0,  
'Kyrgyzstan': 0,

```
'Laos': 0,
'Latvia': 0,
'Lebanon': 0,
'Lesotho': 0,
'Liberia': 0,
'Albania': 0,
'Liechtenstein': 0,
'Lithuania': 0,
'Luxembourg': 0,
'Macao': 0,
'Madagascar': 0,
'Malawi': 0,
'Maldives': 0,
'Mali': 0,
'Malta': 0,
'Marshall Islands': 0,
'Martinique': 0,
'Mauritania': 0,
'Jersey': 0,
'Isle of Man': 0,
'Falkland Islands (Islas Malvinas)': 0,
'Iraq': 0,
'Faroe Islands': 0,
'Fiji': 0,
'French Guiana': 0,
'French Polynesia': 0,
'French Southern Territories': 0,
'Gabon': 0,
'Gambia': 0,
'Georgia': 0,
'Ghana': 0,
'Gibraltar': 0,
'Greenland': 0,
'Grenada': 0,
'Guadeloupe': 0,
'Guam': 0,
'Guatemala': 0,
'Guernsey': 0,
'Guinea': 0,
'Guinea-Bissau': 0,
'Guyana': 0,
'Haiti': 0,
'Heard & McDonald Islands': 0,
'Honduras': 0,
'Iceland': 0,
'Aland Islands': 0}
```

```
from datetime import datetime
fecha = datetime.now().strftime('%d-%m-%Y')
fecha
```

```
'04-10-21'
```

```
for t in temas:
    # completar
```

```
Creando 04-10-21/Python.txt
Creando 04-10-21/IA.txt
Creando 04-10-21/Meditación.txt
Creando 04-10-21/Jazz Rock.txt
Creando 04-10-21/Sustentabilidad.txt
Creando 04-10-21/Neurociencias.txt
```

 Open in Colab

Recordá abrir en una nueva pestaña

## 25. Automatización II: outputs, envío de mail, volcado a GSheets, scheduling

### 25.1. Hoja de ruta

- Ejemplo scraping sencillo con *Pandas*.
- Uso de la API de *Google Sheets*.
- Carga de datos en *Google Data Studio* y creación de un gráfico sencillo.

- Ejemplo envío automático de mail con *smtplib*.
- Scheduling: cron para Mac y linux, GUI en Windows.

## 25.2. Scrapping con Pandas

Vamos a utilizar la librería de pandas para obtener las tablas que contiene una página.

Documentación: [https://pandas.pydata.org/docs/reference/api/pandas.read\\_html.html](https://pandas.pydata.org/docs/reference/api/pandas.read_html.html)

```
import pandas as pd
import numpy as np

pd.set_option('display.float_format', lambda x: '%.7f' % x)

# Datos de criptomonedas
url = "https://coinmarketcap.com/new/"
```

Ahora con *pandas.read\_html()* es posible conseguir una lista con las tablas que contiene una *url*.

```
listadoTablas = pd.read_html(url)

listadoTablas[0].info()
```

Seleccionamos la tabla de las criptomonedas recientemente añadidas.

```
df = listadoTablas[0].copy()
df.drop(['#', 'Unnamed: 10', 'Unnamed: 0'], axis=1, inplace=True) # Eliminamos columna valores nulos.
df.columns = ['Name', 'Price', 'PctChnge_1h', 'PctChnge_24h',
'FullDiluted_MarketCap', 'Volume', 'Blockchain', 'Added_HoursAgo']
df.replace('--', '0', inplace=True) # Imputamos valores nulos como 0
df.replace('\.\.\.', '', regex=True, inplace=True) #Para liberarnos de los tres puntos. OJO: No devuelve el valor real, sino que le sacamos ceros.
df.fillna(0, inplace=True)
df.head(20)

# Convertimos a los tipos deseados

df['Price'] = df['Price'].replace( '[\$,)',' ', regex=True ).astype(float)
df['FullDiluted_MarketCap'] = df['FullDiluted_MarketCap'].replace( '[\$,)',' ', regex=True )
df['Volume'] = df['Volume'].replace( '[\$,)',' ', regex=True ).astype(float)
df['PctChnge_1h'] = df['PctChnge_1h'].replace( '[%,)',' ', regex=True ).astype(float)/100
df['PctChnge_24h'] = df['PctChnge_24h'].replace( '[%,)',' ', regex=True ).astype(float)/100
df['Added_HoursAgo'] = df['Added_HoursAgo'].apply(lambda x: x[0])

df.head()
```

## 25.3. Vinculación con Google Drive

Para poder escribir/leer un archivo que se encuentra en *Google Drive*, será necesario primero, contar con un archivo de *autenticación* a su vez que compartir el archivo pertinente con el servicio creado desde Python.

### 25.3.1. Generación de archivo de autenticación. (Conexión local)

Para ello entraremos en el siguiente [link](#) ingresando con la cuenta de google que querremos vincular. Una vez dentro de la plataforma de Google Cloud, crearemos un proyecto. Una vez dentro del proyecto entraremos en la sección **API y servicios** y daremos click en la opción **habilitar API y servicio**.

Google Cloud Platform - Humai

Menú de navegación

API y servicios + HABILITAR API Y SERVICIOS

Panel

Biblioteca

Credenciales

Pantalla de consentimiento ...

Verificación del dominio

Acuerdos de uso de páginas

Tráfico

No data is available for the selected time frame.

oct. 10 oct. 17 oct. 24 oct. 31

Allí dentro seleccionaremos el tipo de API que estemos necesitando. En este caso la de Google Drive y Google Sheets. Una vez habilitada la API dentro de nuestro proyecto, iremos a la sección de **credenciales**, y dentro de la misma daremos click a **crear credenciales**. Seleccionamos la opción de *Cuenta de servicio*. Una vez que la *Cuenta de servicio* haya sido generada, será posible acceder a sus configuraciones y generar una clave en formato json dentro de la misma.

Google Cloud Platform - Humai

API y servicios Google Drive API Credenciales + CREAR CREDENCIALES BORRAR

Descripción general

Métricas

Cuotas

Credenciales

Integración con IU de Drive

**Credenciales con**

ID de cliente de OAuth

Solicita el consentimiento del usuario para que tu app pueda acceder a sus datos

Cuenta de servicio

Habilita la autenticación de servidor a servidor en el nivel de la app mediante cuentas robot

Ayúdame a elegir

Responde algunas preguntas para que sepamos qué tipo de credencial usar

ID de clientes OAuth

Nombre Fecha de creación ↓

Cuentas de servicio

Correo electrónico

Nombre ↑

Claves

Humai-bot

DETALLES PERMISOS CLAVES MÉTRICAS REGISTROS

Las claves de cuenta de servicio podrían poner en riesgo la seguridad si se ven comprometidas. Te recomendamos que no descargas claves de cuenta de servicio, ya que es la mejor manera de autenticar las cuentas de servicio en Google Cloud [aquí](#).

Agrega un nuevo par de claves o sube un certificado de clave pública de un par de claves existente.

Impide la creación de claves de cuentas de servicio con las políticas de clave pública. [Más información para configurar políticas de la organización en cuentas de servicio](#)

AGREGAR CLAVE

Crear clave privada para "Humai-bot"

Descarga un archivo que contiene la clave privada. Almacena el archivo en un lugar seguro, ya que no es posible recuperar la clave si se pierde.

Tipo de clave

JSON Recomendado

P12 Para compatibilidad inversa con código en formato P12

CANCELAR CREAR

Más información sobre como crear un proyecto y habilitar una API [aquí](#)

```

# Esta funcion incluye todo lo que hicimos antes, para poder actualizar nuestro
DataFrame
def coinmarketcap_scraper():
    """Scraper de la pagina https://coinmarketcap.com/new/
    Obtiene los datos, los limpia y los devuelve como un DataFrame de Pandas.
    """
    url = "https://coinmarketcap.com/new/"

    # Scrapeamos la tabla con Pandas
    listadoTablas = pd.read_html(url)

    # creamos DataFrame
    df = listadoTablas[0].copy()

    # Limpieza de datos
    df.drop(['#', 'Unnamed: 10', 'Unnamed: 0'], axis=1, inplace=True) # Eliminamos
    columna valores nulos.
    df.columns = ['Name', 'Price', 'PctChnge_1h', 'PctChnge_24h',
    'FullDiluted_MarketCap', 'Volume', 'Blockchain', 'Added_HoursAgo']
    df.replace('--', '0', inplace=True) # Imputamos valores nulos como 0
    df.replace('.\.\.', '', regex=True, inplace=True) #Para liberarnos de los tres
    puntos. OJO: No devuelve el valor real, sino que le sacamos ceros.
    df.fillna('', inplace=True)
    df.head(20)

    # Convertimos a los tipos deseados
    df['Price'] = df['Price'].replace('[$,]', '', regex=True).astype(float)
    df['FullDiluted_MarketCap'] = df['FullDiluted_MarketCap'].replace('[$,]', '',
    regex=True).astype(float)
    df['Volume'] = df['Volume'].replace('[$,]', '', regex=True).astype(float)
    df['PctChnge_1h'] = df['PctChnge_1h'].replace('[%,]', '', regex=True
    ).astype(float)/100
    df['PctChnge_24h'] = df['PctChnge_24h'].replace('[%,]', '', regex=True
    ).astype(float)/100
    df['Added_HoursAgo'] = df['Added_HoursAgo'].apply(lambda x: x[0])

    return df

```

### 25.3.2. Usando gspread para interactuar con las hojas de cálculo de Google

Documentación: <https://docs.gspread.org/en/v5.3.0/>

```

# Instalamos y hacemos un upgrade de gspread porque la funcion que necesitamos
esta a partir de la version 3.6
!pip install gspread --upgrade

```

Importamos la librería gspread y chequeamos la versión

```

import gspread
print(f'Version de gspread:{gspread.__version__}')

```

```
Version de gspread:5.3.2
```

#### 25.3.2.1. Interactuamos con Google Sheets.

Creamos una nueva hoja de cálculo con su debido título y la compartimos con la cuenta desde la cual querremos acceder.

##### Autenticación local

- Documentación: <https://docs.gspread.org/en/latest/oauth2.html>

```

# Paso 1: Accedemos a nuestra cuenta y creamos la hoja de calculo
gc = gspread.service_account(filename='/content/credenciales_gsheets.json')

nombre = 'humai2'
hoja_de_calculo = gc.create(nombre)

# Para hacer visible el archivo es necesario compartirlo
tu_mail = 'mi_mail@mail.com'
hoja_de_calculo.share(tu_mail, perm_type='user', role='writer')

```

```
# Paso 2: Compartir la hoja con el 'client_email' que viene en el json
# Para eso simplemente abrimos nuestra hoja, vamos a 'Compartir' y ahí agregamos
el mail que encontramos
#en el json como si fuera un usuario mas
```

```
# Paso 3: Abrimos accedemos al documento
# Abrimos el documento
hoja_de_calculo = gc.open("humai2")

# Agarramos la primera de las hojas
worksheet = hoja_de_calculo.sheet1

# Actualizo la hoja
worksheet.update([df.columns.values.tolist()] + df.values.tolist())
```

```
# Paso 4: Obtenemos los valores desde nuestra hoja de calculo
nuestra_hoja = worksheet.get_all_values()

# Cargamos con Pandas
df_aux = pd.DataFrame(nuestra_hoja)
df_aux.columns = df_aux.iloc[0,:]
df_aux = df_aux.iloc[1:,:]
df_aux.head()
```

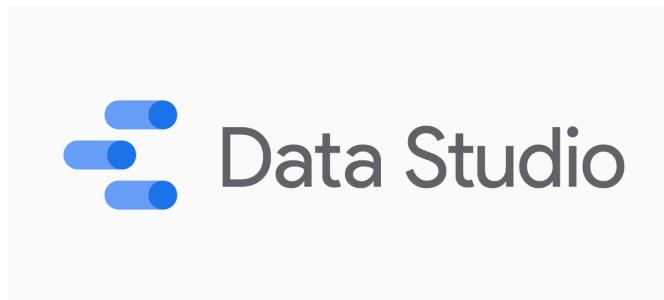
```
# actualizamos el df
df = coinmarketcap_scraper()
df
```

#### Otorgar acceso desde Google Colab

- En caso de acceder desde Google Colab resulta más sencillo autenticarse dado que lo hace automáticamente con la cuenta linkada.
- Una vez que ya tenemos nuestro objeto `gc` podemos trabajar usando los mismos métodos que en el caso local.

```
# Autenticarse con Colab
import gspread
from google.auth import default
creds, _ = default()
gc = gspread.authorize(creds)
```

#### 25.4. Interacción con Google Data Studio



[Google Data Studio](#) es una herramienta en línea para convertir datos en paneles e informes personalizables

#### 25.5. Envío automático de e-mails

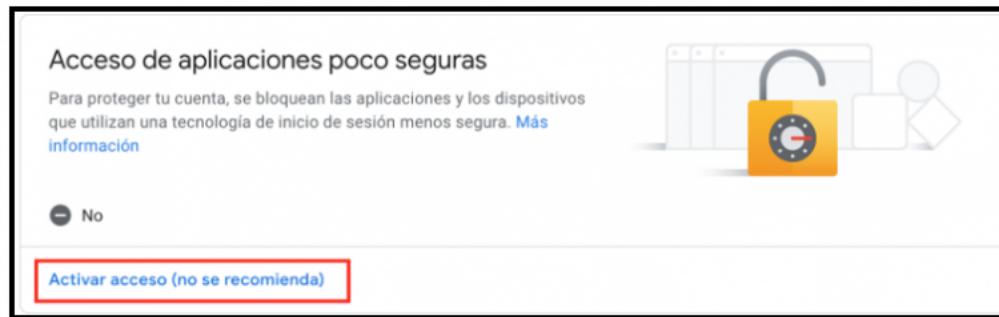
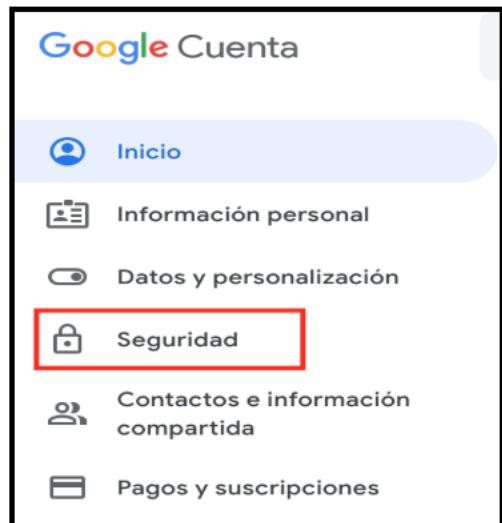
En esta sección haremos un ejemplo de como enviar mails desde Python

##### 25.5.1. Protocolo SMTP

El protocolo para transferencia simple de correo (en inglés: Simple Mail Transfer Protocol o SMTP) es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos (PDA, teléfonos móviles, impresoras, etcétera).

## 25.5.2. Usando Python

Para poder usar Python desde gmail tendremos que habilitar el uso de “aplicaciones poco seguras”.



Fuente: <https://docs.rocketbot.co/?p=1567>

- Si tienen problemas mirar aca: <https://stackoverflow.com/questions/26852128/smtpauthENTICATIONerror-when-sending-mail-using-gmail-and-python>

```
import smtplib
from email.message import EmailMessage

msg = EmailMessage()

# Contenido
msg['From']="curso_de_automatizacion_de_humai@gmail.com"
msg['To']="mi_mail@mail.com"
msg['Subject']= "Probando mandar mails!"
cuerpo_del_mail = 'Este es un mail enviado con Python en la clase! =D'
msg.set_content(cuerpo_del_mail)

# No se quedan en los detalles aquí, pero pueden leer más sobre el protocolo SMTP
#acá: https://es.wikipedia.org/wiki/Protocolo_para_transferencia_simple_de_correo
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()

# Usuario y contraseña
usuario = 'mi_usuario'

server.login(usuario, password)

# enviar
server.send_message(msg)
server.quit();
```

## 25.5.3. Enviar más de un mail

Podría existir el caso de uso donde queremos enviar más de un mail. Por ejemplo a todos nuestros alumnos con la nota de su parcial.

```

import smtplib
from email.message import EmailMessage
import time

notas = [10, 8, 7]
alumnos = ['Lupe', 'Juan', 'Sofia']
mails = ['Juan@mail.com', 'Sofia@mail.com', 'Lupe@mail.com']

# Usuario y contraseña
usuario = 'mi_usuario'

with smtplib.SMTP('smtp.gmail.com', 587) as server:
    for i in range(len(notas)):
        # Contenido
        msg = EmailMessage()

        msg['From']="curso_de_automatizacion_de_humai@gmail.com"
        msg['To']="tu_mail@mail.com" # Obviamente habría que ir variando los mails,
aca no lo voya hacer pero seria poner mails[i]
        msg['Subject']="Probando mandar mails!"
        cuerpo_del_mail = f'Hola {alumnos[i]}, tu nota en el parcial fue de
{notas[i]}. \n\nSaludos!'
        msg.set_content(cuerpo_del_mail)
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(usuario, password)
        # server.starttls()

        # enviar
        server.send_message(msg)
        time.sleep(3)
        print(f'mail enviado a {alumnos[i]}')

```

#### 25.5.4. Enviar archivos adjuntos

Podemos agregar archivos adjuntos como por ejemplo imágenes o PDFs.

```

# Enviar archivos adjuntos

import smtplib
# El módulo img HDR determina el tipo de imagen contenida en un archivo.
import img HDR
from email.message import EmailMessage

msg = EmailMessage()

# Contenido
msg['From']="curso_de_automatizacion_de_humai@gmail.com"
msg['To']="mi_mail@gmail.com"
msg['Subject']="Probando mandar mails!"
cuerpo_del_mail = 'Te estoy enviando una imagen con Python! =D'
msg.set_content(cuerpo_del_mail)

path_imagen = '/content/humai_logo.png'

with open(path_imagen, 'rb') as f:
    image_data = f.read()
    # Para saber el tipo de archivo
    image_type = img HDR.what(f.name)
    image_name = f.name

msg.add_attachment(image_data, maintype='image', subtype=image_type,
filename=image_name)

# No se quedan en los detalles aquí, pero pueden leer más sobre el protocolo SMTP
#acá: https://es.wikipedia.org/wiki/Protocolo_para_transferencia_simple_de_correo
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()

# Usuario y contraseña
usuario = 'mi_usuario'

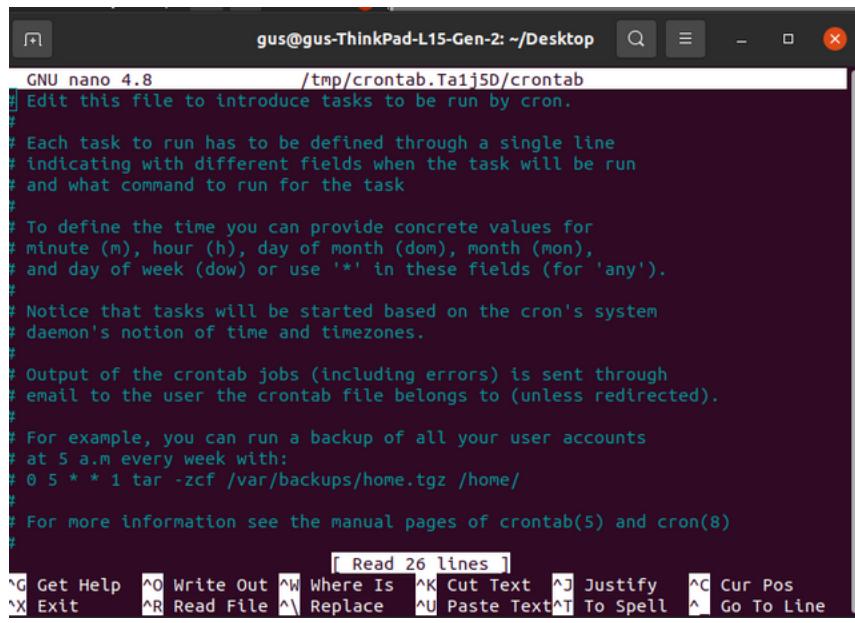
server.login(usuario, password)

# enviar
server.send_message(msg)
print('Mail enviado')
server.quit();

```

## 25.6. Scheduling con CRON

En el sistema operativo Unix, cron es un administrador regular de procesos en segundo plano (demonio) que ejecuta procesos o guiones a intervalos regulares (por ejemplo, cada minuto, día, semana o mes). Los procesos que deben ejecutarse y la hora a la que deben hacerlo se especifican en el archivo crontab.



```
GNU nano 4.8          /tmp/crontab.Ta1jSD/crontab
Edit this file to introduce tasks to be run by cron.

# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task

# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').

# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.

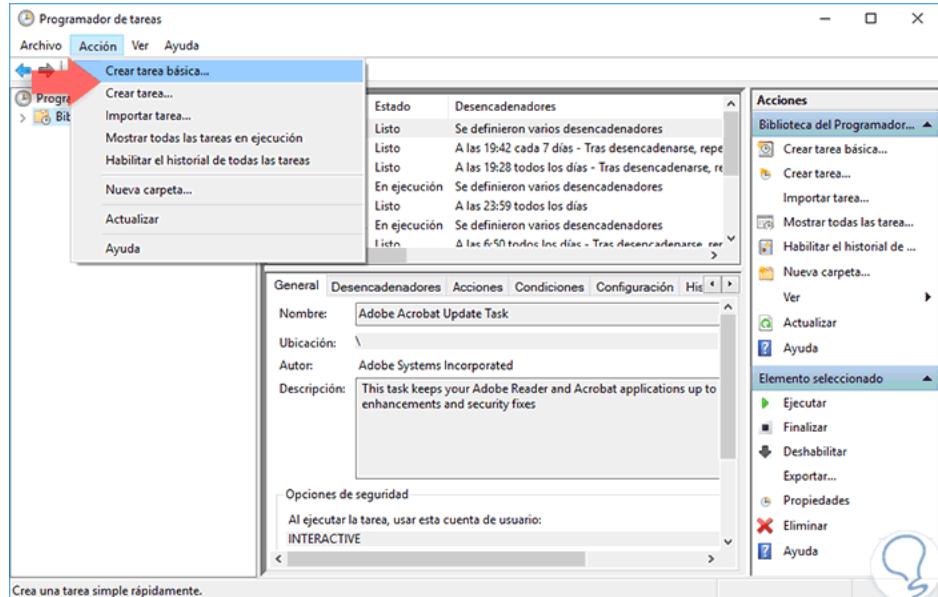
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).

# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/

# For more information see the manual pages of crontab(5) and cron(8)

[ Read 26 lines ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^L Go To Line
```

Cron se puede definir como el equivalente a Tareas Programadas de Windows.



Fuente: <https://www.solvetic.com/tutoriales/article/3441-como-abrir-y-configurar-programador-tareas-windows-10/>

### 25.6.1. Comandos básicos de Cron

En la terminal:

**crontab -l** -> Permite ver la lista de las tareas programadas

**crontab -e** -> Permite ver editar las tareas programadas

## ATENCION!

**crontab -r** -> Permite borrar las tareas programadas

Es importante que lo uses cuando quieras que tu tarea deje de ser ejecutada, sino va a quedar funcionando indefinidamente

```

#           Minutos (0 - 59)
#           |   Hora (0 - 23)
#           |   |   Dia del mes (1 - 31)
#           |   |   |   Mes (1 - 12) o jan,feb,mar,apr,may,jun,jul... (meses en
#           |   |   |   |   inglés)
#           |   |   |   |   |   día de la semana (0-6) (domingo=0 o 7) o
#           |   |   |   |   |   sun,mon,tue,wed,thu,fri,sat (días en inglés)
#           |   |   |   |   |
#           |   |   |   |   comando_a_ejecutar

```

Algunos ejemplos:

Todos los días a las 12 y media del mediodía corre esto

```
30 12 * * * python /ruta/a/mi/archivo/script.py
```

El 10 de cada mes corre esto a las 3 de la tarde

```
* 3 10 * * * python /ruta/a/mi/archivo/script.py
```

Consideraciones

- Dependiendo del intérprete de Python que tengan instalado pueden tener que poner algo distinto a la palabra python. Ejemplo: \* \* \* \* \*

```
python3 /ruta/a/mi/archivo/script.py
```
- Otra forma es que cron se posicione en la ruta del archivo y luego solo lo corra. Ejemplo: \* \* \* \* \* cd /ruta/a/mi/archivo && python

```
script.py
```

```

def escribir_archivo():
    with open('prueba.txt', 'a+') as f:
        f.write('Escribiendo archivo desde Cron\n')

if __name__ == '__main__':
    escribir_archivo()

```

## 25.6.2. Recursos

- Google Data Studio
  - <https://datastudio.google.com/gallery> les recomendamos ver la galeria para inspirarse y ver todo lo que se puede hacer con esta herramienta
- cron:
  - [Video de Corey Schafer](#), aunque esta en inglés se los super recomiendo
  - <https://crontab.guru/> Ayuda a escribir los comandos de cron

 Open in Colab

Recordá abrir en una nueva pestaña

# 26. Ejercicios Automatización II

## 26.1. Gsheets

```

# Instalamos y hacemos un upgrade de gspread porque la función que necesitamos
# esta a partir de la versión 3.6
!pip install gspread --upgrade

```

Importamos la librería gspread y chequeamos la versión

```

import gspread
print(f'Versión de gspread:{gspread.__version__}')

```

```
Version de gspread:5.3.2
```

### 26.1.1. Interactuamos con Google Sheets.

Creamos una nueva hoja de calculo con su debido título y la compartimos con la cuenta desde la cual querremos acceder.

```
# Accedemos a nuestra cuenta y creamos una hoja de calculo
gc = gspread.service_account(filename='/content/credenciales.json')

nombre = 'humai5'
hoja_de_calculo = gc.create(nombre)

# Para hacer visible el archivo es necesario compartirlo
tu_mail = 'tu_mail@gmail.com'
hoja_de_calculo.share(tu_mail, perm_type='user', role='writer')

hoja_de_calculo = gc.open(nombre)

# Agarramos la primera de las hojas
worksheet = hoja_de_calculo.sheet1
```

## 26.2. Automatizacion de Gsheets

- Recomendamos aprender a leer documentación: <https://docs.gspread.org/en/v5.3.2/>

1. Pedirle a la persona que ingrese datos de un producto y guardarlos en una hoja de google:

- Nombre del producto
- Precio
- Cantidad

La hoja tiene que tener una primer fila con esos tres nombres y cada celda tiene que tener fondo verde

Ayuditas :

- `input()`
- De la documentación:

```
# Update a range of cells using the top left corner address
wks.update('A1', [[1, 2], [3, 4]])
```

- Existe un método `format()`

```
#@title Solución

nombre = input('ingrese el nombre')
precio = input('ingrese el precio')
cantidad = input('ingrese la cantidad')

# Ejemplo basico
worksheet.update('A1', [['nombre', 'precio', 'cantidad'], [nombre, precio, cantidad]])

worksheet.format("A1:C1", {
    "backgroundColor": {
        "red": 0.0,
        "green": 1.0,
        "blue": 0.0
    },
    "horizontalAlignment": "CENTER",
    "textFormat": {
        "foregroundColor": {
            "red": 0.0,
            "green": 0.0,
            "blue": 0.0
        },
        "fontSize": 12,
        "bold": True
    }
})
```

## 26.3. Procesamiento automático de PDFs

Vamos a organizar en una hoja de calculo los pdfs según el tema del cual hablen (*Deep Learning, Neurociencia u otros*)

```
# Obtenemos los datos
# descargamos los papers ejemplo
!wget https://unket.s3.sa-east-1.amazonaws.com/data/papers.zip
# si ejecutan localmente o no tienen unzip en la terminal, descomprimir a mano
!unzip papers.zip
```

```
# Instalamos la libreria textract que nos permite extraer texto de documentos PDF
#(entre otros)
!apt-get install python-dev libxml2-dev libxslt1-dev antiword unrtf poppler-utils
\
pstoimage tesseract-ocr \
flac ffmpeg lame libmad0 libsox-fmt-mp3 sox libjpeg-dev swig libasound2-dev
libpulse-dev

!pip install git+https://github.com/deanmalmgren/textract
!pip install textract --upgrade
import textract
```

```
#Listamos los archivos de esa carpeta
import os

pdf_folder = '/content/papers'
files = [i for i in os.listdir(pdf_folder) if i.split('.')[1] == 'pdf']
files
```

```
# Funciones auxiliares
def limpiar(string):
    """Función para limpiar los renglones quitando los caracteres que no queremos
    considerar"""

    # \w+ significa "1 o más de un carácter alfanumérico". Repasar clase
    # de regex

    # versión sin regex:
    # no_valen = '0123456789-:.\'
    # return ''.join([i for i in string if i not in no_valen])

    return ''.join(re.findall('\w+', string))

def get_title(texto, largo_min=20):
    """Función para extraer un título. Podríamos mejorar la lógica o usar
    distintos enfoques"""
    renglones = [t for t in texto.split('\n') if len(limpiar(t)) > largo_min]
    if len(renglones) > 0:
        return renglones[0]
    else:
        return None
```

Ahora, recorremos cada archivo y para cada uno vamos a:

- Leer el texto con textract
- Extraer el título
- Estimar la categoría

```

import re

for filename in files:
    # ruta completa al archivo
    full_name = os.path.join(pdf_folder, filename)
    print('Documento: ', full_name.split('/')[-1])
    # de pdf a texto
    text = textract.process(full_name, language='eng').decode()
    # asumimos que el título está en los primeros 1000 caracteres
    top = text[:1000]

    # vamos a intentar extraer el título con nuestra función
    # si no encontramos un sub-string que cumpla con las condiciones,
    # mantenemos el nombre original
    title = get_title(top)
    if title == None:
        title = filename.replace('.pdf', '')

    # para categorizar los textos, vamos a usar esta lógica sencilla:
    # nos fijamos si contiene alguno de los siguientes términos clave
    is_deep = ('deep learning' in text.lower()) or ('statistic' in text.lower())
    is_neuro = ('neuro' in text.lower()) or ('brain' in text.lower())

    # para cada texto, lo vamos a mover a su carpeta y vamos a mostrar su
    # categoría
    print('título: ', title)
    if is_neuro:
        print('Corresponde a la categoría: Neuro')
    elif is_deep:
        print('Corresponde a la categoría: Deep')
    else:
        print('Corresponde a la categoría: Otros')

```

### 26.3.1. Ejercicio

Modificar el código anterior:

1. Agrupar los títulos del paper y del archivo del que vienen y guardarlos en alguna estructura de Python
2. Crear una nueva hoja de Google Sheets y generar automáticamente una tabla que contenga 3 columnas (deep, neuro y otros) con una fila por cada título correspondiente a esa categoría

```

#@title Posible solución
import re
neuro_list = []
deep_list = []
otros_list = []

for filename in files:
    # ruta completa al archivo
    full_name = os.path.join(pdf_folder, filename)
    print('full_name: ', full_name)
    # de pdf a texto
    text = extract_process(full_name, language='eng').decode()
    # asumimos que el título está en los primeros 1000 caracteres
    top = text[:1000]

    # vamos a intentar extraer el título con nuestra función
    # si no encontramos un sub-string que cumpla con las condiciones,
    # mantenemos el nombre original
    title = get_title(top)
    if title == None:
        title = filename.replace('.pdf', '')

    # para categorizar los textos, vamos a usar esta lógica sencilla:
    # nos fijamos si contiene alguno de los siguientes términos clave
    is_deep = ('deep learning' in text.lower()) or ('statistic' in text.lower())
    is_neuro = ('neuro' in text.lower()) or ('brain' in text.lower())

    # para cada texto, lo vamos a mover a su carpeta y vamos a mostrar su
    # categoría
    print(title)
    if is_neuro:
        print('Corresponde a la categoría: Neuro')
        neuro_list.append(f'{title} -> {filename}')
    elif is_deep:
        print('Corresponde a la categoría: Deep')
        deep_list.append(f'{title} -> {filename}')
    else:
        print('Corresponde a la categoría: Otros')
        otros_list.append(f'{title} -> {filename}')


import pandas as pd

df = pd.DataFrame(
    {'Neurociencia': pd.Series(neuro_list),
     'Deep Learning': pd.Series(deep_list),
     'Otros': pd.Series(otros_list)
    })

df.fillna('', inplace=True)
df

worksheet.update([df.columns.values.tolist()] + df.values.tolist())

```

## 26.4. Extracción de imágenes de PDF + Envío automático de e-mails

- Extraer imágenes del pdf
- Mandarlas por mail

### Vamos a usar [PyMuPDF](#):

*PyMuPDF is a Python binding for MuPDF – a lightweight PDF, XPS, and E-book viewer, renderer, and toolkit, which is maintained and developed by Artifex Software, Inc*

*MuPDF can access files in PDF, XPS, OpenXPS, CBZ, EPUB and FB2 (e-books) formats, and it is known for its top performance and high rendering quality.*

```
!pip3 install PyMuPDF Pillow
```

```
Requirement already satisfied: PyMuPDF in /usr/local/lib/python3.7/dist-packages
(1.19.6)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages
(7.1.2)
```

```

import fitz # PyMuPDF

# Funciones auxiliares

def get_pixmaps_in_pdf(pdf_filename):
    """Extrae las imágenes del documento .pdf"""
    doc = fitz.open(pdf_filename)
    xrefs = set()
    for page_index in range(doc.pageCount):
        for image in doc.get_page_images(page_index):
            xrefs.add(image[0])
    pixmaps = [fitz.Pixmap(doc, xref) for xref in xrefs]
    doc.close()
    return pixmaps

def write_pixmaps_to_pngs(pixmaps, path='/content/papers/', nombre='imagen'):
    """Guarda las imágenes como .png"""
    for i, pixmap in enumerate(pixmaps):
        pixmap.save(f'{path}{nombre}{i}.png') # Might want to come up with a better name

archivo = files[5]
pixmaps = get_pixmaps_in_pdf(os.path.join(pdf_folder, archivo))
write_pixmaps_to_pngs(pixmaps)

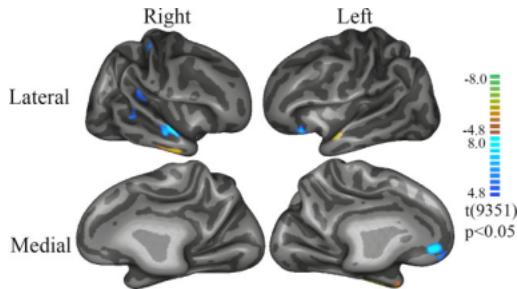
```

```

# Para renderizar una imagen en colab

from google.colab.patches import cv2_imshow # for image display
from skimage import io
image = io.imread('/content/papers/imagen1.png')
cv2_imshow(image)

```



#### 26.4.1. Enviar archivos adjuntos

Podemos agregar archivos adjuntos como por ejemplo imágenes o PDFs.

Tarea:

1. Extraer las imágenes de alguno de los papers
2. Envíarlas todas como archivos adjuntos por mail

```

#@title Posible solucion

import smtplib
# El módulo imghdr determina el tipo de imagen contenida en un archivo.
import imghdr
from email.message import EmailMessage

msg = EmailMessage()

# Contenido
msg['From']="curso_de_automatizacion_de_humai@gmail.com"
msg['To']="tu_mail@gmail.com"
msg['Subject']= "Probando mandar mails!"
cuerpo_del_mail = 'Te estoy enviando una imagen con Python! =D'
msg.set_content(cuerpo_del_mail)

filelist= [file for file in os.listdir('/content/papers') if
file.endswith('.png')]

for i in range(len(filelist)):
    with open('/content/papers/'+filelist[i], 'rb') as f:
        image_data = f.read()
        # Para saber el tipo de archivo
        image_type = imghdr.what(f.name)
        image_name = f.name

    msg.add_attachment(image_data, maintype='image', subtype=image_type,
filename=image_name)

# No se quedan en los detalles aquí, pero pueden leer más sobre el protocolo SMTP
#acá: https://es.wikipedia.org/wiki/Protocolo_para_transferencia_simple_de_correo
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()

# Usuario y contraseña
usuario = 'tu_mail@gmail.com'
password = 'tu_pass'

server.login(usuario, password)

# enviar
server.send_message(msg)
print('Mail enviado')
server.quit();

```

## 26.5. Enviar mails a distintas cuentas de mail guardadas en gsheets

Crear una hoja de calculo con dos columnas ("nombre" y "email") y 3 filas (los datos de 3 personas).

Luego tomar esa informacion de la hoja de calculo y enviarle los mails saludando a cada persona con el nombre correspondiente (similar al ejercicio mostrado en clase).

```

#@title Posible respuesta
### 1. Creo la hoja de calculo
import pandas as pd

df = pd.DataFrame(
    {'Nombre': pd.Series(['persona1', 'persona2', 'persona3']),
     'Mail': pd.Series(['mail1@gmail.com', 'mail2@gmail.com', 'mail3@gmail.com'])})

worksheet.update([df.columns.values.tolist()] + df.values.tolist())

### 2. Obtengo nuevamente la informacion en Python
nuestra_hoja = worksheet.get_all_values()
df_aux = pd.DataFrame(nuestra_hoja)
df_aux.columns = df_aux.iloc[0,:]
df_aux = df_aux.iloc[1:,:]
df_aux.head()

### 3. Envio los mails
import smtplib
from email.message import EmailMessage
import time

alumnnes = df_aux['Nombre'].to_list()
mails = df_aux['Mail'].to_list()

# Usuario y contraseña
usuario = 'tu_mail@gmail.com'

with smtplib.SMTP('smtp.gmail.com', 587) as server:
    for i in range(len(alumnnes)):
        # Contenido
        msg = EmailMessage()

        msg['From']="curso_de_automatizacion_de_humai@gmail.com"
        msg['To']= mails[i]
        msg['Subject']= "Probando mandar mails!"
        cuerpo_del_mail = f'Hola {alumnnes[i]}, ¿Todo bien? \n\nSaludos!'
        msg.set_content(cuerpo_del_mail)
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(usuario, password)
        # server.starttls()

        # enviar
        server.send_message(msg)
        time.sleep(3)
        print(f'mail enviado a {alumnnes[i]}')

```

 Open in Colab

Recordá abrir en una nueva pestaña

## 27. Expresiones Regulares

También conocido popularmente como **RegEx**. Son un mini lenguaje de programación diseñado para realizar búsquedas en strings. Son extremadamente útiles para:

- Extraer datos de distintos tipos de archivos, texto o con otro tipo de codificación.
- Web scraping: como veremos en las próximas clases, las regex son un buen método para encontrar la información que se necesita en un sitio web.
- Limpieza de datos: herramienta fundamental en el repertorio del científico de datos para limpiar datos quitando caracteres “ruidosos”, o armando nuevos “features” según la presencia o no de cierto texto.

### 27.1. Recursos útiles

- [Sitio para armar RegEx online](#)
- [Alternativa](#)
- [CheatSheet](#)

Python utiliza la librería llamada **re** para todo lo relacionado a regular expressions

```

import re

# a- extraer números de una oración.
texto = "Mi nombre es Juan y mi teléfono es 1564232324"
regla_de_busqueda = "15\d+"
print(texto)
print(re.findall(regla_de_busqueda, texto))

```

Las funciones principales de la librería re son:

- re.findall(pattern, string) para encontrar todos los resultados de una búsqueda
- re.search(pattern, string) para encontrar el primer resultado que coincide
- re.sub(pattern, replace, string) para substituir un texto por otro

## Sintaxis para construir regex

### Grupos de captura

( )	grupo de captura
(?:)	grupo de no captura

### Tipos de datos

\w	caracter alfanumérico	.	cualquier cosa menos \n
\d	dígito		operador “or”
\s	espacio en blanco	[m-z3-9]	rangos
\n	paso de línea	\W	Con mayusc. negamos. Ej: NO alfanumérico

### Operadores

	operador “or”
\	Escapar, o interpretar literalmente
[]	conjunto
[m-z3-9]	rangos

### Cuantificadores

+	Uno o más del elemento anterior
*	Cero o más del elemento anterior
{4,}	Cuatro o más del elemento anterior
?	Cambia el operador anterior de lazy a greedy

## 27.2. ¿Cómo se usa? Veamos ejemplos

```

# En realidad los teléfonos no son cualquier seguidilla de numeros
# suelen tener entre 6 y 8 numeros despues del 15
texto = "Mi nombre es María y mi teléfono es 1564232324"
regla_de_busqueda = "15\d{6,8}"
re.findall(regla_de_busqueda, texto)

```

```
[ '1564232324' ]
```

```
# En realidad los telefonos no arrancan siempre con 15
# capaz empiezan con 11 si son de buenos aires por ejemplo
texto = "Mi nombre es Carlos y mi teléfono es 1164232324"
regla_de_busqueda = "(?:15|11)\d{6,8}"
re.findall(regla_de_busqueda, texto)
```

```
['1164232324']
```

```
# En realidad los telefonos pueden tener un guión o espacio a parte de números
texto = "Mi nombre es asfasfeaf33 y mi teléfono es 11 6423-2324"
regla_de_busqueda = "(?:15|11)[0-9\s-]{6,10}"
re.findall(regla_de_busqueda, texto)
```

```
['11 6423-2324']
```

```
# b- Como extraer el mes de un texto
texto = "REPORTE DE PERFORMANCE - MES DE JUNIO"
regla_de_busqueda = "(MES DE(?:JULIO|AGOSTO|JUNIO))"
re.findall(regla_de_busqueda, texto)
```

```
['MES DE JUNIO']
```

```
# ¿Cómo hago que pare de buscar el operador * ?
text = "me llamo pedro. me gusta el rock."
regla_de_busqueda_no_greedy = "(.*?)\."
regla_de_busqueda_greedy = "(.*)"\\."
print(re.findall(regla_de_busqueda_no_greedy, text))
print(re.findall(regla_de_busqueda_greedy, text))
```

```
['me llamo pedro', ' me gusta el rock']
['me llamo pedro. me gusta el rock']
```

```
import re

comentario_de_mercadolibre = 'hola soy @mariadominguez, me interesa el producto,
te dejo mi celu 1565525233, saludos'

def encontrar_telefonos(texto):
    regla_de_busqueda = r'(15[0-9]{8})'
    return re.findall(regla_de_busqueda, texto)

def encontrar_usuarios(texto):
    regla_de_busqueda = r'@[a-zA-Z]+'
    return re.findall(regla_de_busqueda, texto)

print(encontrar_telefonos(comentario_de_mercadolibre))
print(encontrar_usuarios(comentario_de_mercadolibre))
```

```
['1565525233']
['mariadominguez']
```

## 27.3. Ejercicio

Usa regex para hacer una función que busque todos los emails en un texto

```
# Resolución

texto = "Hola te paso mi mail python@hotmail.com, saludos. Si no te funciona
mandame a este otro, pedro_2010@yahoo.com"
encontrar_emails(texto)
```

```
['python@hotmail.com', 'pedro_2010@yahoo.com']
```

### Ejercicio

Vamos a usar como ejemplo el [DSM](#), el libro de psiquiatría más importante en el mundo, en formato txt. El mismo se descargó en PDF y convirtió a texto usando [textract](#), una cómoda librería de Python.

De este texto:

1. Extraer los nombres de los médicos que aparecen.

Extracto de ejemplo:

```
Allan Burstein, M.D.  
David M. Clark, Ph.D.  
Lee Anna Clark, Ph.D.  
Deborah S. Cowley, M.D.
```

Es decir, en **cada renglón** sigue el patrón “[nombres], M.D.”.

Tip: recuerden cómo se representa el paso de línea (o “newline”):!

```
# Descargamos el texto  
!wget -nc https://unket.s3.sa-east-1.amazonaws.com/data/DSM.txt
```

```
--2021-10-04 20:30:07-- https://unket.s3.sa-east-1.amazonaws.com/data/DSM.txt  
Resolving unkett.s3.sa-east-1.amazonaws.com (unkett.s3.sa-east-1.amazonaws.com)...  
52.95.163.122  
Connecting to unkett.s3.sa-east-1.amazonaws.com (unkett.s3.sa-east-  
1.amazonaws.com)|52.95.163.122|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 2445688 (2,3M) [text/plain]  
Saving to: 'DSM.txt'  
  
DSM.txt          100%[=====] 2,33M 2,04MB/s   in 1,1s  
  
2021-10-04 20:30:09 (2,04 MB/s) - 'DSM.txt' saved [2445688/2445688]
```

```
import re  
  
with open('DSM.txt', 'r') as inp:  
    texto = inp.read()
```

```
# Completar
```

```
['ALLEN FRANCES, M.D.',  
'HAROLD ALAN PINCUS, M.D.',  
'MICHAEL B. FIRST, M.D.',  
'Magda Campbell, M.D.',  
'Judith H. Gold, M.D.',  
'Robert E. Hales, M.D.',  
'David J. Kupfer, M.D.',  
'Roger Peele, M.D.',  
'A. John Rush, M.D.',  
'Marc Alan Schuckit, M.D.]
```

Ahora buscar a las personas con PhDs (observen el extracto anterior)

Hacer una RegEx para extraer nombres de condiciones mentales, ejemplo:

Ejemplo

```
Clinical judgment must be used in distinguishing developmentally appropriate  
levels of separation anxiety from the clinically significant concerns about separation  
seen in Separation Anxiety Disorder.  
I Diagnostic criteria for 309.21 Separation Anxiety  
Disorder  
A. Developmentally inappropriate and excessive anxiety concerning sep-  
aration
```

Debería dar “Separation Anxiety”. Es decir, el patrón es: “Diagnostic criteria for [números con punto] [nombre de la condición]”

Tip: podemos hacer que el patrón contemple mayúsculas y minúsculas (con [A-Z] y [a-z]) o indicar un “flag” para que se ignoren las mayúsculas con

```
re.findall(patrón, string, flags=re.IGNORECASE)
```

Extraer los nombres de los países de los profesionales que fueron colaboradores internacionales en esta última edición

Extracto de ejemplo:

```
Michael Gelder, M.D. (England)
Semyon Gluzman, M.D. (former USSR)
Judith H. Gold, M.D. (Canada)
Marcus Grant, Ph.D. (Switzerland)
Herta A. Guttman, M.D. (Canada)
Heinz Hafner, M.D. (Germany)
Robert Hare, Ph.D. (Canada)
```

Pistas:

1. Para que sea más sencillo consideren solo países cuyo nombre sea una sola palabra.
2. También pueden considerar que siempre está **después de M.D. o de PhD.** (recuerden cómo buscar una u otra cosa!).
3. Cuidado también con “escapar” los paréntesis literales

```
paises = None
```

Ahora contemos cuántas veces aparece cada país con la siguiente clase Counter. Simplemente pasen su lista de `paises` como argumento al constructor Counter() y les dará el resultado.

```
from collections import Counter
Counter(paises)
```

```
Counter({'Sweden': 4,
          'Uruguay': 1,
          'Switzerland': 8,
          'Nigeria': 1,
          'England': 22,
          'Austria': 2,
          'Denmark': 5,
          'Australia': 9,
          'Hungary': 1,
          'Poland': 3,
          'Canada': 22,
          'Ireland': 1,
          'Italy': 4,
          'Korea': 1,
          'Brazil': 3,
          'Argentina': 1,
          'Norway': 4,
          'Germany': 8,
          'France': 4,
          'Scotland': 2,
          'Japan': 4,
          'Netherlands': 5,
          'Singapore': 2,
          'Bulgaria': 1,
          'Israel': 3,
          'Spain': 1,
          'Greece': 2,
          'Belgium': 2,
          'Kenya': 1,
          'Egypt': 1,
          'Luxembourg': 1,
          'Finland': 2,
          'Mexico': 1,
          'Peru': 2,
          'Venezuela': 1,
          'Romania': 1,
          'China': 1})
```

 Open in Colab

Recordá abrir en una nueva pestaña

## 28. Práctica: GSHEET y PyTrends

En esta clase veremos:

- Repaso Python
- Interactuando con Google Sheets
- Manipulación de strings

- Diccionario
- Función Enviar Mail
- Pytrends

## 28.1. Preparación

Para operar sobre Google Sheets desde Python, tenemos que contar con un archivo de autenticación y compartir el archivo a manipular con el servicio o “bot” creado. Desde Colab, vamos a poder autenticar a nuestro usuario directamente:

```

import gspread

# Colab
from google.colab import auth
from oauth2client.client import GoogleCredentials
#auth.authenticate_user()
#gc = gspread.authorize(GoogleCredentials.get_application_default())

# Local
!pip install gspread --upgrade
gc = gspread.service_account(filename='alumnos.json')

sheet_name = 'Alumnos'
sh = gc.open(sheet_name)
data = sh.sheet1.get_all_records()

print(type(data))
print(type(data[0]))

<class 'list'>
<class 'dict'>

data

for d in data:
    print(d['Mail'].split('@')[0])

```

### 28.1.1. Ejercicio:

- ¿Qué estructura de datos es?
- Los nombres donde aparece el apellido primero, darlos vuelta a la forma “Nombre Apellido”

Pseudocódigo:

```

para cada elemento:
    si el nombre está al revés:
        separar el nombre por la coma
        invertirlo
        unirlo

for d in data:
    if ',' in d['Nombre']:
        d['Nombre'] = ' '.join(d['Nombre'].split(',')[:-1]).strip()
    data

```

### 28.1.2. Ejercicio:

Definir la llave “Primer nombre” con el primer nombre en cada entrada

Pseudocódigo

```

para cada elemento:
    separar el string
    guardar la primera palabra en el dict con el key "primer nombre"

```

```
for d in data:  
    d['Primer Nombre'] = d['Nombre'].split()[0]  
data
```

### 28.1.3. Ejercicio:

Guardar una lista con los dominios únicos de los mails

Pseudocódigo

```
para cada elemento:  
    tomar el mail  
    extraer el dominio  
    si no está en la lista  
        guardarlo
```

```
dominios = []  
  
for d in data:  
    dominio = d['Mail'].split('@')[1].split('.')[0]  
    if dominio not in dominios:  
        dominios.append(dominio)  
  
dominios
```

```
['gmail', 'intalbid', 'iadb', 'outlook', 'live', 'connectamericas', 'ihum']
```

### 28.1.4. Bonus:

- Usar regular expressions

```
import re  
  
dominios = []  
  
for d in data:  
    dominio = re.findall('@(\w+)\.', d['Mail'])[0]  
    if dominio not in dominios:  
        dominios.append(dominio)  
  
dominios
```

```
['gmail', 'intalbid', 'iadb', 'outlook', 'live', 'connectamericas', 'ihum']
```

### 28.1.5. Bonus:

- En vez de una lista, hacer un diccionario con la frecuencia de cada dominio

```
import re  
  
dominios = {}  
  
for d in data:  
    dominio = re.findall('@(\w+)\.', d['Mail'])[0]  
    if dominio not in dominios:  
        dominios[dominio] = 1  
    else:  
        dominios[dominio] += 1  
  
dominios
```

```
{'connectamericas': 1,  
 'gmail': 3,  
 'iadb': 8,  
 'ihum': 1,  
 'intalbid': 2,  
 'live': 1,  
 'outlook': 1}
```

## 28.1.6. Mandando mails

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

msg = MIMEMultipart()

# Contenido
msg['From']="ejemplo.cuenta.246@gmail.com"
msg['To']="mg@ihum.ai"
msg['Subject']= "Probando mandar mails!"
msg.attach(MIMEText('Este es un mail enviado con Python', 'plain'))

# No se queden en los detalles aquí
server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()

# Usuario y contraseña
server.login("ejemplo.cuenta.246@gmail.com", "Ejemplo246")

# enviar
server.send_message(msg)
server.quit();
```

## 28.1.7. Ejercicio:

- Hacer una función con el código anterior para mandar mails

```
def mandar_mail(usuario, contra, receptor, asunto, contenido):
    msg = MIMEMultipart()

    # Contenido
    msg['From']= usuario
    msg['To']= receptor
    msg['Subject']= asunto
    msg.attach(MIMEText(contenido, 'plain'))

    # No se queden en los detalles aquí
    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.starttls()

    # Usuario y contraseña
    server.login(usuario, contra)

    # enviar
    server.send_message(msg)
    server.quit();
```

## 28.1.8. Ejercicio

### Enviando mail

- Hacer un bucle que rellene la plantilla para cada diccionario
- Definir un string como plantilla para llenar con “Primer Nombre” y “Tema”
- Si el mail es el tuyo... enviar el mail con la plantilla. Si no, hacer print
- Mostrar “Enviando mail a...” con el mail a quien se envió

```
mi_nombre = 'Matías Grinberg'
plantilla = '{Hola! {nombre}. Tu tema es {tema}'

for d in data:
    texto = plantilla.format(nombre=d['Primer Nombre'], tema=d['Tema'])

    if d['Nombre'] == mi_nombre:
        mandar_mail("ejemplo.cuenta.246@gmail.com", "Ejemplo246", d['Mail'],
                    "Tema", texto)
        print(f'Enviando Mail a {d["Mail"]}')
    else:
        print(texto)
```

## 28.1.9. Bonus

Usar la función .update\_cell() del objeto sh para anotar “OK” en una nueva columna llamada “Mail Enviado”, donde corresponda

```

enviado_idx = sheet.find('Enviado').col
mi_nombre = 'Matias Grinberg'
plantilla = '¡Hola! {nombre}. Tu tema es {tema}'

for d in data:
    texto = plantilla.format(nombre=d['Primer Nombre'], tema=d['Tema'])

    if d['Nombre'] == mi_nombre:
        mandar_mail("ejemplo.cuenta.246@gmail.com", "Ejemplo246", d['Mail'],
                    "Tema", texto)
        print(f'Enviando Mail a {d["Mail"]}')
        sh.update_cell(row_idx, enviado_idx, 'OK')
    else:
        print(texto)

```

## 28.1.10. Bonus

Usar la siguiente función para conseguir las tendencias de Google para el tema de cada persona

```
!pip install pytrends
```

```

Requirement already satisfied: pytrends in /usr/local/lib/python3.6/dist-packages
(4.7.3)
Requirement already satisfied: lxml in /usr/local/lib/python3.6/dist-packages (from
pytrends) (4.2.6)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages
(from pytrends) (2.23.0)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.6/dist-
packages (from pytrends) (1.0.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-
packages (from requests->pytrends) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-
packages (from requests->pytrends) (2020.6.20)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests->pytrends) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->pytrends) (2.10)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.25->pytrends) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in
/usr/local/lib/python3.6/dist-packages (from pandas>=0.25->pytrends) (2.8.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.25->pytrends) (1.18.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages
(from python-dateutil>=2.6.1->pandas>=0.25->pytrends) (1.15.0)

```

```

import pandas as pd
from pytrends.request import TrendReq

def get_trends(query):
    pytrend = TrendReq()
    pytrend.build_payload(kw_list=[query])
    df = pytrend.interest_by_region()
    return df.sort_values(query, ascending=False)[query].to_dict()

```

```
query = 'Python'
tendencias = get_trends(query)
```

```
tendencias
```

```
{'Afghanistan': 0,
 'Albania': 0,
 'Algeria': 0,
 'American Samoa': 0,
 'Andorra': 0,
 'Angola': 0,
 'Anguilla': 0,
 'Antarctica': 0,
 'Antigua & Barbuda': 0,
 'Argentina': 1,
 'Armenia': 0,
 'Aruba': 0,
 'Australia': 4,
 'Austria': 3,
 'Azerbaijan': 0,
```

'Bahamas': 0,  
'Bahrain': 0,  
'Bangladesh': 2,  
'Barbados': 0,  
'Belarus': 3,  
'Belgium': 3,  
'Belize': 0,  
'Benin': 0,  
'Bermuda': 0,  
'Bhutan': 0,  
'Bolivia': 0,  
'Bosnia & Herzegovina': 0,  
'Botswana': 0,  
'Bouvet Island': 0,  
'Brazil': 1,  
'British Indian Ocean Territory': 0,  
'British Virgin Islands': 0,  
'Brunei': 0,  
'Bulgaria': 2,  
'Burkina Faso': 0,  
'Burundi': 0,  
'Cambodia': 0,  
'Cameroon': 0,  
'Canada': 4,  
'Cape Verde': 0,  
'Caribbean Netherlands': 0,  
'Cayman Islands': 0,  
'Central African Republic': 0,  
'Chad': 0,  
'Chile': 2,  
'China': 100,  
'Christmas Island': 0,  
'Cocos (Keeling) Islands': 0,  
'Colombia': 2,  
'Comoros': 0,  
'Congo - Brazzaville': 0,  
'Congo - Kinshasa': 0,  
'Cook Islands': 0,  
'Costa Rica': 0,  
'Croatia': 3,  
'Cuba': 0,  
'Curaçao': 0,  
'Cyprus': 0,  
'Czechia': 4,  
'Côte d'Ivoire': 0,  
'Denmark': 4,  
'Djibouti': 0,  
'Dominica': 0,  
'Dominican Republic': 0,  
'Ecuador': 1,  
'Egypt': 1,  
'El Salvador': 0,  
'Equatorial Guinea': 0,  
'Eritrea': 0,  
'Estonia': 0,  
'Eswatini': 0,  
'Ethiopia': 0,  
'Falkland Islands (Islas Malvinas)': 0,  
'Faroe Islands': 0,  
'Fiji': 0,  
'Finland': 4,  
'France': 3,  
'French Guiana': 0,  
'French Polynesia': 0,  
'French Southern Territories': 0,  
'Gabon': 0,  
'Gambia': 0,  
'Georgia': 0,  
'Germany': 4,  
'Ghana': 0,  
'Gibraltar': 0,  
'Greece': 2,  
'Greenland': 0,  
'Grenada': 0,  
'Guadeloupe': 0,  
'Guam': 0,  
'Guatemala': 0,  
'Guernsey': 0,  
'Guinea': 0,  
'Guinea-Bissau': 0,  
'Guyana': 0,  
'Haiti': 0,  
'Heard & McDonald Islands': 0,  
'Honduras': 0,  
'Hong Kong': 6,  
'Hungary': 2,  
'Iceland': 0,

'India': 5,  
'Indonesia': 1,  
'Iran': 1,  
'Iraq': 0,  
'Ireland': 4,  
'Isle of Man': 0,  
'Israel': 9,  
'Italy': 1,  
'Jamaica': 0,  
'Japan': 3,  
'Jersey': 0,  
'Jordan': 0,  
'Kazakhstan': 1,  
'Kenya': 2,  
'Kiribati': 0,  
'Kosovo': 0,  
'Kuwait': 0,  
'Kyrgyzstan': 0,  
'Laos': 0,  
'Latvia': 0,  
'Lebanon': 0,  
'Lesotho': 0,  
'Liberia': 0,  
'Libya': 0,  
'Liechtenstein': 0,  
'Lithuania': 0,  
'Luxembourg': 0,  
'Macao': 0,  
'Madagascar': 0,  
'Malawi': 0,  
'Malaysia': 2,  
'Maldives': 0,  
'Mali': 0,  
'Malta': 0,  
'Marshall Islands': 0,  
'Martinique': 0,  
'Mauritania': 0,  
'Mauritius': 0,  
'Mayotte': 0,  
'Mexico': 1,  
'Micronesia': 0,  
'Moldova': 0,  
'Monaco': 0,  
'Mongolia': 0,  
'Montenegro': 0,  
'Montserrat': 0,  
'Morocco': 2,  
'Mozambique': 0,  
'Myanmar (Burma)': 0,  
'Namibia': 0,  
'Nauru': 0,  
'Nepal': 0,  
'Netherlands': 4,  
'New Caledonia': 0,  
'New Zealand': 4,  
'Nicaragua': 0,  
'Niger': 0,  
'Nigeria': 2,  
'Niue': 0,  
'Norfolk Island': 0,  
'North Korea': 0,  
'North Macedonia': 0,  
'Northern Mariana Islands': 0,  
'Norway': 4,  
'Oman': 0,  
'Pakistan': 2,  
'Palau': 0,  
'Palestine': 0,  
'Panama': 0,  
'Papua New Guinea': 0,  
'Paraguay': 0,  
'Peru': 1,  
'Philippines': 2,  
'Pitcairn Islands': 0,  
'Poland': 2,  
'Portugal': 3,  
'Puerto Rico': 0,  
'Qatar': 0,  
'Romania': 2,  
'Russia': 3,  
'Rwanda': 0,  
'Réunion': 0,  
'Samoa': 0,  
'San Marino': 0,  
'Saudi Arabia': 1,  
'Senegal': 0,  
'Serbia': 2,

```
'Seychelles': 0,
'Sierra Leone': 0,
'Singapore': 9,
'Sint Maarten': 0,
'Slovakia': 2,
'Slovenia': 4,
'Solomon Islands': 0,
'Somalia': 0,
'South Africa': 3,
'South Georgia & South Sandwich Islands': 0,
'South Korea': 8,
'South Sudan': 0,
'Spain': 2,
'Sri Lanka': 3,
'St. Barthélemy': 0,
'St. Helena': 12,
'St. Kitts & Nevis': 0,
'St. Lucia': 0,
'St. Martin': 0,
'St. Pierre & Miquelon': 0,
'St. Vincent & Grenadines': 0,
'Sudan': 0,
'Suriname': 0,
'Svalbard & Jan Mayen': 0,
'Sweden': 4,
'Switzerland': 5,
'Syria': 0,
'São Tomé & Príncipe': 0,
'Taiwan': 5,
'Tajikistan': 0,
'Tanzania': 0,
'Thailand': 1,
'Timor-Leste': 0,
'Togo': 0,
'Tokelau': 0,
'Tonga': 0,
'Trinidad & Tobago': 0,
'Tunisia': 3,
'Turkey': 1,
'Turkmenistan': 0,
'Turks & Caicos Islands': 0,
'Tuvalu': 0,
'U.S. Outlying Islands': 0,
'U.S. Virgin Islands': 0,
'Uganda': 0,
'Ukraine': 2,
'United Arab Emirates': 2,
'United Kingdom': 4,
'United States': 5,
'Uruguay': 0,
'Uzbekistan': 0,
'Vanuatu': 0,
'Vatican City': 0,
'Venezuela': 0,
'Vietnam': 1,
'Wallis & Futuna': 0,
'Western Sahara': 0,
'Yemen': 0,
'Zambia': 0,
'Zimbabwe': 0,
'Aland Islands': 0}
```

```
mi_nombre = 'Matías Grinberg'
plantilla = '¡Hola! {nombre}. Tu búsqueda de <b>"{tema}"</b> trajo los siguientes resultados: \n {resultados} \n ¡Saludos! MatiBot'

for d in data:
    tendencias = get_trends(d['Tema'])
    texto = plantilla.format(nombre=d['Primer Nombre'], tema=d['Tema'],
    resultados=tendencias)

    if d['Nombre'] == mi_nombre:
        mandar_mail("ejemplo.cuenta.246@gmail.com", "Ejemplo246", d['Mail'],
        "Tema", texto)
        print(f'Enviando Mail a {d["Mail"]}')
    else:
        print(texto)
```

¡Eso es todo por hoy!

 Open in Colab

Recordá abrir en una nueva pestaña

## 29. Ejercicio: Organizando PDFs

Supongamos que tenemos una carpeta con muchos artículos científicos bajados de ArXiv como [este](#) de Maldacena, el renombrado físico argentino. En esta notebook vamos a usar [textract](#) para leer textos de PDFs y así:

- Reemplazar el título de algo como "9711200.pdf" al título del trabajo
- Categorizar los trabajos
- Crear carpetas por categoría
- Mover los artículos a cada una de las carpetas

```
# instalamos la librería que vamos a utilizar  
!pip install textract
```

```
import textract  
import os  
from shutil import copyfile
```

```
# descargamos los papers ejemplo  
!wget https://unket.s3.sa-east-1.amazonaws.com/data/papers.zip  
# si ejecutan localmente o no tienen unzip en la terminal, descomprimir a mano  
!unzip papers.zip
```

```
--2021-10-06 19:35:31-- https://unket.s3.sa-east-1.amazonaws.com/data/papers.zip  
Resolving unkett.s3.sa-east-1.amazonaws.com (unkett.s3.sa-east-1.amazonaws.com)...  
52.95.165.122  
Connecting to unkett.s3.sa-east-1.amazonaws.com (unkett.s3.sa-east-  
1.amazonaws.com)|52.95.165.122|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 50053476 (48M) [application/zip]  
Saving to: 'papers.zip'  
  
papers.zip      100%[=====] 47,73M  3,56MB/s    in 24s  
  
2021-10-06 19:35:55 (2,03 MB/s) - 'papers.zip' saved [50053476/50053476]  
  
Archive: papers.zip  
  creating: papers/  
  inflating: papers/1907.12461.pdf  
  inflating: papers/1909.06349.pdf  
  inflating: papers/2002.07655.pdf  
  inflating: papers/2009.11423.pdf  
  inflating: papers/1508.06576.pdf  
  inflating: papers/1601.00670.pdf  
  inflating: papers/1603.07285v1.pdf  
  inflating: papers/1606.05908.pdf  
  inflating: papers/1606.07873.pdf  
  inflating: papers/1611.03673.pdf  
  inflating: papers/1611.08024 (1).pdf  
  inflating: papers/1706.03762.pdf  
  inflating: papers/1903.01458.pdf  
  inflating: papers/barratt1959.pdf  
  inflating: papers/eysenck1977.pdf  
  inflating: papers/eysenck1985.pdf  
  inflating: papers/fpsyg-02-00080.pdf  
  inflating: papers/langenecker2007.pdf  
  inflating: papers/li2009.pdf  
  inflating: papers/luengo1991.pdf  
  inflating: papers/patmehenysonbook.pdf
```

```
# la ruta a la carpeta con los trabajos  
pdf_dir = r"./papers"
```

```
# listamos todos los archivos de esa carpeta  
files = os.listdir(pdf_dir)
```

Creamos un directorio para cada una de las 3 categorías, "Neuro", "Deep" (para Deep Learning) y "Otros".

```
ruta_neuro = 'Neuro'  
os.makedirs(ruta_neuro, exist_ok=True)
```

```
ruta_deep = 'Deep'  
os.makedirs(ruta_deep, exist_ok=True)
```

```

ruta_otros = 'Otros'
os.makedirs(ruta_otros, exist_ok=True)

```

Vamos a necesitar una función para extraer el título. Como no queremos usar regex todavía (se ve próximamente), vamos a usar también esta siguiente función para limpiar un renglón para ver si tiene el largo suficiente para ser un título.

```

def limpiar(string):
    """Función para limpiar los renglones quitando los caracteres que no queremos considerar"""

    # \w+ significa "1 o más de un carácter alfanumérico". Ver más en la clase
    # de regex

    # versión sin regex:
    # no_valen = '0123456789-:.\'
    # return ''.join([i for i in string if i not in no_valen])

    return ''.join(re.findall('\w+', string))

def get_title(texto, largo_min=20):
    """Función para extraer un título. Podríamos mejorar la lógica o usar
    distintos enfoques"""
    renglones = [t for t in texto.split('\n') if len(limpiar(t)) > largo_min]
    if len(renglones) > 0:
        return renglones[0]
    else:
        return None

```

Ahora, recorremos cada archivo y para cada uno vamos a:

- Leer el texto con textextract
- Extraer el título
- Estimar la categoría
- Mover y renombrarlo

```

for filename in files:
    # ruta completa al archivo
    full_name = os.path.join(pdf_dir, filename)
    # de pdf a texto
    text = textextract.process(full_name, language='eng').decode()
    # asumimos que el título está en los primeros 1000 caracteres
    top = text[:1000]

    # vamos a intentar extraer el título con nuestra función
    # si no encontramos un sub-string que cumpla con las condiciones,
    # mantenemos el nombre original
    title = get_title(top)
    if title == None:
        title = filename.replace('.pdf', '')

    # para categorizar los textos, vamos a usar esta lógica sencilla:
    # nos fijamos si contiene alguno de los siguientes términos clave
    is_deep = ('deep learning' in text.lower()) or ('statistic' in text.lower())
    is_neuro = ('neuro' in text.lower()) or ('brain' in text.lower())

    # para cada texto, lo vamos a mover a su carpeta y vamos a mostrar su
    # categoría
    print(title)
    if is_neuro:
        print('Neuro')
        copyfile(f'{pdf_dir}/{filename}', f'{ruta_neuro}/{title}.pdf')
    elif is_deep:
        print('Deep')
        copyfile(f'{pdf_dir}/{filename}', f'{ruta_deep}/{title}.pdf')
    else:
        print('Otros')
        copyfile(f'{pdf_dir}/{filename}', f'{ruta_otros}/{title}.pdf')
    print()

```

Variational Inference: A Review for Statisticians  
Neuro

arXiv:1603.07285v1 [stat.ML] 23 Mar 2016  
Deep

Person. indiuid. D# Vol. 12, No. 7. pp. 657-667. 1991  
Neuro

Perceptual alzd Motor Skills, 1959, 9, 191-198. @ Southern Universities Press 1959  
Deep

Deep Learning for Cognitive Neuroscience  
Neuro

Under review as a conference paper at ICLR 2017  
Neuro

Br. J. soc. d i n . Psychol. (1977), 16, 57-68  
Neuro

patmethenysongbook  
Otros

Task-Oriented Dialogue as Dataflow Synthesis  
Neuro

This article was downloaded by: [University of Newcastle (Australia)]  
Neuro

arXiv:2002.07655v1 [q-bio.NC] 18 Feb 2020  
Neuro

Tutorial on Variational Autoencoders  
Deep

A Neural Algorithm of Artistic Style  
Neuro

Leveraging Pre-trained Checkpoints for Sequence Generation Tasks  
Deep

Person. indiuid. Off. Vol. 6. No. 5, pp. 613419,  
Neuro

Slice-based Learning: A Programming Model for  
Deep

arXiv:1706.03762v5 [cs.CL] 6 Dec 2017  
Neuro

Hypothesis and Theory Article  
Neuro

Human Brain Mapping 31:410-423 (2010)  
Neuro

EEGNet: A Compact Convolutional Neural Network  
Neuro

arXiv:1606.07873v1 [cs.CV] 25 Jun 2016  
Deep



Recordá abrir en una nueva pestaña

## 30. Edición de Imágenes con Python

Pillow es un módulo para interactuar con archivos de imagen. Éste posee muchas funciones que facilitan por ejemplo la edición, los recortes y los cambios de tamaño. Con el poder de manipular imágenes de la misma manera en la que se haría con otro software como Microsoft Paint o Adobe Photoshop, Python puede editar automáticamente cientos o miles de imágenes con facilidad.

## 31. Color

### 31.1. Función *ImageColor.getcolor()*:

Usualmente los colores se codifican en una tupla RGBA de cuatro elementos: Red, Green, Blue y Alpha. Los tres primeros representan el grado de cada uno de los colores rojo, verde y azul, mientras que Alpha indica la transparencia. Cada parámetro tiene un mínimo de 0 y un máximo de 255. Pillow nos brinda una función que nos puede ayudar a identificar el código RGBA de un color a partir de su nombre en inglés:

```
from PIL import ImageColor  
  
print(ImageColor.getcolor('red', 'RGBA'))  
print(ImageColor.getcolor('RED', 'RGBA'))  
print(ImageColor.getcolor('Black', 'RGBA'))  
print(ImageColor.getcolor('chocolate', 'RGBA'))  
print(ImageColor.getcolor('CornflowerBlue', 'RGBA'))
```

```
(255, 0, 0, 255)  
(255, 0, 0, 255)  
(0, 0, 0, 255)  
(210, 105, 30, 255)  
(100, 149, 237, 255)
```

#Abrir imágenes

Para manipular una imagen con Pillow, primero necesitamos abrirla mediante la función *Image.open()*

```
!wget https://ihum.ai/static/logos/ISOTIPO.png -O logo.png #Traemos el logo de  
Humai desde la página oficial :)
```

```
--2021-11-10 17:32:48-- https://ihum.ai/static/logos/ISOTIPO.png  
Resolving ihum.ai (ihum.ai)... 18.190.92.144  
Connecting to ihum.ai (ihum.ai)|18.190.92.144|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 17855 (17K) [image/png]  
Saving to: 'logo.png'  
  
logo.png          100%[=====] 17,44K  --.-KB/s   in 0s  
  
2021-11-10 17:32:49 (613 MB/s) - 'logo.png' saved [17855/17855]
```

```
from PIL import Image  
  
logo = Image.open('logo.png')  
  
logo
```



Todas las rotaciones, cambios de tamaño, cortes, dibujos y otras tareas vamos a hacerlas llamando métodos de este objeto Image.

```
type(logo)
```

```
PIL.PngImagePlugin.PngImageFile
```

#Tamaño

Las dimensiones de una imagen se ordenan a partir de una *Box tuple* de cuatro coordenadas:

- Left: La coordenada *x* del borde izquierdo de la imagen.
- Top: La coordenada *y* del borde superior de la imagen.
- Right: La coordenada *x* de un pixel a la derecha del borde derecho de la imagen. Este entero debe ser mayor a Left.
- Bottom: La coordenada *y* de un pixel más abajo del borde inferior de la imagen. Este entero debe ser mayor a Top.

(Observemos que los enteros de Left y Top están incluidos, mientras que los de Right y Bottom están excluidos)

## 32. Métodos de manipulación

Con `size` podemos obtener el tamaño de nuestra imagen en una tupla de dos elementos:

```
logo.size
```

```
(300, 391)
```

Separamos la tupla en ancho y alto:

```
ancho, alto = logo.size  
print(ancho)  
print(alto)
```

```
300  
391
```

También podemos obtener información como el nombre del archivo de la imagen...

```
logo.filename
```

```
'logo.png'
```

...el formato...

```
logo.format
```

```
'PNG'
```

...o incluso la descripción del formato.

```
logo.format_description
```

```
'Portable network graphics'
```

Con el método `save` guardamos una copia, pasándole como parámetro un string que especifique el nombre del archivo y el formato deseado:

```
logo.save('muñequito_humai.png')
```

Con `crop()` podemos recortar un rectángulo específico dentro de la imagen:

```
recorte = logo.crop((115, 0, 205, 70))  
recorte
```



También podemos guardar el recorte realizado dentro de un archivo:

```
recorte.save('recorte.png')
```

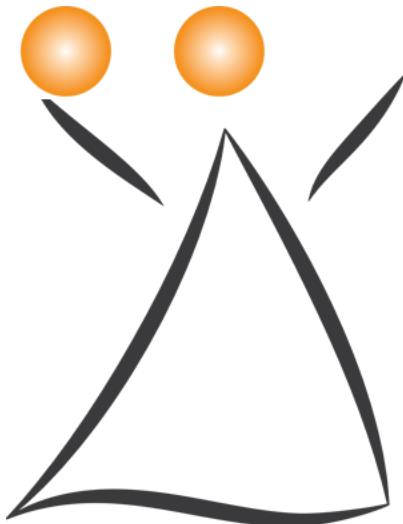
El método `copy()` nos permite generar un nuevo objeto `Image` igual a aquél desde el cual fue llamado. Esto resulta ventajoso si queremos manipular una imagen pero al mismo tiempo mantener una versión intacta de la misma:

```
logo2 = logo.copy()  
logo2
```



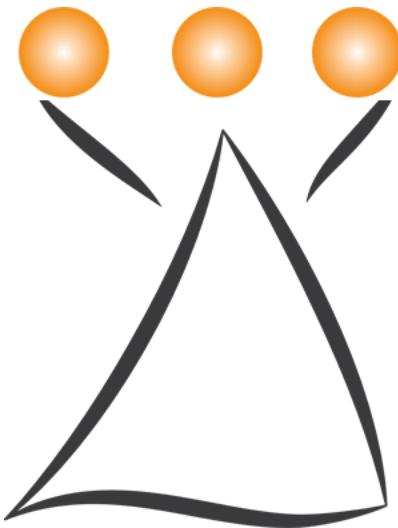
El método `paste()` nos permite pegar una imagen sobre otra. En este caso vamos a superponer el recorte que hicimos sobre nuestra copia:

```
logo2.paste(recorte, (0, 0))  
logo2
```



¡¡Ahora el logo de Humai tiene dos pelotitas!!! Podemos agregarle una más del otro lado, modificando el parámetro que determina la coordenada superior izquierda de la imagen a pegar:

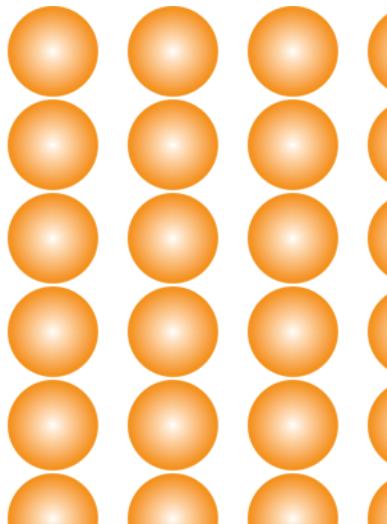
```
logo2.paste(recorte, (220, 0))  
logo2
```



Nota: Ni el método `copy()` ni el método `paste()` utilizan el portapapeles de nuestra computadora.

Otra cosa que podemos hacer es pegar nuestro recorte de manera que recubra toda la imagen mediante un bucle anidado dentro de otro, de la siguiente manera:

```
ancho_copia, alto_copia = logo2.size #Separamos ancho y alto de ambos objetos.  
ancho_recorte, alto_recorte = recorte.size  
  
for left in range(0, ancho_copia, ancho_recorte): #Para un rango del ancho de la  
    #imagen, con un intervalo del ancho del recorte...  
        for top in range(0, alto_copia, alto_recorte): #...y a su vez para un rango  
            #del alto de la imagen, con un intervalo del alto del recorte...  
                logo2.paste(recorte, (left, top)) #...pegar el recorte.  
  
logo2
```



Veamos algunos ejemplos un poco más complejos

```
!wget https://codersera.com/blog/wp-content/uploads/2021/06/Beginners-Guide-To-Python.jpeg -O img.jpeg
```

```
--2021-11-10 17:32:49-- https://codersera.com/blog/wp-content/uploads/2021/06/Beginners-Guide-To-Python.jpeg  
Resolving codersera.com (codersera.com)... 172.67.149.220, 104.21.29.227, 2606:4700:3034::ac43:95dc, ...  
Connecting to codersera.com (codersera.com)|172.67.149.220|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: unspecified [image/jpeg]  
Saving to: 'img.jpeg'  
  
img.jpeg [ <=> ] 17,16K ---KB/s in 0,009s  
  
2021-11-10 17:32:50 (1,80 MB/s) - 'img.jpeg' saved [17574]
```

```
from PIL import ImageDraw, ImageFont
```

```
# Leemos la imagen
py = Image.open('img.jpeg').convert('RGBA')

# la achicamos
py.thumbnail((250, 250))
py
```



```
def storify(image, color=None, coloroff=1, ratio=16 / 9):
    """Función para llevar una imagen a una proporción dada, completando con el
    color dominante.
    image:= Imagen PIL
    color:= Color a usar, si no se quiere el dominante,
    coloroff := offset del color para no tomar el más predominante,
    ratio := proporción a la que llevar la imagen
    """

    old_size = image.size
    # calculamos las nuevas dimensiones
    max_dimension, min_dimension = max(old_size), min(old_size)
    desired_size = (max_dimension, int(max_dimension * ratio))
    # coordenadas medias para pegar la imagen original
    x_mid = int(desired_size[0] / 2 - old_size[0] / 2)
    y_mid = int(desired_size[1] / 2 - old_size[1] / 2)
    position = (x_mid, y_mid)

    # si el color es None, buscamos el color más presente
    image = image.convert("RGBA")
    if color is None:
        pixels = image.getcolors(image.width * image.height) # get colors
        devuelve la cantidad de pixeles de cada color RGB
        sorted_pixels = sorted(pixels, key=lambda t: t[0])
        color = sorted_pixels[-coloroff][1][: -1]
    # creamos una nueva imagen de la dimensión deseada y pegamos la original en el
    centro
    blank_image = Image.new("RGB", desired_size, color=color)
    blank_image.paste(image, position, image)

    return blank_image, color, position
```

```
img, color, pos = storify(py, ratio=1)
```

```
img
```



```
img, color, pos = storify(py, color=None, ratio=16/9)
img
```



```
!wget -nc https://unket.s3.sa-east-1.amazonaws.com/static/Roboto-Regular.ttf -O font.ttf
```

```
--2021-11-10 17:37:52-- https://unket.s3.sa-east-1.amazonaws.com/static/Roboto-Regular.ttf
Resolving unket.s3.sa-east-1.amazonaws.com (unket.s3.sa-east-1.amazonaws.com)... 52.95.163.55
Connecting to unket.s3.sa-east-1.amazonaws.com (unket.s3.sa-east-1.amazonaws.com)|52.95.163.55|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 168260 (164K) [binary/octet-stream]
Saving to: 'font.ttf'

font.ttf          100%[=====] 164,32K   732KB/s    in 0,2s

2021-11-10 17:37:52 (732 KB/s) - 'font.ttf' saved [168260/168260]
```

```
def add_text_center(
    img,
    txt,
    y_offset=0,
    x_offset=0,
    font_size=50,
    color="black",
    font_ttf="font.ttf",
):
    """Función para pegar un texto en una imagen PIL.
    Por default, el texto se pega en el centro, con x e y offset se puede
    modificar.
    """

    draw = ImageDraw.Draw(img)
    # leemos la tipografía de un archivo ttf
    font = ImageFont.truetype(font_ttf, int(font_size))
    # vemos el tamaño resultante del texto para centrarlo
    size = font.getsize(txt)

    width, height = img.size
    text_x, text_y = size

    # por default el texto se centra, los offsets nos permiten correrlo del centro
    x = (width - text_x) / 2 + x_offset
    y = (height - text_y) / 2 + y_offset
    draw.text((x, y), txt, font=font, fill=color)

    return img
```

```
img, color, pos = storify(img, color='white')
img
```



```
for i in range(0, 100, 10):
    img = add_text_center(img, 'Automatización con Python',
                          font_size=30 - (i/10),
                          y_offset=-img.size[1]*0.4 - i,
                          color=(0+i*5, 0+i*5, 0+i*5))
```

```
logo = logo.convert('RGBA')
img = img.convert('RGBA')
```

```
logo.thumbnail((100, 100))
```

```
img.alpha_composite(logo, (180, 650))
```

```
img
```

# Automatización con Python



## 33. APIs

En esta sección incluimos materiales vinculados al consumo de APIs públicas.

1. APIs Geográficas: introducción a las APIs y protocolos básicos. Estudiamos APIs vinculadas a datos geográficos de Argentina. Clase elaborada por Agustín Benassi.

- Clase: [https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/1\\_APIs\\_Geograficas/clase-1.ipynb](https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/1_APIs_Geograficas/clase-1.ipynb)
- Ejercicios: [https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/1\\_APIs\\_Geograficas/clase-1-ejercicios.ipynb](https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/1_APIs_Geograficas/clase-1-ejercicios.ipynb)

1. APIs Series de Tiempo: en esta clase analizamos APIs con datos de series de tiempo. Vemos la API de Series de Tiempo de Argentina [<https://datospobar.github.io/series-tiempo-ar-api/>] y también las de FRED, Quandl y Banco Mundial. Clase elaborada por Agustín Benassi.

- Clase: [https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/2\\_APIs\\_Series\\_Tiempo/clase-2.ipynb](https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/2_APIs_Series_Tiempo/clase-2.ipynb)
- Ejercicios: [https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/2\\_APIs\\_Series\\_Tiempo/ejercicios/ejercicios.ipynb](https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/2_APIs_Series_Tiempo/ejercicios/ejercicios.ipynb)

Open in Colab

## 34. APIs geográficas

```
import requests
import pandas as pd
import geopandas as gpd
import helpers
import json
```

## 34.1. Requests

### 34.1.1. Interfaces de aplicaciones

¿Qué es una API? Los sistemas tienen distintos tipos de interfaces que permiten interactuar con ellos.

- **GUI (Graphical User Interface)**: El usuario clickea e interactúa con distintos objetos para ejecutar acciones y lograr sus objetivos. Las páginas web tienen interfaces gráficas.
- **API (Application Programming Interface)**: El usuario escribe líneas de código para interactuar con el sistema, ejecutar acciones y lograr sus objetivos.

Todos los sitios web tienen una interfaz gráfica con la que estamos acostumbrados a interactuar, y también tienen una interfaz de programación más o menos desarrollada, o más o menos expuesta, con la cual tal vez no estemos tan acostumbrados a interactuar.

Ejemplo:

Si van a <https://www.mercadolibre.com.ar/> pueden hacer click en un input box, escribir "pelotas", apretar "Enter" y el sistema les devolverá el resultado de una búsqueda. Si lo cambian por "pelotas futbol" y hacen "Enter" de nuevo, cambiará el resultado de la búsqueda. Esta es la **interfaz gráfica del sitio**.

Sin embargo también podrían cumplir el mismo objetivo sólo escribiendo distintas URLs en el navegador:

- <https://listado.mercadolibre.com.ar/pelotas>
- <https://listado.mercadolibre.com.ar/pelotas-futbol>

Esto es más parecido a lo que llamaríamos una **interfaz de programación**.

### 34.1.2. APIs REST

En la web, las interfaces de programación de uso más difundido son las APIs REST ([Representational state transfer](#)) que especifica protocolos y métodos para interactuar con los vastos recursos de internet, escribiendo líneas de código.

La web utiliza ampliamente el protocolo HTTP para interactuar con sus recursos. Este protocolo indica cómo estructurar un mensaje de texto que describa la petición (**request**) del usuario a un servidor. Hay distintos tipos de peticiones que un usuario puede realizar, algunas de ellas son:

- **GET**: Sigue una representación de un recurso alojado en el servidor.
- **POST**: Envía datos al servidor para crear un recurso nuevo.
- **PUT**: Crea o modifica un recurso del servidor.
- **DELETE**: Elimina un recurso del servidor.

Existen otros métodos que hoy no nos interesan.

### 34.1.3. GET request

Cada vez que vamos al navegador y escribimos la dirección de una página web, **estamos haciendo un GET request** a un servidor. Esto es una petición para adquirir el código de un recurso que queremos visualizar en el navegador.

La URL es la parte más importante de la definición de un GET request (aunque el navegador agrega otras cosas también, que no vemos) y nos permite cambiar la representación deseada de ese mismo recurso de distintas maneras:

- <https://deportes.mercadolibre.com.ar/pelotas-futbol> pide al servidor pelotas de fútbol.
- [https://deportes.mercadolibre.com.ar/pelotas-futbol\\_OrderId\\_PRICE](https://deportes.mercadolibre.com.ar/pelotas-futbol_OrderId_PRICE) pide al servidor pelotas de fútbol ordenadas por precio.

Cuando escribimos una URL en un navegador, la mayoría de las veces hacemos GET requests que devuelven código HTML (es el código que usa el navegador para renderizar una página web) pero las GET requests pueden devolver datos en otros formatos (por ejemplo en JSON y en CSV).

**Las APIs REST que definen GET requests capaces de devolver datos en formato JSON y CSV, son particularmente interesantes para enriquecer nuestras capacidades de análisis de datos.**

Ahora vamos a conocer algunas de ellas.

## 34.2. API Georef

La [API Georef](#) o, en su nombre más largo, el **Servicio de Normalización de Datos Geográficos de Argentina** es una API REST desarrollada y mantenida por el Estado Nacional de Argentina para la normalización de datos geográficos, que es capaz de:

- Georeferenciar una dirección
- Normalizar nombres de provincias, departamentos y otras entidades
- Devolver los nombres de provincias, departamentos y otras entidades que contienen a un par de coordenadas (georreferenciación inversa)
- Sirve de referencia, devolviendo las listas canónicas con nombres y códigos oficiales de estas entidades, en formato CSV, JSON, SHP y otros.

A continuación, vamos a ver algunos de los usos que tienen los recursos de esta API para potenciar nuestras capacidades de analizar datos.

### 34.2.1. Referencia

La API Georef contiene las listas con nombres y códigos oficiales de Argentina para, entre otras, las siguientes entidades:

- Provincias
- Departamentos
- Municipios
- Localidades
- Calles

Cada una de ellas se consulta usando un **recurso** de la API.

#### 34.2.1.1. Librería `requests` y formato JSON

Si estuviéramos desarrollando una aplicación web que contiene, por ejemplo, un formulario donde el usuario debe elegir el nombre de la provincia donde vive con el objeto de guardar la información en una base de datos, podríamos usar la API Georef para obtener esa lista.

La librería `requests` en python permite hacer todo tipo de requests APIs REST y es sencilla de utilizar.

```
response = requests.get("https://apis.datos.gob.ar/georef/api/provincias")  
  
# el status code de una request indica si esta fue realizada con éxito o no, y por  
qué  
# el código 200 indica que una request fue realizada exitosamente  
response.status_code  
  
200  
  
provincias = response.json()  
provincias
```

El método `response.json()` de la librería `requests` devuelve una respuesta de formato JSON, en un familiar diccionario de python.

```
provincias["provincias"][0]  
  
{'centroide': {'lat': -26.8753965086829, 'lon': -54.6516966230371},  
 'id': '54',  
 'nombre': 'Misiones'}
```

Las APIs tienen distintos parámetros que permiten modificar la consulta a un recurso determinado, estos se agregan al final usando un signo de interrogación `?` y separándolos con el carácter `ampersand &`.

- **Recurso:** [apis.datos.gob.ar/georef/api/provincias](https://apis.datos.gob.ar/georef/api/provincias)
- **Parámetros:** `?orden=nombre&aplanar`
- **URL completa:** [apis.datos.gob.ar/georef/api/provincias?orden=nombre&aplanar](https://apis.datos.gob.ar/georef/api/provincias?orden=nombre&aplanar)

El parámetro “orden” permite elegir un campo por el cual ordenar los resultados. El parámetro “aplanar” devuelve una estructura plana o no anidada de los resultados que a veces es más fácil de utilizar, dependiendo de lo que estemos haciendo.

Podés consultar los otros parámetros disponibles para el recurso “provincias” en [https://datospobar.github.io/georef-ar-api/open-api/#/Recursos/get\\_provincias](https://datospobar.github.io/georef-ar-api/open-api/#/Recursos/get_provincias)

**Ejercicio:** crea una lista ordenada alfabéticamente con tuplas que contengan id y nombre de cada provincia, que usarías para armar tu selector de provincias en una página web.

#### 34.2.1.2. `pandas` y formato CSV

Si estamos creando una tabla auxiliar o de consulta para obtener los nombres y códigos oficiales de las provincias, es posible que queramos descargarla en CSV o leerla directamente en un dataframe de pandas.

```
df = pd.read_csv('https://apis.datos.gob.ar/georef/api/provincias?formato=csv')
```

```
df
```

	provincia_id	provincia_nombre	provincia_centroide_lat	provincia_centroide_lon
0	54	Misiones	-26.875397	-54.651697
1	74	San Luis	-33.757726	-66.028130
2	70	San Juan	-30.865368	-68.889491
3	30	Entre Ríos	-32.058874	-59.201448
4	78	Santa Cruz	-48.815485	-69.955762
5	62	Río Negro	-40.405796	-67.229330
6	26	Chubut	-43.788623	-68.526759
7	14	Córdoba	-32.142933	-63.801753
8	50	Mendoza	-34.629887	-68.583123
9	46	La Rioja	-29.685776	-67.181736
10	10	Catamarca	-27.335833	-66.947682
11	42	La Pampa	-37.131554	-65.446655
12	86	Santiago del Estero	-27.782412	-63.252387
13	18	Corrientes	-28.774305	-57.801219
14	82	Santa Fe	-30.706927	-60.949837
15	90	Tucumán	-26.947800	-65.364758
16	58	Neuquén	-38.641758	-70.118571
17	66	Salta	-24.299134	-64.814463
18	22	Chaco	-26.386431	-60.765831
19	34	Formosa	-24.894973	-59.932441
20	38	Jujuy	-23.320078	-65.764252
21	2	Ciudad Autónoma de Buenos Aires	-34.614493	-58.445856
22	6	Buenos Aires	-36.676942	-60.558832
23	94	Tierra del Fuego, Antártida e Islas del Atlánt...	-82.521518	-50.742749

Notá que la respuesta viene “aplanada” por defecto, ya que el CSV es un formato tabular (no anidado).

**Ejercicio:** crea una tabla con la lista de departamentos de la provincia de Santa Fé. Para esto usa el recurso “departamentos” y el parámetro “provincia” para filtrarlo con los resultados de la provincia de Santa Fé.

Puedes usar tanto el id como el nombre de la provincia para filtrar.

### 34.2.1.3. geopandas y el formato SHP (shapefile)

La API georef también permite obtener las geometrías de las entidades para, por ejemplo, graficarlas en un mapa. Sólo debe cambiarse el formato por "shp".

Geopandas puede leer un shapefile directo de la URL, tal como hicimos con el CSV.

```
gdf = gpd.read_file('https://apis.datos.gob.ar/georef/api/departamentos?  
formato=shp&provincia=82')
```

```
gdf.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4159ab4fd0>
```

```
len(gdf)
```

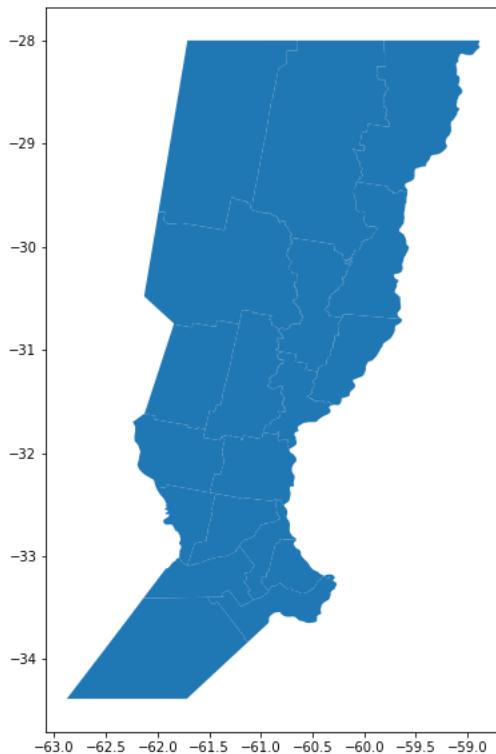
```
10
```

Por qué faltan departamentos? La API devuelve 10 resultados como máximo por defecto, pero se puede aumentar la cantidad de resultados con el parámetro "max" hasta 5000.

```
gdf = gpd.read_file('https://apis.datos.gob.ar/georef/api/departamentos?  
formato=shp&provincia=82&max=5000')
```

```
gdf.plot(figsize=(10, 10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4159dfc9d0>
```



**Ejercicio:** crea un mapa de los departamentos de la provincia de Buenos Aires. Y luego crea otro solamente de aquellos departamentos con nombres de santos.

Podés usar el parámetro "nombre" para filtrar los nombres de los departamentos que contengan la palabra "san".

### 34.2.2. Normalización

### 34.2.2.1. Métodos POST y consultas bulk

Los recursos se pueden usar para normalizar nombres de unidades territoriales mal escritos, corrigiendo con el nombre oficial y agregando el código oficial. Esto facilita enormemente la crusa de datos.

```
requests.get("https://apis.datos.gob.ar/georef/api/provincias?nombre=sant  
fe").json()
```

```
{'cantidad': 1,  
'inicio': 0,  
'parametros': {'nombre': 'sant fe'},  
'provincias': [{ 'centroide': {'lat': -30.7069271588117,  
'lon': -60.9498369430241},  
'id': '82',  
'nombre': 'Santa Fe'}],  
'total': 1}
```

Sin embargo, los datasets a normalizar generalmente tendrán muchas filas y el método GET en este caso no nos llevará muy lejos.. Consumiremos rápidamente la cuota de la API y no conseguiremos el objetivo.

Para esto podemos usar la versión “bulk” del recurso, y realizar una request POST.

Imaginemos que tenemos un dataset con varias filas a normalizar.

```
bioetanol = pd.read_csv('http://datos[minem].gob.ar/dataset/5ce77ad1-c729-42cd-  
a8b5-2407de005e5b/resource/0df1eeda-854b-44b0-8ea6-  
1a537f167fa4/download/bioetanol.csv')
```

```
bioetanol
```

	fecha	empresa	localidad	provincia	insumo	cupo_total_asignado
0	1/1/2012	ALCONOA S.R.L.	EL TABACAL	SALTA	CAÑA DE AZÚCAR	350
1	1/1/2012	BIO SAN ISIDRO S.A.	CAMPO SANTO	SALTA	CAÑA DE AZÚCAR	300
2	1/1/2012	BIOENERGÍA LA CORONA S.A.	CONCEPCIÓN	TUCUMÁN	CAÑA DE AZÚCAR	60
3	1/1/2012	BIOENERGÍA SANTA ROSA S.A.	LEÓN ROUGES	TUCUMÁN	CAÑA DE AZÚCAR	120
4	1/1/2012	BIOLEDESMA S.A.	LEDESMA	JUJUY	CAÑA DE AZÚCAR	400
...	...	...	...	...	...	...
967	3/1/2018	ENERGÍAS ECOLÓGICAS DEL TUCUMÁN S.A.	AGUILARES	TUCUMÁN	CAÑA DE AZÚCAR	1
968	3/1/2018	FRONTERITA ENERGÍA S.A.	FAMAILLÁ	TUCUMÁN	CAÑA DE AZÚCAR	1
969	3/1/2018	PROMAÍZ S.A.	ALEJANDRO ROCA	CÓRDOBA	MAÍZ	1
970	3/1/2018	RÍO GRANDE ENERGÍA S.A.	LA MENDIETA	JUJUY	CAÑA DE AZÚCAR	1
971	3/1/2018	VICENTÍN S.A.I.C.	AVELLANEDA	SANTA FE	MAÍZ	1

972 rows × 17 columns

Los métodos POST se usan cuando una request debe enviarle datos al servidor para cumplir su cometido. Generalmente las POST requests generan cambios en el servidor (por ejemplo, agregan un comentario a una nota periodística) pero también se usan cuando la consulta requiere parámetros más complejos o voluminosos.

El método POST permite enviar un JSON (por ejemplo) aparte de la URL.

```
data = {
    'provincias': [
        {
            'nombre': 'JUJUY',
            'max': 1,
            'campos': 'id,nombre'
        },
        {
            'nombre': 'TUCUMAN',
            'max': 1,
            'campos': 'id,nombre'
        },
        {
            'nombre': 'CORDOBA',
            'max': 1,
            'campos': 'id,nombre'
        },
        {
            'nombre': 'SALTA',
            'max': 1,
            'campos': 'id,nombre'
        }
    ]
}
```

En este diccionario, cada objeto en la lista de provincias es una consulta individual. Se pueden acumular hasta 1000 consultas en un solo POST request de esta API.

En todos los casos ponemos 'max' = 1 porque solo queremos el mejor match posible de cada caso, y elegimos que devuelva solo los campos que necesitamos, para agilizar la consulta.

```
resultado_provs = requests.post(
    "https://apis.datos.gob.ar/georef/api/provincias",
    json=data
).json()

resultado_provs
```

```
{'resultados': [{"cantidad': 1,
    'inicio': 0,
    'parametros': {'campos': ['id', 'nombre'], 'max': 1, 'nombre': 'JUJUY'},
    'provincias': [{'id': '38', 'nombre': 'Jujuy'}],
    'total': 1},
    {'cantidad': 1,
    'inicio': 0,
    'parametros': {'campos': ['id', 'nombre'], 'max': 1, 'nombre': 'TUCUMAN'},
    'provincias': [{'id': '90', 'nombre': 'Tucumán'}],
    'total': 1},
    {'cantidad': 1,
    'inicio': 0,
    'parametros': {'campos': ['id', 'nombre'], 'max': 1, 'nombre': 'CORDOBA'},
    'provincias': [{'id': '14', 'nombre': 'Córdoba'}],
    'total': 1},
    {'cantidad': 1,
    'inicio': 0,
    'parametros': {'campos': ['id', 'nombre'], 'max': 1, 'nombre': 'SALTA'},
    'provincias': [{'id': '66', 'nombre': 'Salta'}],
    'total': 1}]}}

resultado_provs['resultados'][0]['provincias']
```

```
[{'id': '38', 'nombre': 'Jujuy'}]
```

Para usar este método necesitamos generar el diccionario de una forma más flexible (no lo podemos hacer a mano!).

```
provincias_normalizar = list(bioetanol.provincia)
```

```
data = {
    'provincias': [
        {
            'nombre': provincia,
            'max': 1,
            'campos': 'id,nombre'
        }
        for provincia in provincias_normalizar
    ]
}
```

```
resultado_provs = requests.post(
    "https://apis.datos.gob.ar/georef/api/provincias",
    json=data
).json()
```

```
resultado_provs['resultados'][:10]
```

```
provincias_normalizadas = [
    resultado['provincias'][0]
    for resultado in resultado_provs['resultados']
]
```

```
provincias_normalizadas[:10]
```

```
[{'id': '66', 'nombre': 'Salta'},
 {'id': '66', 'nombre': 'Salta'},
 {'id': '90', 'nombre': 'Tucumán'},
 {'id': '90', 'nombre': 'Tucumán'},
 {'id': '38', 'nombre': 'Jujuy'},
 {'id': '90', 'nombre': 'Tucumán'},
 {'id': '90', 'nombre': 'Tucumán'},
 {'id': '90', 'nombre': 'Tucumán'},
 {'id': '38', 'nombre': 'Jujuy'},
 {'id': '66', 'nombre': 'Salta'}]
```

```
df_provincias_normalizadas = pd.DataFrame(provincias_normalizadas)
```

```
df_provincias_normalizadas
```

	<b>id</b>	<b>nombre</b>
0	66	Salta
1	66	Salta
2	90	Tucumán
3	90	Tucumán
4	38	Jujuy
...	...	...
967	90	Tucumán
968	90	Tucumán
969	14	Córdoba
970	38	Jujuy
971	82	Santa Fe

972 rows × 2 columns

```
bioetanol['provincia_id'] = df_provincias_normalizadas['id']
bioetanol['provincia_nombre'] = df_provincias_normalizadas['nombre']
```

```

def normalizar_unidades_territoriales(entidades, entidades_tipo):
    # genera el JSON con la lista de consultas a realizar
    data = {
        entidades_tipo: [
            {
                'nombre': nombre,
                'max': 1,
                'campos': 'id,nombre'
            }
            for nombre in entidades
        ]
    }

    # realiza la consulta
    resultados = requests.post(
        f"https://apis.datos.gob.ar/georef/api/{entidades_tipo}",
        json=data
    ).json()

    # parsea la respuesta
    entidades_normalizadas = [
        resultado[entidades_tipo][0]
        for resultado in resultados['resultados']
    ]

    return pd.DataFrame(entidades_normalizadas)

```

**Ejercicio:** usá la función para normalizar las localidades del dataset. Qué te parece que puede estar mal en el resultado, en este caso?

Tal vez necesitemos modificar un poco la función para este caso. Como no queremos pasar todo el tiempo con este tema, en el módulo `helpers.py` te dejamos ya armada una función para normalizar distintas entidades usando la API Georef.

```

localidades_normalizadas = helpers.normalizar_unidades_territoriales(
    bioetanol[['localidad', 'provincia']],
    entidades_tipo='localidades',
    nombre_campo='localidad',
    provincia_campo='provincia',
)

```

```
localidades_normalizadas
```

	<b>id</b>	<b>nombre</b>	<b>provincia_id</b>	<b>provincia_nombre</b>
0	66126030000	EL TABACAL	66	Salta
1	66049010000	CAMPO SANTO	66	Salta
2	90021050000	CONCEPCION	90	Tucumán
3	None	None	None	None
4	38035080002	PUEBLO LEDESMA	38	Jujuy
...	...	...	...	...
967	90077010001	AGUILARES	90	Tucumán
968	90028030002	FAMAILLA	90	Tucumán
969	14056010000	ALEJANDRO ROCA	14	Córdoba
970	38063090000	LA MENDIETA	38	Jujuy
971	82049020000	AVELLANEDA	82	Santa Fe

972 rows × 4 columns

```

bioetanol['localidad_id'] = df_provincias_normalizadas['id']
bioetanol['localidad_nombre'] = df_provincias_normalizadas['nombre']

```

### 34.2.3. Georreferenciación

En la API Georef, casi todos los recursos tienen una versión GET (más fácil de usar, pero que permite hacer pocas consultas) y una versión POST (más difícil de usar, pero que permite hacer más consultas y más eficientemente).

En las proximas dos secciones te vamos a presentar otras funcionalidades de la API Georef con el metodo GET correspondiente, y luego te ofreceremos una funcion ya escrita para hacer consultas masivas.

La funcion de **georreferenciaci<sup>on</sup>** permite normalizar y descomponer una direccion de Argentina en partes, asi como obtener sus coordenadas.

```
requests.get(  
    'https://apis.datos.gob.ar/georef/api/direcciones?direccion=balcarce  
50&provincia=02'  
).json()
```

No es obligatorio especificar una provincia, pero esto mejora mucho la precision del resultado. Tambien se puede especificar departamento u otros tipos de filtros. La API reconoce esquinas y diversas modalidades de especificar una direccion.

```
requests.get(  
    'https://apis.datos.gob.ar/georef/api/direcciones?direccion=santa fe y  
pueyrredon&provincia=02'  
).json()
```

```
{'cantidad': 1,  
 'direcciones': [{  
     'altura': {'unidad': None, 'valor': None},  
     'calle': {'categoria': 'AV',  
              'id': '0201401001740',  
              'nombre': 'AV SANTA FE'},  
     'calle_cruce_1': {'categoria': 'AV',  
                      'id': '0201401001675',  
                      'nombre': 'AV PUEYRREDON'},  
     'calle_cruce_2': {'categoria': None, 'id': None, 'nombre': None},  
     'departamento': {'id': '02014', 'nombre': 'Comuna 2'},  
     'localidad_censal': {'id': '020000010',  
                          'nombre': 'Ciudad Autonoma de Buenos Aires'},  
     'nomenclatura': 'AV SANTA FE (ESQUINA AV PUEYRREDON), Comuna 2, Ciudad Autonoma  
de Buenos Aires',  
     'piso': None,  
     'provincia': {'id': '02', 'nombre': 'Ciudad Autonoma de Buenos Aires'},  
     'ubicacion': {'lat': -34.5944399212952, 'lon': -58.4023531671055}}],  
 'inicio': 0,  
 'parametros': {'direccion': {'altura': {'unidad': None, 'valor': None},  
                             'calles': ['santa fe', 'pueyrredon'],  
                             'piso': None,  
                             'tipo': 'interseccion'},  
                'provincia': ['02']},  
 'total': 1}
```

```
requests.get(  
    'https://apis.datos.gob.ar/georef/api/direcciones?direccion=santa fe (esq  
pueyrredon)&provincia=02'  
).json()
```

```
{'cantidad': 1,  
 'direcciones': [{  
     'altura': {'unidad': None, 'valor': None},  
     'calle': {'categoria': 'AV',  
              'id': '0201401001740',  
              'nombre': 'AV SANTA FE'},  
     'calle_cruce_1': {'categoria': 'AV',  
                      'id': '0201401001675',  
                      'nombre': 'AV PUEYRREDON'},  
     'calle_cruce_2': {'categoria': None, 'id': None, 'nombre': None},  
     'departamento': {'id': '02014', 'nombre': 'Comuna 2'},  
     'localidad_censal': {'id': '020000010',  
                          'nombre': 'Ciudad Autonoma de Buenos Aires'},  
     'nomenclatura': 'AV SANTA FE (ESQUINA AV PUEYRREDON), Comuna 2, Ciudad Autonoma  
de Buenos Aires',  
     'piso': None,  
     'provincia': {'id': '02', 'nombre': 'Ciudad Autonoma de Buenos Aires'},  
     'ubicacion': {'lat': -34.5944399212952, 'lon': -58.4023531671055}}],  
 'inicio': 0,  
 'parametros': {'direccion': {'altura': {'unidad': None, 'valor': None},  
                             'calles': ['santa fe', 'pueyrredon'],  
                             'piso': None,  
                             'tipo': 'interseccion'},  
                'provincia': ['02']},  
 'total': 1}
```

**Ejercicio:** Genera un datafarme con direcciones para georreferenciar, que se pueda usar en la funcion de abajo.

```
# completa con direcciones y la provincia a la que pertenecen
direcciones = pd.DataFrame([
    {'direccion': 'balcarce 50', 'provincia': '02'},
    {'direccion': 'balcarce 50', 'provincia': '02'},
])
```

```
direcciones_georreferenciadas = helpers.georreferenciar_direcciones(
    direcciones[['direccion', 'provincia']],
    nombre_campo='direccion',
    provincia_campo='provincia',
)
direcciones_georreferenciadas
```

	direccion_normalizada	latitud	longitud	provincia_id	provincia_nombre
0	BALCARCE 50, Comuna 1, Ciudad Autónoma de Buenos Aires	-34.608341	-58.370885	02	Ciudad Autónoma de Buenos Aires
1	BALCARCE 50, Comuna 1, Ciudad Autónoma de Buenos Aires	-34.608341	-58.370885	02	Ciudad Autónoma de Buenos Aires
2	BALCARCE 50, Comuna 1, Ciudad Autónoma de Buenos Aires	-34.608341	-58.370885	02	Ciudad Autónoma de Buenos Aires
3	BALCARCE 50, Comuna 1, Ciudad Autónoma de Buenos Aires	-34.608341	-58.370885	02	Ciudad Autónoma de Buenos Aires
4	BALCARCE 50, Comuna 1, Ciudad Autónoma de Buenos Aires	-34.608341	-58.370885	02	Ciudad Autónoma de Buenos Aires

#### 34.2.4. Georreferenciación inversa

La funcionalidad de **georreferenciacion inversa** toma un punto de coordenadas y devuelve aquellas unidades territoriales que lo contienen.

No es esctictamente una funcion georreferenciacion inversa en realidad, ya que no devuelve una aproximacion de la direccion a la que pertenece ese punto de coordenadas, solo devuelve las unidades territoriales que contienen al punto.

Esta funcion es muy util cuando ya se tiene un dataset con coordenadas, pero queremos tener tambien las columnas de provincias, departamentos y municipios.

```
requests.get(
    'https://apis.datos.gob.ar/georef/api/ubicacion?
lat=-34.6037389&lon=-58.3815704'
).json()
```

```
{'parametros': {'lat': -34.6037389, 'lon': -58.3815704},
'ubicacion': {'departamento': {'id': '02007', 'nombre': 'Comuna 1'},
'lat': -34.6037389,
'lon': -58.3815704,
'municipio': {'id': None, 'nombre': None},
'provincia': {'id': '02', 'nombre': 'Ciudad Autónoma de Buenos Aires'}}}
```

```
requests.get(
    'https://apis.datos.gob.ar/georef/api/ubicacion?
lat=-32.9477132&lon=-60.6304658'
).json()
```

```
{'parametros': {'lat': -32.9477132, 'lon': -60.6304658},
'ubicacion': {'departamento': {'id': '82084', 'nombre': 'Rosario'},
'lat': -32.9477132,
'lon': -60.6304658,
'municipio': {'id': None, 'nombre': None},
'provincia': {'id': '82', 'nombre': 'Santa Fe'}}}
```

```
requests.get(
    'https://apis.datos.gob.ar/georef/api/ubicacion?
lat=-38.0048387&lon=-57.5483175'
).json()
```

```
{'parametros': {'lat': -38.0048387, 'lon': -57.5483175},
'ubicacion': {'departamento': {'id': '06357', 'nombre': 'General Pueyrredón'},
'lat': -38.0048387,
'lon': -57.5483175,
'municipio': {'id': '060357', 'nombre': 'General Pueyrredón'},
'provincia': {'id': '06', 'nombre': 'Buenos Aires'}}}
```

Por ejemplo, si tenemos las coordenadas de todos los aeropuertos del país, podemos agregarle fácilmente los códigos y nombres oficiales de las provincias, departamentos y municipios que los contienen.

```
aeropuertos = pd.read_csv(
    'https://servicios.transporte.gob.ar/gobierno_abierto/descargar.php?
t=aeropuertos&d=detalle',
    sep=';',
)
```

```
aeropuertos
```

	local	oaci	iata	tipo	denominacion	coordenadas	latitud	longitud
0	ACB	NaN	NaN	Aeródromo	CORONEL BOGADO/AGROSERVICIOS	33°16'20"S 60°34'14"W	-60.5701°S -57.5483175°W	
1	ACH	NaN	NaN	Aeródromo	GENERAL ACHA	37°24' 6"S 64°36'49"W	-64.6131°S -57.5483175°W	
2	ACM	NaN	NaN	Aeródromo	ARRECIFES/LA CURA MALAL	34° 4'33"S 60° 8'30"W	-60.1411°S -57.5483175°W	
3	ADO	SAWD	PUD	Aeródromo	PUERTO DESEADO	47°44' 6"S 65°54'15"W	-65.9041°S -57.5483175°W	
4	ADT	NaN	NaN	Aeródromo	BANDERA/AGROSERVICIOS DOÑA TERESA	28°51'19"S 62°15'53"W	-62.2641°S -57.5483175°W	
...	...	...	...	...	...	...	...	...
688	YOS	SAZH	OYO	Aeródromo	TRES ARROYOS	38°23' 8"S 60°19'39"W	-60.3271°S -57.5483175°W	
689	YPY	NaN	NaN	Aeródromo	YAPEYÚ	29°21'32"S 56°47'45"W	-56.7951°S -57.5483175°W	
690	ZAP	SAHZ	APZ	Aeródromo	ZAPALA	38°58'33"S 70° 6'48"W	-70.1131°S -57.5483175°W	
691	ZLM	NaN	NaN	Aeródromo	BELL VILLE/LA ZULEMA	32°29'23"S 62°40'17"W	-62.6711°S -57.5483175°W	
692	ZUL	SAZA	NaN	Aeródromo	AZUL	36°50'12"S 59°52'51"W	-59.8801°S -57.5483175°W	

693 rows × 23 columns

Notemos que el dataset tiene un error! Los nombres de las columnas latitud y longitud, están invertidos.

```
lat = aeropuertos.iloc[0]['latitud']
lon = aeropuertos.iloc[0]['longitud']

requests.get(
    f'https://apis.datos.gob.ar/georef/api/ubicacion?lat={lat}&lon={lon}'
).json()
```

```
{'parametros': {'lat': -60.57066, 'lon': -33.27226},
'ubicacion': {'departamento': {'id': None, 'nombre': None},
'lat': -60.57066,
'lon': -33.27226,
'municipio': {'id': None, 'nombre': None},
'provincia': {'id': None, 'nombre': None}}}
```

```
lat = aeropuertos.iloc[0]['longitud']
lon = aeropuertos.iloc[0]['latitud']

requests.get(
    f'https://apis.datos.gob.ar/georef/api/ubicacion?lat={lat}&lon={lon}'
).json()
```

```
{'parametros': {'lat': -33.27226, 'lon': -60.57066},
'ubicacion': {'departamento': {'id': '82084', 'nombre': 'Rosario'},
'lat': -33.27226,
'lon': -60.57066,
'municipio': {'id': '823393', 'nombre': 'Coronel Bogado'},
'provincia': {'id': '82', 'nombre': 'Santa Fe'}}}
```

Alcanza con notar esto para poder especificar los nombres invertidos en la funcion.

```
unidades_territoriales = helpers.ubicar_coordenadas(
    aeropuertos[['latitud', 'longitud']],
    'longitud', 'latitud'
)
unidades_territoriales
```

	<b>provincia_id</b>	<b>provincia_nombre</b>	<b>departamento_id</b>	<b>departamento_nombre</b>	<b>municipio_id</b>	<b>municipio_nombre</b>
0	82	Santa Fe	82084	Rosario	823393	Coronel Bogado
1	42	La Pampa	42154	Utracán	420000	N
2	06	Buenos Aires	06077	Arrecifes	060000	N
3	78	Santa Cruz	78014	Deseado	780000	N
4	86	Santiago del Estero	86077	General Taboada	860000	N
...	...	...	...	...	...	...
688	06	Buenos Aires	06833	Tres Arroyos	060000	N
689	18	Corrientes	18147	San Martín	180000	N
690	58	Neuquén	58112	Zapala	580000	N
691	14	Córdoba	14182	Unión	140000	N
692	06	Buenos Aires	06049	Azul	060000	N

693 rows × 6 columns

**Ejercicio:** ubica las coordenadas de las estaciones de medicion de recursos hidricos (<https://datos.mininterior.gob.ar/dataset/93913deb-d22e-4c0d-bca3-f465e7b2bb94/resource/26d292a4-4d7f-48e1-872a-50a0dca35386/download/estacioneslatlong.csv>) en sus unidades territoriales. Tene en cuenta que este dataset **esta codificado en 'latin1'**. Tenes que especificar la codificacion en la llamada pd.read\_csv() para que lo lea sin errores.

### 34.3. API Nominatim

<https://nominatim.org/>

Por supuesto, la API Georef del Estado Nacional no es la unica API de georreferenciacion que podes usar, existen muchas otras con distintos alcances y funcionalidades.

La API de Nominativa, utiliza la base de datos de la iniciativa Open Street Map para proveer un servicio abierto de georreferenciacion con alcance global.

Es importante que si usas esta API regularmente, leas los [terminos y condiciones](#). Entre ellas no permite hacer mas de 1 request por segundo, debe citarse la fuente y no se permite el uso para geocodificacion bulk. Esta API es para usar en forma esporadica, pero no en produccion (salvo que se instale la API en infraestructura propia).

Podes leer la documentacion de uso de la API, aca: <https://nominatim.org/release-docs/develop/api/Overview/>

#### 34.3.1. Busqueda de objetos en el espacio

Uno de los usos de la API es buscar objetos en el espacio, funciona de manera parecida a como a veces usamos Google Maps. Le podemos decir 'escuelas en berlin', 'hospitales en montevideo' y otras cosas parecidas, y devolvera una lista de resultados de lo que encuentre.

```
# hospitales en Montevideo
requests.get(
    'https://nominatim.openstreetmap.org/?'
    'addressdetails=1&q=hospitales+en+montevideo+uruguay&format=json&limit=3'
).json()
```

**Ejercicio:** genera una lista de nombres y direcciones de hospitales en la Ciudad de Buenos Aires, a partir de la API de Nominatim.

### 34.3.2. Georreferenciaciacion inversa

El recurso /reverse permite recuperar una direccion, a partir de un par de coordenadas.

```
requests.get(
    'https://nominatim.openstreetmap.org/reverse?'
    'format=jsonv2&lat=-34.6083411208197&lon=-58.37088525365451'
).json()
```

```
{'place_id': 14124008,
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0.
 https://osm.org/copyright',
 'osm_type': 'way',
 'osm_id': 208809079,
 'lat': '-34.6090457',
 'lon': '-58.3709199',
 'place_rank': 30,
 'category': 'place',
 'type': 'house',
 'importance': -0.1,
 'address_type': 'place',
 'name': None,
 'display_name': '302, Avenida Hipólito Yrigoyen, Monserrat, Buenos Aires, Ciudad Autónoma de Buenos Aires, Comuna 1, C1086AAD, Argentina',
 'address': {'house_number': '302',
 'road': 'Avenida Hipólito Yrigoyen',
 'suburb': 'Monserrat',
 'city': 'Buenos Aires',
 'state_district': 'Comuna 1',
 'postcode': 'C1086AAD',
 'country': 'Argentina',
 'country_code': 'ar'},
 'boundingbox': ['-34.6090957', '-34.6089957', '-58.3709609', '-58.3708609']}
```

**Ejercicio:** usa alguna de las coordenadas de las estaciones de medicion hidricas, para encontrar su direccion con este metodo. Funciona? Que resultados da esta API para esos casos?

### 34.3.3. Chequea el estado del servicio

Si la API no parece funcionar, podes asegurarte consultando si el servicio esta funcionando en este momento o no.

```
requests.get(
    'https://nominatim.openstreetmap.org/status.php?format=json'
).json()
```

```
{"status": 0, "message": "OK", "data_updated": "2020-07-30T20:04:01+00:00"}
```

## 34.4. API normalización GCBA

<https://usig.buenosaires.gob.ar/apis/>

Si lo que necesitas hacer tiene un alcance acotado al Area Metropolitana de Buenos Aires (AMBA), tambien podes usar la API de normalizacion de direcciones del AMBA del Gobierno de la Ciudad de Buenos Aires. Una ventaja que tiene este servicio, es que tambien lo podes instalar como una libreria de python para no necesitar internet ni tener limite de cuotas (<https://github.com/usig/normalizador-amba>).

### 34.4.1. Georreferenciaciacion

```

requests.get(
    'https://ws.usig.buenosaires.gob.ar/rest/normalizar_y_geocodificar_direcciones?
    calle=balcarce&altura=50&desambiguar=1'
).json()

```

```

{'Normalizacion': {'TipoResultado': 'DireccionNormalizada',
    'DireccionesCalleAltura': {'direcciones': [{'CodigoCalle': '2014',
        'Calle': 'BALCARCE',
        'Altura': '50'}]}, 
    'DireccionesCalleCalle': {'direcciones': []}}, 
    'GeoCodificacion': {'x': '108487.447694', 'y': '102343.772587'}}
}

```

La API de normalizacion de la USIG del GCBA devuelve coordenadas en el sistema de proyeccion Buenos Aires Gauss Krueguer, que es mas preciso para trabajar dentro del espacio geografico de la CABA pero tambien menos facil de usar para otros usos.

Hay otro recurso de la API que transforma esas coordenadas en EPSG 4326 (el sistema de coordenadas al que estamos mas acostumbrados).

```

requests.get(
    'https://ws.usig.buenosaires.gob.ar/rest/convertir_coordenadas?
    x=108487.447694&y=102343.772587&output=lonlat'
).json()

```

```

{'tipo_resultado': 'Ok', 'resultado': {'x': '-58.370769', 'y': '-34.608107'}}
}

```

#### 34.4.2. Datos utiles

Otro metodo de las APIs de USIG devuelve datos utiles del contexto de una direccion. Es parecido al de 'ubicacion' de la API Georef, pero tiene otras areas / unidades territoriales que contienen al punto, y son propias del AMBA.

```

requests.get(
    'https://ws.usig.buenosaires.gob.ar/datos_utiles?calle=balcarce&altura=50'
).json()

```

```

{'comuna': 'Comuna 1',
    'barrio': 'Monserrat',
    'comisaria': '2',
    'area_hospitalaria': 'HTAL. DR. C. ARGERICH',
    'region_sanitaria': 'I (Este)',
    'distrito_escolar': 'Distrito Escolar IV',
    'comisaria_vecinal': '1D',
    'seccion_catastral': '02',
    'distrito_economico': '',
    'codigo_de_planeamiento_urbano': '',
    'partido_amba': '',
    'localidad_amba': '',
    'codigo_postal': '1064',
    'codigo_postal_argentino': 'C1064AAB'}
}

```

#### 34.5. API Transporte GCBA

<https://www.buenosaires.gob.ar/desarrollourbano/transporte/apitransporte>

La API unificada de transporte del GCBA ofrece acceso en tiempo real a las localizaciones de subtes, colectivos, trenes, monopatines y otros muchos datos sobre la movilidad en la Ciudad de Buenos Aires. Es gratuita pero requiere registrarse y usar credenciales para cada consulta.

Una vez registrado, tenes que guardar tus credenciales en un archivo JSON `api_transportecreds.json` en el directorio de este notebook, o copiarlas directamente en la URL de la llamada.

```

# carga credenciales
with open('api_transportecreds.json', 'r') as f:
    creds = json.load(f)

```

```

client_id = creds['client_id']
client_secret = creds['client_secret']

```

```
colectivos_ahora = requests.get(  
    f'https://apitransporte.buenosaires.gob.ar/colectivos/vehiclePositionsSimple?  
    client_id={client_id}&client_secret={client_secret}'  
).json()  
colectivos_ahora[:5]
```

```
gdf = gpd.GeoDataFrame(colectivos_ahora)
```

```
gdf['geometry'] = gpd.points_from_xy(gdf.longitude, gdf.latitude)
```

```
gdf.plot(figsize=(10, 10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4158f6f150>
```



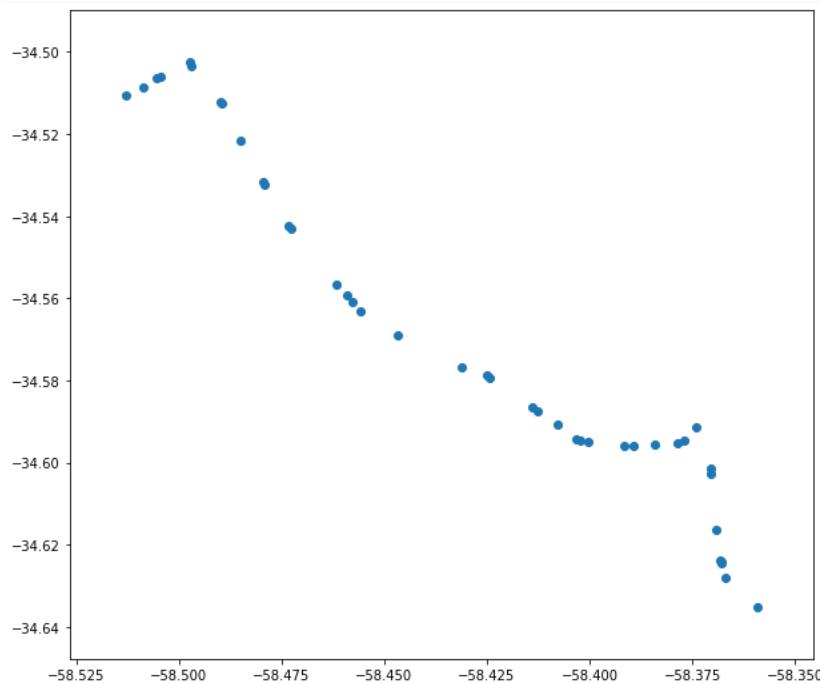
```
lineas = gdf.groupby(['route_short_name']).count()
```

```
lineas.sort_values('id', ascending=False).head(20)
```

route_short_name	route_id	latitude	longitude	speed	timestamp	id	direction	agen
343A	78	78	78	78	78	78	78	78
329TR1	57	57	57	57	57	57	57	57
53C	48	48	48	48	48	48	48	48
12A	47	47	47	47	47	47	47	47
41A	44	44	44	44	44	44	44	44
92A	42	42	42	42	42	42	42	42
152A	41	41	41	41	41	41	41	41
67A	36	36	36	36	36	36	36	36
10A	33	33	33	33	33	33	33	33
106A	33	33	33	33	33	33	33	33
707ROJ	31	31	31	31	31	31	31	31
102A	30	30	30	30	30	30	30	30
343B	29	29	29	29	29	29	29	29
124A	29	29	29	29	29	29	29	29
17A	29	29	29	29	29	29	29	29
96G	29	29	29	29	29	29	29	29
151A	29	29	29	29	29	29	29	29
118A	28	28	28	28	28	28	28	28
257A	28	28	28	28	28	28	28	28
59A	27	27	27	27	27	27	27	27

```
gdf[gdf.route_short_name == '152A'].plot(figsize=(10, 10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4158f13bd0>
```



```
<a href="https://colab.research.google.com/github/institutohumai/cursos-python/blob/master/APIs/1_APIs_Geograficas/clase-1-ejercicios.ipynb"> <img src='https://colab.research.google.com/assets/colab-badge.svg' /> </a>
<div align="center"> Recordá abrir en una nueva pestaña </div>
```

## 35. APIs Geográficas: Ejercicios

```
import requests
import pandas as pd
import geopandas as gpd
from urllib.parse import urlencode, urljoin
import matplotlib
import helpers
%matplotlib inline
```

### 35.1. Ejercicio 1: API Georef

- Genera una lista en un dataframe de todas las calles que se llaman “San Martin” en la Argentina. Cuantas son?
- Hace un mapa de las representaciones diplomáticas extranjeras en la Argentina. Para esto descarga las direcciones del dataset “[Guía Diplomática](#)”, georreferencia las, crea un GeoDataframe con ellas y mapealas sobre una capa de provincias de Argentina. Sugerencias:
  - Usa `pd.read_excel()` directamente con la URL de descarga de la lista de representaciones extranjeras.
  - Cuando descargas las embajadas, elimina aquellas que no tengan dirección, porque hace fallar la georreferenciación  
`(embajadas.dropna(subset=['representacion_direccion'])`

### 35.2. Ejercicio 2: API Nominatim

Crea distintos mapas que tengan al mapa de las comunas de la Ciudad de Buenos Aires como base, y sobre el dibujen los puntos de coordenadas de:

- Los 10 restaurantes mas cercanos al centroide del barrio de Almagro
- Los 10 hospitales mas cercanos al centroide de Villa Urquiza
- Las 10 escuelas mas cercanas al centroide de Villa Devoto

Sugerencia: crea una función “buscar” que tome un texto de búsqueda y haga la consulta a la API de Nominatim, y otra función “mapear” que tome un texto de búsqueda y haga el mapa de los puntos sobre las comunas de la CABA. Usa la función `mapear()` para hacer los 3 mapas solicitados.

## 36. APIs de Series de Tiempo

Esta clase presume un conocimiento basico de lo que es una API rest, del uso de la libreria `requests` y de `pandas`. Para una breve introducción a APIs REST se recomienda revisar primero la [clase 1 de APIs](#).

La siguiente celda se usa para instalar en el entorno de ejecución las librerías que no vienen pre instaladas en Google Colab.

```
!pip install markdown
!pip install pdfkit
!pip install requests
!pip install arrow
```

```
Looking in indexes: https://nexus.corp.indeed.com/repository/pypi/simple
Requirement already satisfied: markdown in
/home/abenassi/anaconda3/lib/python3.8/site-packages (3.2.2)
Looking in indexes: https://nexus.corp.indeed.com/repository/pypi/simple
Requirement already satisfied: pdfkit in
/home/abenassi/anaconda3/lib/python3.8/site-packages (0.6.1)
Looking in indexes: https://nexus.corp.indeed.com/repository/pypi/simple
Requirement already satisfied: requests in
/home/abenassi/anaconda3/lib/python3.8/site-packages (2.24.0)
Requirement already satisfied: urllib3!=1.25.0,!>=1.25.1,<1.26,>=1.21.1 in
/home/abenassi/anaconda3/lib/python3.8/site-packages (from requests) (1.25.9)
Requirement already satisfied: idna<3,>=2.5 in
/home/abenassi/anaconda3/lib/python3.8/site-packages (from requests) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
/home/abenassi/anaconda3/lib/python3.8/site-packages (from requests) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in
/home/abenassi/anaconda3/lib/python3.8/site-packages (from requests) (2020.6.20)
Looking in indexes: https://nexus.corp.indeed.com/repository/pypi/simple
Requirement already satisfied: arrow in
/home/abenassi/anaconda3/lib/python3.8/site-packages (0.16.0)
Requirement already satisfied: python-dateutil>=2.7.0 in
/home/abenassi/anaconda3/lib/python3.8/site-packages (from arrow) (2.8.1)
Requirement already satisfied: six>=1.5 in
/home/abenassi/anaconda3/lib/python3.8/site-packages (from python-dateutil>=2.7.0->arrow) (1.15.0)
```

```
import requests
import pandas as pd
import arrow
import json

# nos servirán para generar PDFs
import markdown
import pdfkit

# permite que los gráficos se vean directamente en el notebook
%matplotlib inline

# aplica una hoja de estilos para los gráficos
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# establece un alto y ancho default para todos los graficos
plt.rcParams['figure.figsize'] = (12, 8)
```

### 36.1. Series de tiempo

Las series de tiempo son datos estructurados en forma tal que existe una **variable numérica que evoluciona en el tiempo, a intervalos regulares**.

El PBI, la inflación, las exportaciones, la temperatura, la cantidad de llamadas a un call center... son todas variables que se pueden tratar como series de tiempo. Lo distintivo de esta estructura de datos es que tiene una sola dimensión de apertura que configura su índice o clave primaria (aquella que permite devolver una sola fila de la tabla): **el tiempo**.

### 36.2. API Series de Tiempo

La [API Series de Tiempo de la Republica Argentina](#) es una API REST desarrollada y mantenida por el Estado Nacional de Argentina para la consulta de estadísticas en formato de series de tiempo. Contiene mas de 25 mil series publicadas por mas de 10 organismos publicos, tanto de series historicas como actualizadas.

La API permite:

- [Buscar series](#) por texto. Tambien se pueden buscar en el sitio web de [datos.gob.ar](https://datos.gob.ar): <https://datos.gob.ar/series>
- Cambiar la frecuencia (por ejemplo: convertir series diarias en mensuales)
- Elegir la funcion de agregacion de valores, usada en el cambio de frecuencia (una serie se puede convertir de diaria a mensual promediando, sumando, sacando el maximo, el minimo, el ultimo valor del periodo, etc).
- Filtrar por rango de fechas
- Elegir el formato (CSV o JSON)
- Cambiar configuracion del CSV (caracter separador, caracter decimal)

En <https://datos.gob.ar/series> podés buscar series de tiempo publicadas por distintos organismos de la Administración Pública Nacional en Argentina y usar el link al CSV para leerlos directamente desde python con pandas.

Tambien podes **buscar los ids de las series de interes** y juntarlos en la misma consulta para armar una tabla de hasta 40 series.

En esta clase vamos a ver como se usa, y algunos ejemplos de cosas utiles que podes hacer con ella en python.

### 36.2.1. Parametros del recurso `/series`

La llamada a la API de series tiene la misma estructura que el sitio web de busqueda y consulta. El unico parametro obligatorio es `ids`, que debe contener los ids de 1 o mas series separados con comas.

Por ejemplo:

- URL web: [https://datos.gob.ar/series/api/series/?ids=105.1\\_I2P\\_2016\\_M\\_13,105.1\\_I2CPC\\_2016\\_M\\_27](https://datos.gob.ar/series/api/series/?ids=105.1_I2P_2016_M_13,105.1_I2CPC_2016_M_27) Desde esta UI web podés copiar la URL a un archivo CSV o JSON que tiene la evolucion del precio de la papa y de la picada común
- URL API en CSV: [https://apis.datos.gob.ar/series/api/series/?ids=105.1\\_I2P\\_2016\\_M\\_13,105.1\\_I2CPC\\_2016\\_M\\_27&limit=5000&format=csv](https://apis.datos.gob.ar/series/api/series/?ids=105.1_I2P_2016_M_13,105.1_I2CPC_2016_M_27&limit=5000&format=csv)
- URL API en JSON: [https://apis.datos.gob.ar/series/api/series/?ids=105.1\\_I2P\\_2016\\_M\\_13,105.1\\_I2CPC\\_2016\\_M\\_27&limit=5000&format=json](https://apis.datos.gob.ar/series/api/series/?ids=105.1_I2P_2016_M_13,105.1_I2CPC_2016_M_27&limit=5000&format=json)

Como podes ver, la URL es practicamente igual agregando `apis` al principio, y especificando el formato en el que se desea la respuesta.

En pandas podes leer directamente la URL en CSV.

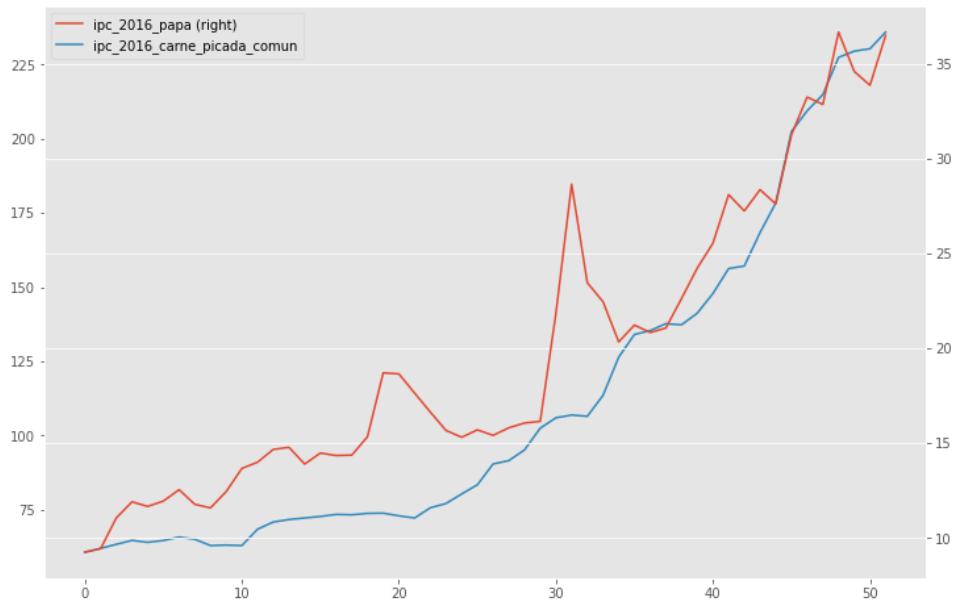
```
df = pd.read_csv("https://apis.datos.gob.ar/series/api/series/?ids=105.1_I2P_2016_M_13,105.1_I2CPC_2016_M_27&limit=5000&format=csv")
```

```
df.tail()
```

	indice_tiempo	ipc_2016_papa	ipc_2016_carne_picada_comun
47	2020-03-01	32.84	214.88
48	2020-04-01	36.65	227.42
49	2020-05-01	34.58	229.53
50	2020-06-01	33.85	230.34
51	2020-07-01	36.48	235.95

```
df.plot(secondary_y="ipc_2016_papa")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3b02ffac0>
```



#### 36.2.1.1. Frecuencia (`collapse` y `collapse_aggregation`)

La API automaticamente elige la frecuencia de mayor desagregacion temporal posible (la frecuencia mas alta), dadas las series solicitadas. Si en la misma llamada se solicita una serie diaria y una mensual, la mayor frecuencia posible es mensual.

Vamos a guardar la base de la llamada a la API para facilitar las cosas. Para estos ejemplos, siempre vamos a querer usar `format=csv` y `last=5000` (la API no devuelve *todos* los valores disponibles por default, `limit` permite elegir hasta los primeros 5000 valores y `last` permite elegir hasta los ultimos 5000 valores).

```
BASE_API = 'https://apis.datos.gob.ar/series/api/series/?format=csv&last=5000'
```

**Tipo de cambio del Banco Nacion (canal electronico, venta, valor de cierre a las 15hs).** Serie diaria.

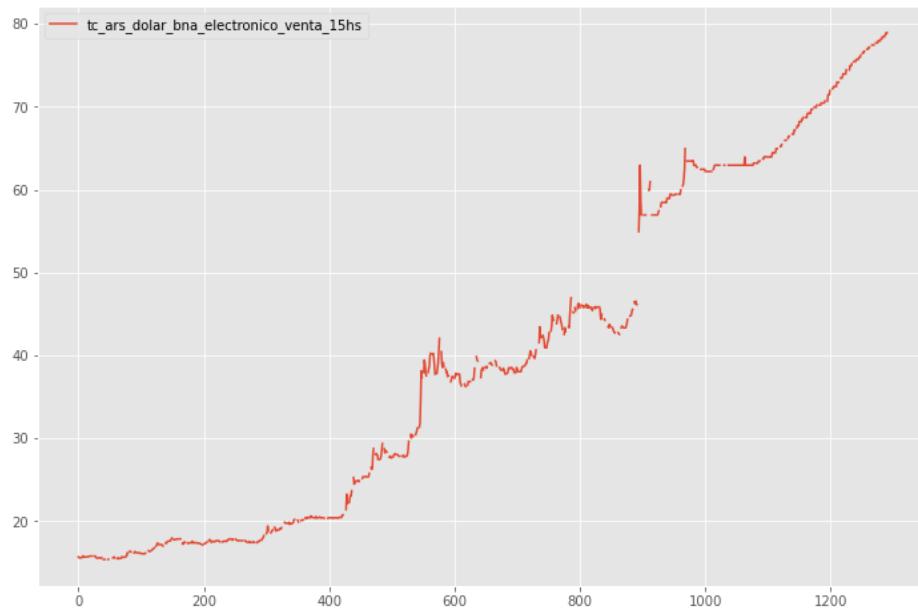
[https://datos.gob.ar/series/api/series/?ids=tc\\_usd\\_bna\\_ev15](https://datos.gob.ar/series/api/series/?ids=tc_usd_bna_ev15)

```
tc_bna_id = "tc_usd_bna_ev15"
tc_bna_api = f'{BASE_API}&ids={tc_bna_id}'
```

**Nota:** si no lo usaste antes, desde la version 3.7 de python se pueden pasar variables directamente en un string de python usando la `f` al principio y los `{}` para separar lo que es codigo, de lo que es texto.

```
pd.read_csv(tc_bna_api).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af83ae80>
```



Podemos adivinar que la serie

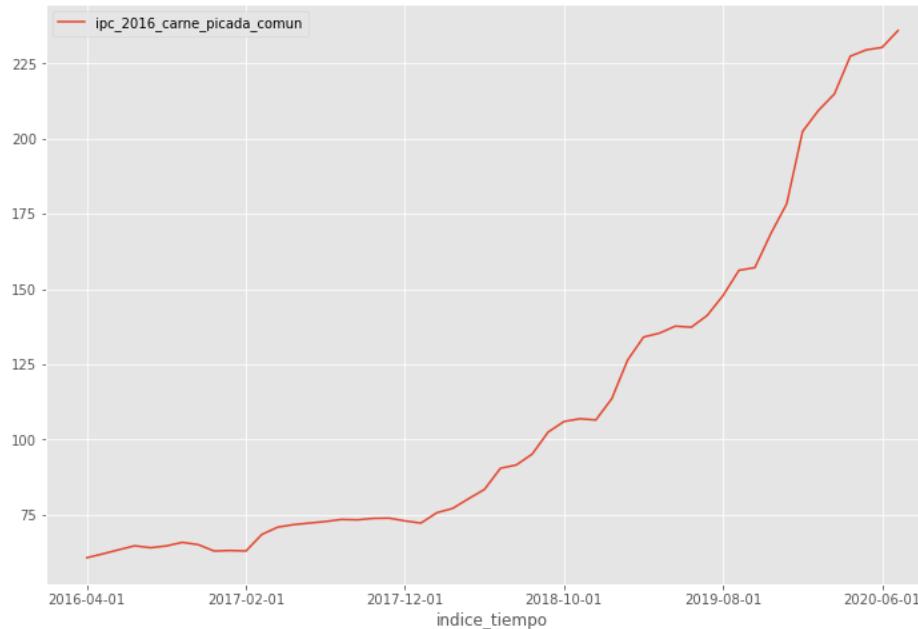
**Precio del kilo de picada comun.** Serie mensual.

[https://datos.gob.ar/series/api/series/?ids=105.1\\_I2CPC\\_2016\\_M\\_27](https://datos.gob.ar/series/api/series/?ids=105.1_I2CPC_2016_M_27)

```
picada_id = "105.1_I2CPC_2016_M_27"  
picada_api = f"{BASE_API}&ids={picada_id}"
```

```
pd.read_csv(picada_api).set_index('indice_tiempo').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3afa2a460>
```

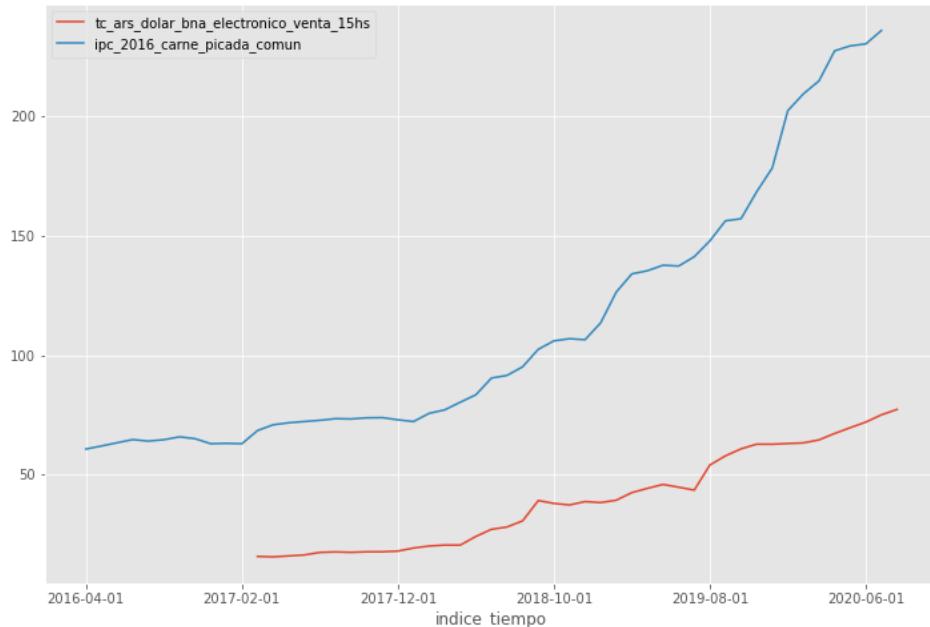


Ahora tratemos de juntar las dos series en la misma llamada, y veamos que pasa.

```
tc_picada_api = f"{BASE_API}&ids={tc_bna_id},{picada_id}"
```

```
pd.read_csv(tc_picada_api).set_index('indice_tiempo').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af9f3fd0>
```



La API convirtió la serie de tipo de cambio en **mensual** y aplicó un **promedio**, para poder tener las dos series en la misma tabla.

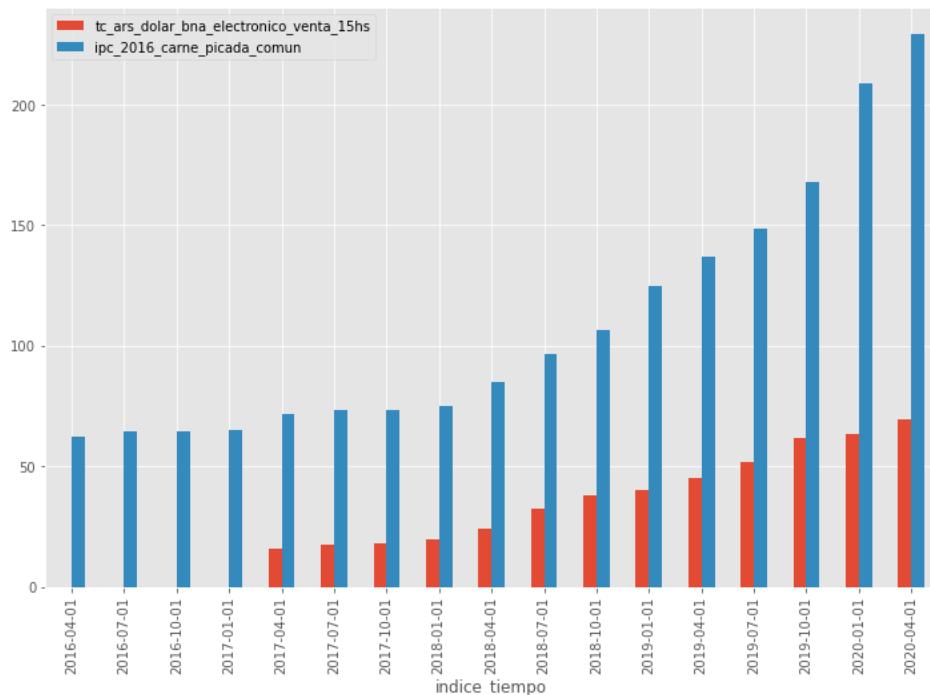
El parámetro `collapse` aplicado a esta consulta nos permite ir hacia frecuencias más bajas, pero nunca más altas. Las frecuencias disponibles son:

- `year`: Muestra datos agregados anualmente.
- `semester`: Muestra datos agregados semestralmente.
- `quarter`: Muestra datos agregados trimestralmente.
- `month`: Muestra datos agregados mensualmente.
- `week`: Muestra datos agregados semanalmente.
- `day`: Muestra datos agregados diariamente.

```
tc_picada_q_api = f'{BASE_API}&ids={tc_bna_id},{picada_id}&collapse=quarter'
```

```
pd.read_csv(tc_picada_q_api).set_index('indice_tiempo').plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3afa9deb0>
```



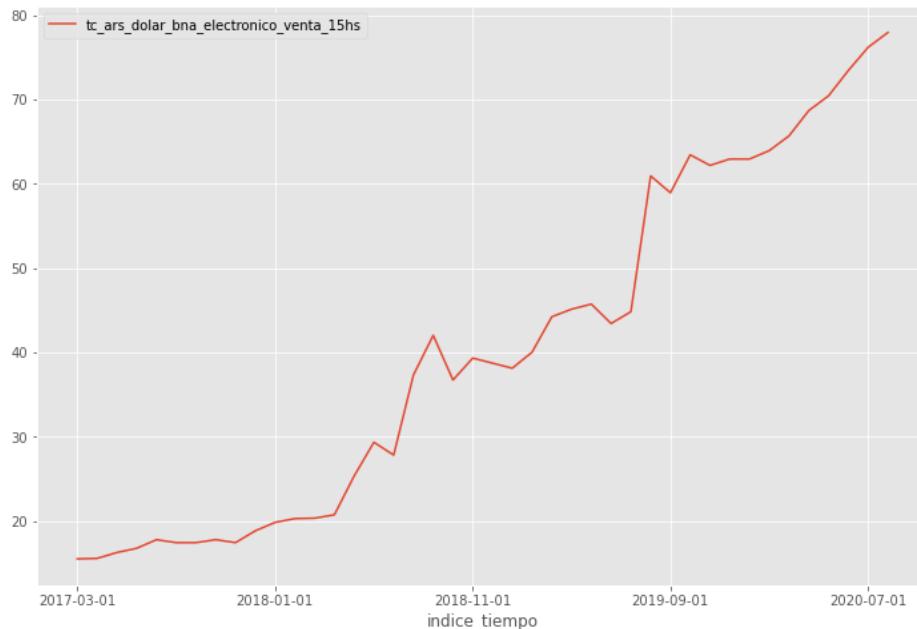
Tambien podemos cambiar la funcion que se usa para agregar datos temporalmente, con `collapse_aggregation`. Las funciones disponibles son:

- `avg`: Realiza el promedio de todos los valores agrupados. Es la opción por defecto si no se indica valor para `collapse_aggregation`.
- `sum`: Suma todos los valores agrupados.
- `end_of_period`: Último valor del período.
- `min`: Mínimo entre los valores agrupados.
- `max`: Máximo entre los valores agrupados.

```
tc_bna_api = f"{BASE_API}&ids={tc_bna_id}&collapse=month&collapse_aggregation=end_of_period"
```

```
pd.read_csv(tc_bna_api).set_index('indice_tiempo').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af9d48e0>
```



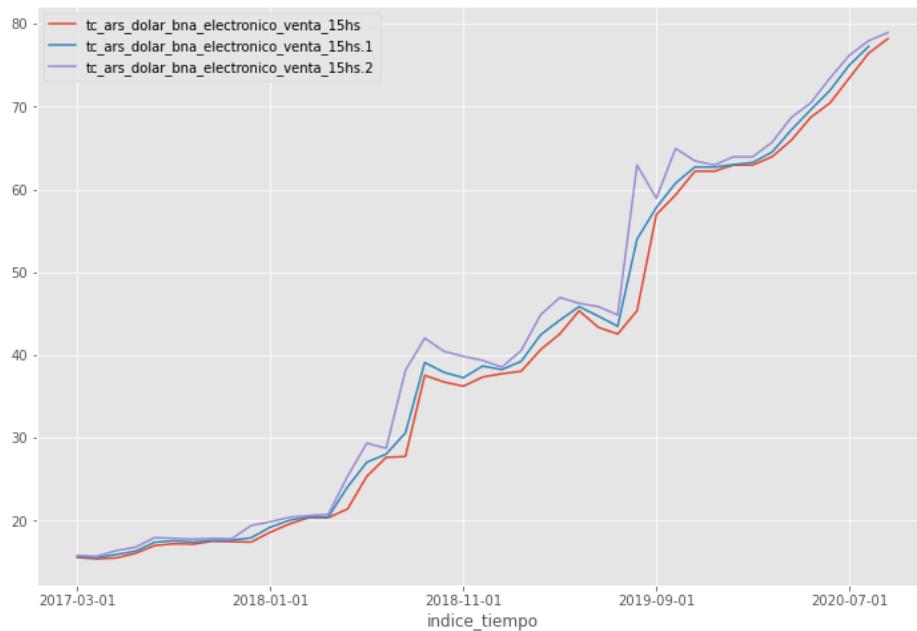
Ese grafico muestra los valores de cierre de cada mes, para el tipo de cambio del Banco Nacion. Pero como hacemos para tener en un mismo grafico los minimos, maximos y promedios mensuales del tipo de cambio?

El parametro `collapse_aggregation` se puede usar en forma individual para cada serie, y las series se pueden repetir en la misma llamada.

```
tc_bna_api = f"{BASE_API}&ids={tc_bna_id}:min,{tc_bna_id}:avg,{tc_bna_id}:max&collapse=month"
```

```
pd.read_csv(tc_bna_api).set_index('indice_tiempo').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af68f5e0>
```

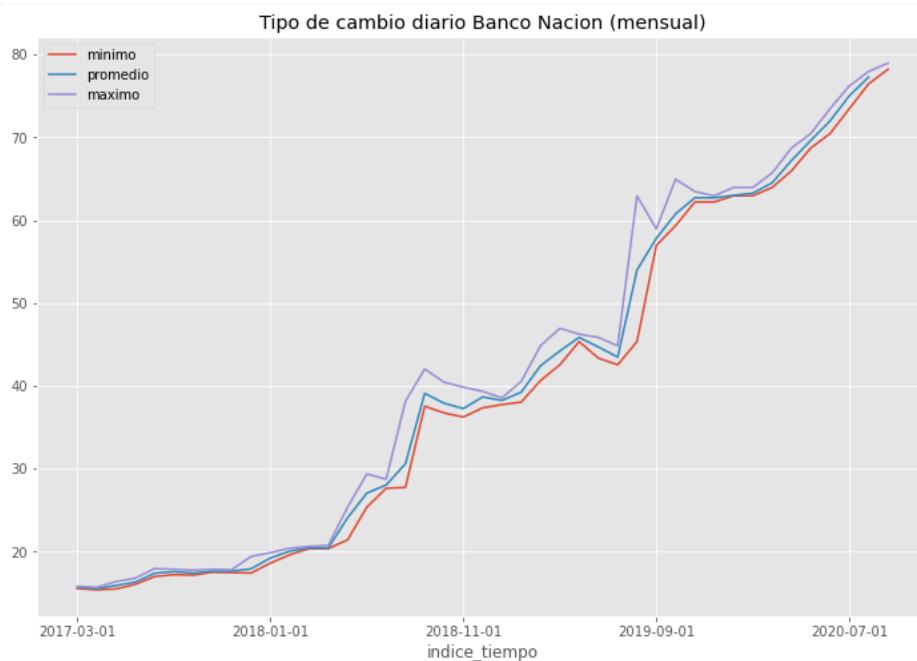


En este caso probablemente convenga cambiar los nombres de las columnas, hacia algo un poco mas claro.

```
df_tc_bna = pd.read_csv(  
    tc_bna_api,  
    # elige nombres nuevos para las columnas  
    names=['indice_tiempo', 'minimo', 'promedio', 'maximo'],  
    # saltea la primera fila, que contiene los otros nombres  
    skiprows=1,  
).set_index('indice_tiempo')
```

```
df_tc_bna.plot(  
    title="Tipo de cambio diario Banco Nacion (mensual)"  
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af600e50>
```



**Ejercicio:** realiza un grafico similar al del tipo de cambio, con las temperaturas diarias minimas, maximas y promedio por mes. Podes usar alguna de las [series del Servicio Meteorologico Nacional](#) para esto. La estacion de Aeroparque sirve como referencia de las temperaturas en la Ciudad de Buenos Aires.

### 36.2.1.2. Modo de representacion (`representation_mode`)

El parametro `representation_mode` permite aplicar transformaciones a las series. Funciona en forma muy similar al `collapse_aggregation` y tambien se puede aplicar a series individuales usando dos puntos.

Los modos de representación disponibles son:

- `value`: Es el modo de representación por defecto. Devuelve el valor medido en la serie.
- `change`: Devuelve la diferencia absoluta entre el valor del período t y el de t-1.
- `change_a_year_ago`: Devuelve la diferencia absoluta entre el valor del período t y el valor del periodo t, del año anterior.
- `change_since_beginning_of_year`: Devuelve la diferencia absoluta entre el valor del período t y el valor del 1 de enero del año al que pertenece t.
- `percent_change`: Devuelve la variación porcentual entre el valor del período t y el de t-1.
- `percent_change_a_year_ago`: Devuelve la variación porcentual entre el valor del período t y el del período t equivalente de hace un año atrás.
- `percent_change_since_beginning_of_year`: Devuelve la variación porcentual entre el valor del período t y el valor del 1 de enero del año al que pertenece t.

Esto nos podria servir, por ejemplo, para comparar el ritmo de aumento de la carne picada comun con el del nivel general de precios de la economia.

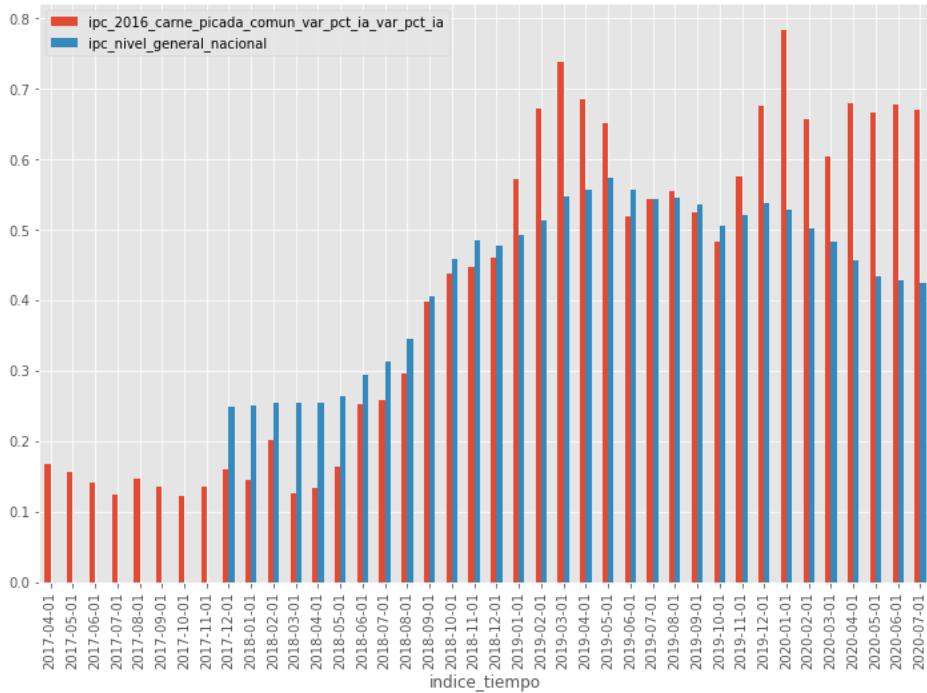
Sin embargo aca conviene no usar el parametro `last`, para tener los calculos hasta los valores mas actualizados (este parametro no funciona bien en combinacion con `representation_mode`).

```
BASE_API = 'https://apis.datos.gob.ar/series/api/series/?format=csv&limit=5000'
```

```
ipc_id = '148.3_INIVELNAL_DICI_M_26'
inflacion_picada_api = f'{BASE_API}&ids={picada_id}, {ipc_id}&representation_mode=percent_change_a_year_ago'
```

```
pd.read_csv(inflacion_picada_api).set_index('indice_tiempo').plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af5e66d0>
```



**Ejercicio:** hacer un grafico donde se compare el ritmo de crecimiento interanual de los precios de 3 productos diferentes. Podes buscar los ids en la distribucion de [precios de referencia del IPC GBA](#).

**Ejercicio:** hacer una tabla (un dataframe) con las variaciones porcentuales mensuales e interanuales del IPC general de las distintas regiones del pais. Podes buscar los ids [en esta distribucion](#).

### 36.2.1.3. Filtros por fechas (`start_date` y `end_date`)

Los parametros `start_date` y `end_date` permiten filtrar por rango de tiempo. Siempre deben usarse respetando el formato ISO 8601 para las fechas, utilizando solo la parte de la fecha y no la de la hora:

YYYY-MM-DD

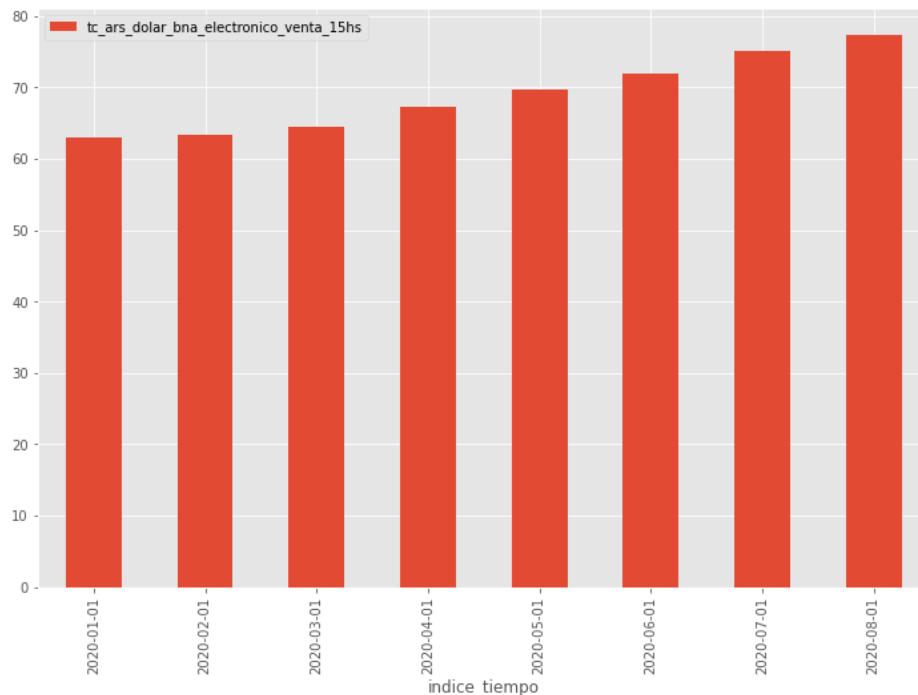
Se puede usar completa, o solo la parte relevante. Ejemplo:

- “2020” filtrara todos los valores que sean anteriores al 2020-01-01
- “2020-03” filtrara todos los valores que sean anteriores al 2020-03-01
- “2020-03-28” filtrara todos los valores que sean anteriores al 2020-03-28

```
tc_bna_api = f"{BASE_API}&ids={tc_bna_id}&collapse=month&start_date=2020"
```

```
pd.read_csv(tc_bna_api).set_index('indice_tiempo').plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3afa008b0>
```

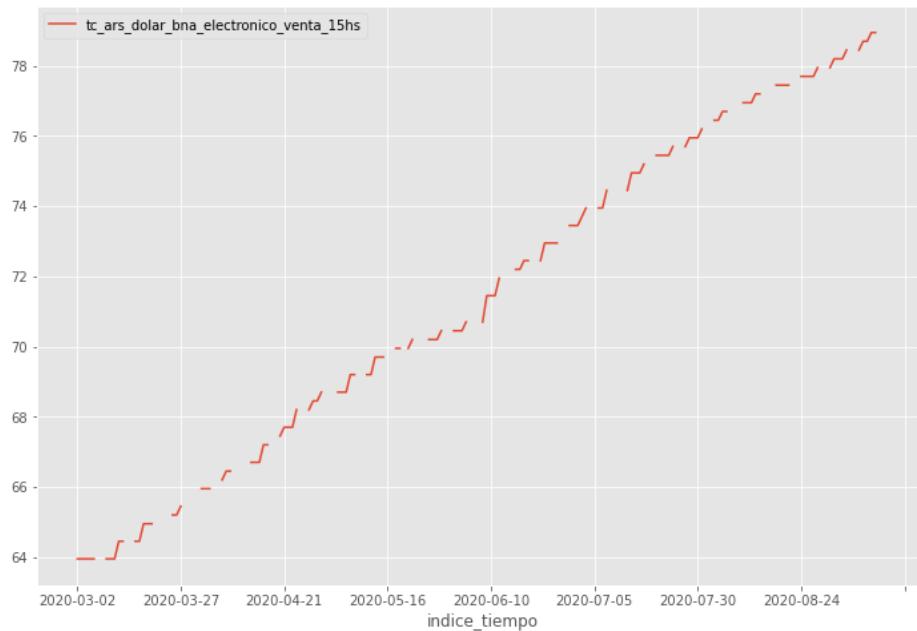


Serie diaria desde marzo de 2020.

```
tc_bna_api = f"{BASE_API}&ids={tc_bna_id}&start_date=2020-03"
```

```
pd.read_csv(tc_bna_api).set_index('indice_tiempo').plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3af3884f0>
```



**Ejercicio:** hacer un grafico del valor del kilo de pan frances desde el comienzo de la cuarentena en Argentina, junto con el indice general de inflacion en un eje secundario.

### 36.2.2. Ejemplo de uso 1: analisis de correlacion con el EMAE

Ahora vamos a usar algunas series de la API para usar algunas funcionalidades de `pandas` muy frecuentemente utilizadas cuando se trabaja con series de tiempo. Para esto vamos usar algunas series del Estimador Mensual de Actividad Económica (EMAE).

```
# tomamos los datos del Estimador Mensual de la Actividad Económica
emae = pd.read_csv(
    f'{BASE_API}&ids=143.3_NO_PR_2004_A_31,143.3_NO_PR_2004_A_28,143.3_NO_PR_2004_A_21'
).set_index('indice_tiempo')
```

```
# para ver las últimas filas
emae.tail(10)
```

indice_tiempo	emae_desestacionalizada	emae_tendencia_ciclo	emae_original
2019-09-01	142.277423	142.092375	135.057471
2019-10-01	143.525550	141.848306	142.240648
2019-11-01	141.767174	141.563450	138.158557
2019-12-01	141.626587	141.255951	135.445432
2020-01-01	141.212241	140.944043	131.987520
2020-02-01	140.632007	140.643578	129.369076
2020-03-01	126.237896	140.368520	127.869512
2020-04-01	103.963419	140.128699	110.931869
2020-05-01	114.030034	139.929137	129.448148
2020-06-01	122.420495	139.769888	132.717002

Si el objetivo solo fuera tener una rutina para generar la tabla actualizada de las series del EMAE que pueda correr, por ejemplo, todos los días lunes, desde un dataframe se puede guardar en una variedad de formatos.

```

emae.to_csv("emae.csv", encoding="utf8")
emae.to_excel("emae.xlsx", encoding="utf8")
emae.to_html("emae.html")
emae.to_stata("emae.dta")

```

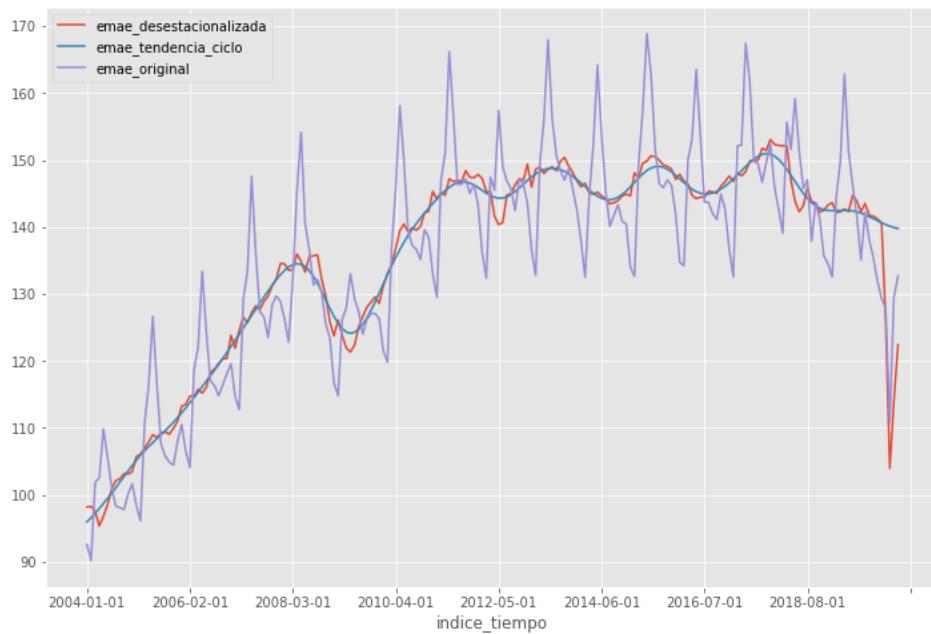
**Ejercicio:** Generar una tabla en Excel o CSV de los tipos de cambio vendedores, canal electronico, valor de cierre a las 15hs, de 3 entidades financieras diferentes. Recuerden que pueden buscar esas series en <https://datos.gob.ar/series/api/search/?q=ARS/USD+electronico+venta+15hs&offset=0&limit=10>

Si el objetivo es generar un grafico actualizado para poner en un informe

```

emae.plot()
plt.savefig('emae.png')

```



**Ejercicio:** hace un grafico de la evolucion del EMAE desde comienzos de 2018 y guardalo en PNG.

Pista: no hace falta que armes una nueva llamada a la API para filtrar por tiempo, podes hacerlo en `pandas: emae[emae.index > '2018']`

Como es un grafico mensual, tal vez quieras eliminar el dia del indice. Podes conseguir esto convirtiendo los valores del indice a “periodos” en `pandas`.

Para esto primero tenemos que convertir el tipo del indice de “texto” a “fecha”.

```

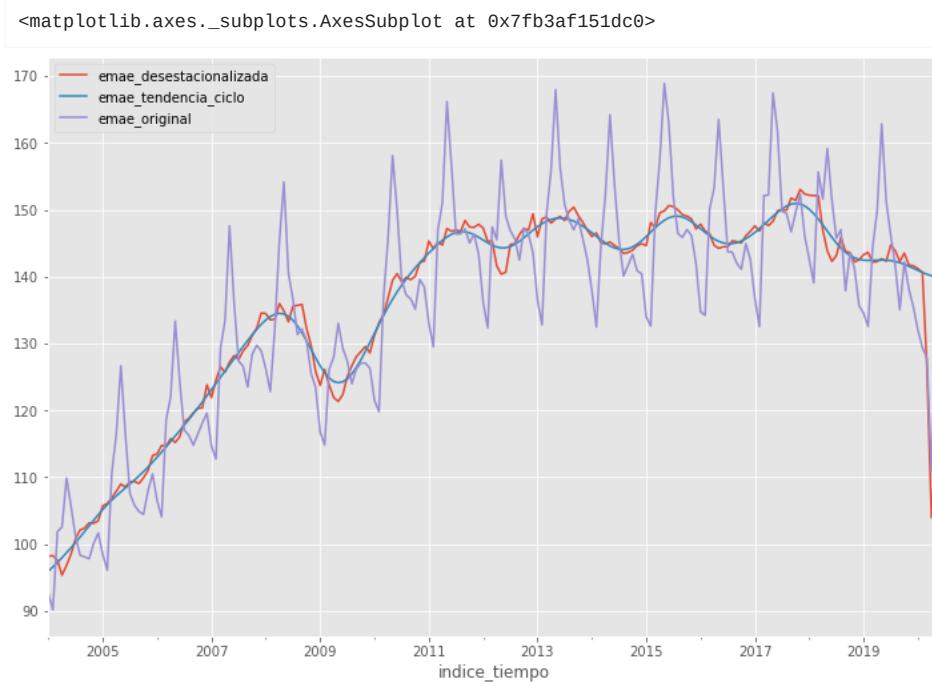
emae.index = pd.to_datetime(emae.index)
emae.index = emae.index.to_period()

```

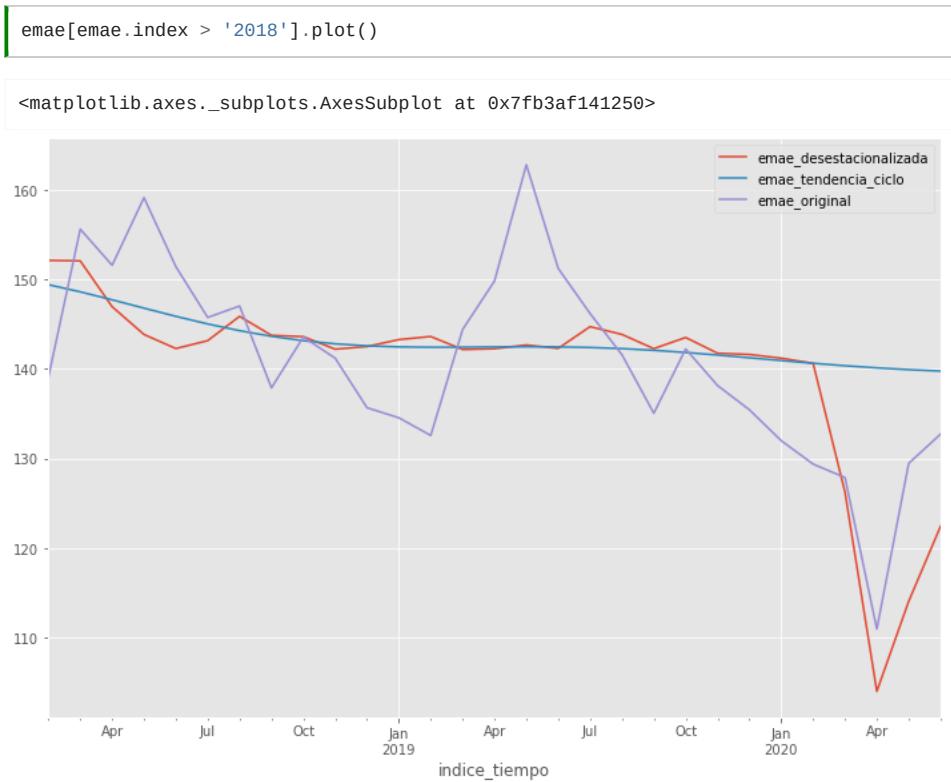
```

emae.plot()

```



Pandas automaticamente tratará el indice ofreciendo el detalle mas conveniente para el nivel de zoom aplicado, por eso se ve solo el año en el grafico anterior.



Ahora vamos a tratar de analizar la correlacion de distintos sectores de la economía con el nivel de actividad general.

```
# descargamos el emae general, construcción, industria, comercio, pesca,
# agricultura y hotelería
nivel_actividad = pd.read_csv(
    f'{BASE_API}&ids=11.3_VMATC_2004_M_12,143.3_NO_PR_2004_A_21,11.3_VMASD_2004_M_23,11.
    3_VIPAA_2004_M_5,11.3_ISOM_2004_M_39,11.3_P_2004_M_20,11.3_AGCS_2004_M_41'
).set_index('indice_tiempo')

# generamos el índice de tiempo
nivel_actividad.index = pd.to_datetime(nivel_actividad.index)
```

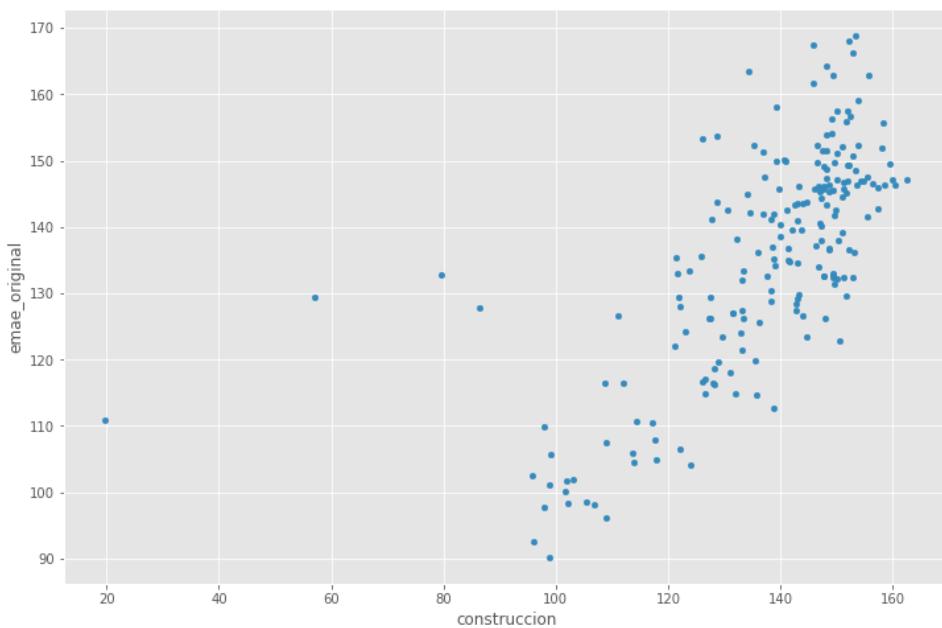
```
nivel_actividad.tail()
```

indice_tiempo	construcion	emae_original	industria_manufacturera	pesca	agricultura
2020-02-01	127.601879	129.369076	102.487593	162.063730	
2020-03-01	86.299511	127.869512	96.988588	77.939813	
2020-04-01	19.629258	110.931869	76.909105	124.102878	
2020-05-01	57.068012	129.448148	92.280154	148.763043	
2020-06-01	79.549698	132.717002	113.095569	83.910534	

El metodo `plot` de un dataframe permite hacer distintos graficos, tanto usando el parametro `kind=bar` como accediendo al tipo de grafico a traves de un punto.

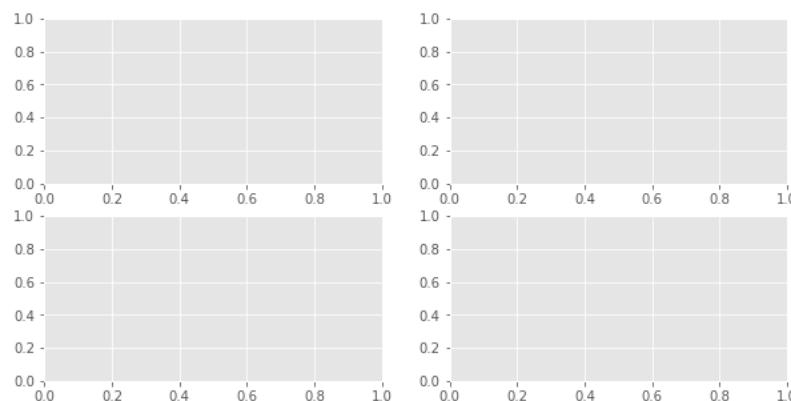
```
nivel_actividad.plot.scatter("construcion", "emae_original")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3ad8c9100>
```



Con `matplotlib` podemos hacer multiples graficos en uno solo, con la funcion `plt.subplots()`.

```
# genera un lienzo con 4 subgráficos vacíos
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 5))
```



Dentro de la misma celda donde se crea el lienzo de subgráficos, deben crearse y asignarse a los "axes" (ejes) c/u de los subgráficos.

```

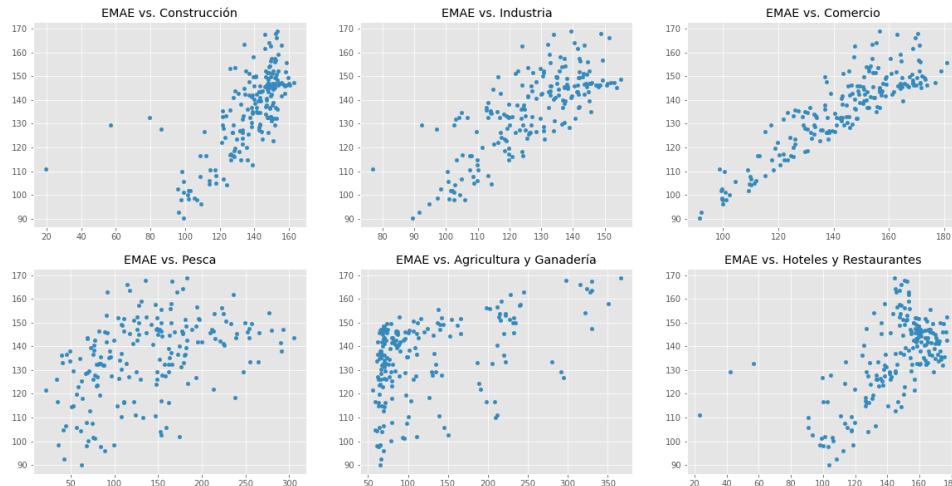
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20, 10))

# graficos de la fila 1
nivel_actividad.plot.scatter("construccion", "emae_original",
                             ax=axes[0,0], title="EMAE vs. Construcción")
nivel_actividad.plot.scatter("industria_manufacturera", "emae_original",
                             ax=axes[0,1], title="EMAE vs. Industria")
nivel_actividad.plot.scatter("comercio_mayorista_minorista_reparaciones",
                             "emae_original",
                             ax=axes[0,2], title="EMAE vs. Comercio")

# graficos de la fila 2
nivel_actividad.plot.scatter("pesca", "emae_original", ax=axes[1,0],
                             title="EMAE vs. Pesca")
nivel_actividad.plot.scatter("agricultura_ganaderia_caza_silvicultura",
                             "emae_original",
                             ax=axes[1,1], title="EMAE vs. Agricultura y
Ganadería")
nivel_actividad.plot.scatter("hoteles_restaurantes", "emae_original",
                             ax=axes[1,2], title="EMAE vs. Hoteles y
Restaurantes")

# elimina los labels de los ejes para que se vea mejor
for row in axes:
    for ax in row:
        pass
        ax.set_xlabel("")
        ax.set_ylabel("")

```



La construcción, la industria, el comercio y los servicios de hotelería y restaurantes parecen correlacionar fuertemente con el nivel de actividad general, mientras que la pesca y el sector agropecuario casi no muestran relación alguna! Podés mirar la matriz de correlaciones de la tabla para corroborar esto.

```
nivel_actividad.corr()
```

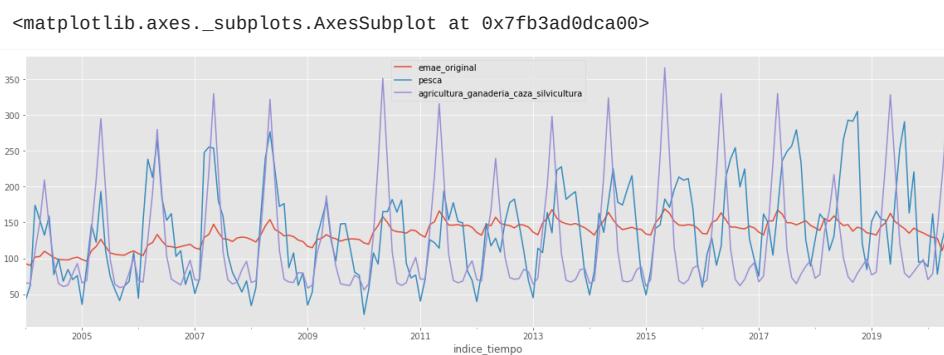
	construccion	emae_original	industria_man
<b>construccion</b>	1.000000	0.694427	
<b>emae_original</b>	0.694427	1.000000	
<b>industria_manufacturera</b>	0.770678	0.768573	
<b>pesca</b>	0.268925	0.439108	
<b>agricultura_ganaderia_caza_silvicultura</b>	-0.062003	0.456038	
<b>hoteles_restaurantes</b>	0.838482	0.623531	
<b>comercio_mayorista_minorista_reparaciones</b>	0.779643	0.893406	

Si ves la pesca y el sector agropecuario en el mismo gráfico con la serie original, es evidente que estos dos sectores tienen una estacionalidad mucho más pronunciada que el nivel general de actividad, esto puede explicar en parte que tengan una baja correlación.

```

nivel_actividad[
    ["emae_original", "pesca", "agricultura_ganaderia_caza_silvicultura"]
].plot(figsize=(20, 6))

```



Probablemente debas chequear si existe relación cuando se consideran las variaciones del índice, en lugar del valor absoluto.

Pero el bloque de código para generar los graficos multiples es bastante engorroso para estar probando ágilmente esas transformaciones y cambiar rápidamente lo que vemos.

**Deberíamos poner ese bloque dentro de una función, cuyos argumentos de entrada sean aquellas cosas que quieras cambiar.**

```
def generate_emae_scatters(nivel_actividad):
    """Grafica el scatter de 6 sectores vs. el nivel general de actividad."""

    fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(20, 10))

    # graficos de la fila 1
    nivel_actividad.plot.scatter("construccion", "emae_original",
                                  ax=axes[0,0], title="EMAE vs. Construcción")
    nivel_actividad.plot.scatter("industria_manufacturera", "emae_original",
                                 ax=axes[0,1], title="EMAE vs. Industria")
    nivel_actividad.plot.scatter("comercio_mayorista_minorista_reparaciones",
                                 "emae_original",
                                 ax=axes[0,2], title="EMAE vs. Comercio")

    # graficos de la fila 2
    nivel_actividad.plot.scatter("pesca", "emae_original", ax=axes[1,0],
                                 title="EMAE vs. Pesca")
    nivel_actividad.plot.scatter("agricultura_ganaderia_caza_silvicultura",
                                 "emae_original",
                                 ax=axes[1,1], title="EMAE vs. Agricultura y
Ganadería")
    nivel_actividad.plot.scatter("hoteles_restaurantes", "emae_original",
                                 ax=axes[1,2], title="EMAE vs. Hoteles y
Restaurantes")

    # elimina los labels de los ejes para que se vea mejor
    for row in axes:
        for ax in row:
            pass
            ax.set_xlabel("")
            ax.set_ylabel("")

    return axes
```

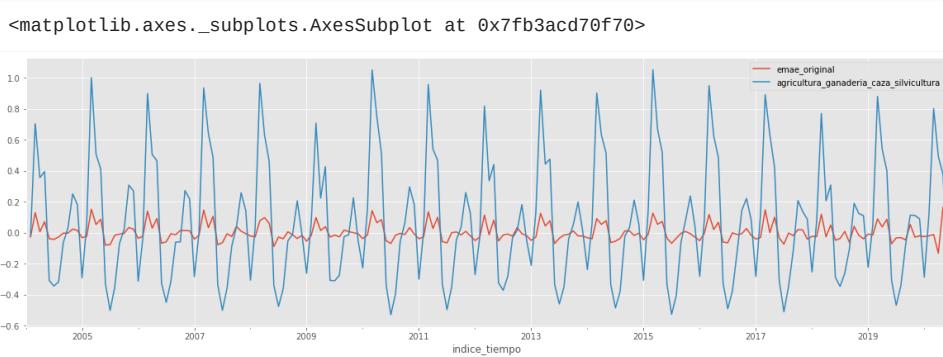
**Ejercicio:** usa la funcion para graficar las correlaciones entre los % de variacion contra el periodo anterior.

Pista: en pandas podes agregar un `.pct_change(1)` para convertir todos los valores a % de variacion respecto del valor anterior. Que crees que pasara si repetis los graficos con `.pct_change(2)`?

Luego de hacer los graficos, calcula tambien la nueva matriz de correlaciones.

Las variaciones en el sector agropecuario casi siempre empujan a las variaciones en el nivel general (que obviamente ocurren en menor magnitud) ya que es un sector con una alta incidencia en la produccion total en Argentina. No es este el caso de la pesca.

```
nivel_actividad.pct_change(1)[
    ["emae_original", "agricultura_ganaderia_caza_silvicultura"]]
].plot(figsize=(20, 6))
```



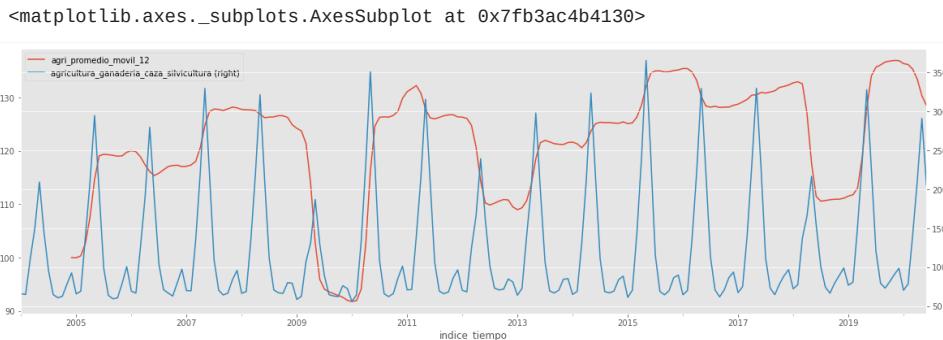
Si querés seguir la evolución de mediano plazo de la producción agropecuaria, una forma sencilla de remover el ruido estacional es aplicar un promedio móvil de 12 meses.

Para esto podés usar la función `rolling` que te permite elegir una ventana de X períodos sobre la cual hacer un cálculo o una agregación.

```
nivel_actividad["agri_promedio_móvil_12"] = 
nivel_actividad.agricultura_ganaderia_caza_silvicultura.rolling(12).mean()
```

Ahora grafiquemos ambas series, con y sin promedio móvil.

```
nivel_actividad[
    ["agri_promedio_móvil_12", "agricultura_ganaderia_caza_silvicultura"]
].plot(figsize=(20,6), secondary_y="agricultura_ganaderia_caza_silvicultura")
```



**Ejercicio:** chequear la correlación entre las variaciones interanuales (12 meses) del EMAE y sus componentes sectoriales.

**Ejercicio para hacer en casa:** crear un simple tablero de seguimiento de variables de coyuntura usando subplots que contenga la evolución del tipo de cambio BNA vendedor (promedio móvil 30 días), la tasa de interés de política monetaria, la inflación interanual mensual (`pct_change(12)`), la expectativa de inflación de los próximos 12 meses, las variaciones porcentuales mensuales del nivel de actividad (EMAE) y el saldo comercial mensual (promedio móvil 3 meses).

### 36.2.3. Ejemplo de uso 2: generacion de un informe

<https://github.com/datosgobar/taller-series-tiempo-mediaparty-2018/blob/master/2A-python.ipynb>

Otro uso frecuente es armar reportes que se puedan posteriormente programar y (por ejemplo) enviar por mail automaticamente.

Vamos a ver como hacer esto de una forma sencilla, usando los templates de texto de python , un modulo auxiliar para enviar mails, que esta incluido en este repositorio, y algunas librerías para generar PDFs.

En este caso, posiblemente nos convenga pedir los datos en formato JSON. Tambien queremos hacer un reporte sencillo que solo tenga el ultimo valor conocido de cada variable, y su variacion interanual.

Asi que vamos a cambiar la llamada base a la API, para pedir solo los valores que necesitamos.

```
BASE_API = 'https://apis.datos.gob.ar/series/api/series/?
format=json&limit=13&sort=desc&collapse=month'
```

Habráas notado que usamos el parametro `sort` para ordenar los resultados en orden descendente (el mas reciente primero). Esto nos va a facilitar seleccionar "el ultimo resultado".

```
api_call = f"{BASE_API}&ids=168.1_T_CAMBIOR_D_0_0_26:avg,168.1_T_CAMBIOR_D_0_0_26:max,168.1_T_CAMBIOR_D_0_0_26:min"
requests.get(api_call).json()
```

La llamada a la API en formato JSON nos devuelve los datos, pero tambien los metadatos. Esto es principalmente util cuando estamos desarrollando aplicaciones web de algun tipo.

Si estamos analizando datos, posiblemente ya hayamos usado los metadatos para elegir las series, pero no los necesitamos mas.

```
tipo_cambio = requests.get(api_call).json()["data"]
tipo_cambio
[['2020-07-01', 74.9758064516129, 76.25, 73.5],
['2020-06-01', 71.98333333333333, 73.5, 70.5],
['2020-05-01', 69.69354838709677, 70.5, 68.75],
['2020-04-01', 67.13333333333334, 68.75, 66.0],
['2020-03-01', 64.66129032258064, 65.75, 64.0],
['2020-02-01', 63.310344827586206, 64.0, 63.0],
['2020-01-01', 63.0, 63.0, 63.0],
['2019-12-01', 62.78225806451613, 63.0, 62.25],
['2019-11-01', 62.76666666666666, 63.5, 62.25],
['2019-10-01', 60.806451612903224, 65.0, 59.4],
['2019-09-01', 58.00833333333333, 61.0, 57.0],
['2019-08-01', 53.81612903225806, 63.0, 45.4],
['2019-07-01', 43.45161290322582, 44.9, 42.6]]
```

Ahora vamos a hacer lo mismo con otras variables para nuestro informe.

```
api_call = f"{BASE_API}&ids=101.1_I2NG_2016_M_22"
ipc = requests.get(api_call).json()["data"]
ipc
[['2020-07-01', 322.6791],
['2020-06-01', 317.4661],
['2020-05-01', 311.0922],
['2020-04-01', 306.4483],
['2020-03-01', 302.2274],
['2020-02-01', 291.7371],
['2020-01-01', 286.4913],
['2019-12-01', 281.1775],
['2019-11-01', 270.8019],
['2019-10-01', 260.2101],
['2019-09-01', 252.1482],
['2019-08-01', 238.3069],
['2019-07-01', 229.4286]]
```

```
api_call = f"{BASE_API}&ids=143.3_NO_PR_2004_A_21"
emae = requests.get(api_call).json()["data"]
```

```
emae
```

```
[['2020-06-01', 132.71700174512995],
 ['2020-05-01', 129.4481481382743],
 ['2020-04-01', 110.93186850418589],
 ['2020-03-01', 127.8695117502436],
 ['2020-02-01', 129.36907641347452],
 ['2020-01-01', 131.98752045748455],
 ['2019-12-01', 135.44543162767988],
 ['2019-11-01', 138.15855707957144],
 ['2019-10-01', 142.24064825618933],
 ['2019-09-01', 135.0574708022306],
 ['2019-08-01', 141.60332874981128],
 ['2019-07-01', 146.2032323361963],
 ['2019-06-01', 151.25410111244764]]
```

Ahora que tenemos los datos, armemos el reporte!

```
reporte = """
# Seguimiento de coyuntura

## Tipo de cambio

El tipo de cambio promedio del mes de {mes_tc:%Y-%m} fue {mes_tc_promedio:.2f} con
un máximo de {mes_tc_max:.2f}
y un mínimo de {mes_tc_min:.2f}.

## Inflación

La inflación de {mes_ipc:%Y-%m} respecto de {mes_ipc_anterior:%Y-%m} fue de
{mes_ipc_var:.2%}, mientras que la inflación interanual
{mes_ipc:%Y-%m}/{mes_ipc_anio_anterior:%Y-%m} fue de {mes_ipc_var_interanual:.2%}.

## Nivel de actividad

El EMAE de {mes_emae:%Y-%m} muestra {alza_o_caida_mensual} respecto del mes
anterior de {mes_emae_var:.2%}, y un
{alza_o_caida_interanual} interanual de {mes_emae_var_interanual:.2%}.
""".format(
    #tipo de cambio
    mes_tc=arrow.get(tipo_cambio[1][0]).datetime,
    mes_tc_promedio=tipo_cambio[1][1],
    mes_tc_max=tipo_cambio[1][2],
    mes_tc_min=tipo_cambio[1][3],

    # ipc
    mes_ipc=arrow.get(ipc[0][0]).datetime,
    mes_ipc_anterior=arrow.get(ipc[1][0]).datetime,
    mes_ipc_var=ipc[0][1]/ipc[1][1] - 1,
    mes_ipc_anio_anterior=arrow.get(ipc[12][0]).datetime,
    mes_ipc_var_interanual=ipc[0][1]/ipc[12][1] - 1,

    # emae
    mes_emae=arrow.get(emae[0][0]).datetime,
    alza_o_caida_mensual="alza" if emae[0][1]/emae[1][1]-1 >= 0 else "caída",
    mes_emae_var=emae[0][1]/emae[1][1]-1,
    alza_o_caida_interanual="alza" if emae[0][1]/emae[12][1]-1 >= 0 else "caída",
    mes_emae_var_interanual=emae[0][1]/emae[12][1]-1
)
```

```
from IPython.core.display import display, HTML
```

```
# genera un HTML a partir del markdown
html = markdown.markdown(reporte)
display(HTML(html))
```

## Tipo de cambio

El tipo de cambio promedio del mes de 2020-06 fue 71.98 con un máximo de 73.50 y un mínimo de 70.50.

## Inflación

La inflación de 2020-07 respecto de 2020-06 fue de 1.64%, mientras que la inflación interanual 2020-07/2019-07 fue de 40.64%.

## Nivel de actividad

El EMAE de 2020-06 muestra alza respecto del mes anterior de 2.53%, y un caída interanual de -12.26%.

```
with open('reporte.html', 'wb') as f:  
    f.write(html.encode('utf8'))
```

Este reporte luego se puede enviar por mail, guardar en PDF u otras cosas que no cubriremos en esta clase.

**Ejercicio para la casa:** agregar las variables Exportaciones e Importaciones al reporte y guardar el HTML.

### 36.3. API FRED

Documentacion API: <https://fred.stlouisfed.org/docs/api/fred/>

La API de FRED ([Federal Reserve Economic Data](#)) es una API de series de tiempo desarrollada por la sucursal de St. Louis de la Reserva Federal de Estados Unidos.

Tiene mas de 700 mil series de Estados Unidos e internacionales, y es ampliamente utilizada por economistas y otros profesionales que analizan fenomenos economicos. La API de Series de Tiempo de Argentina fue inspirada mayormente en la experiencia previa de FRED.

Su uso requiere registrarse y adquirir un API key, pero es gratuita.

```
with open('apis_series_creds.json', 'r') as f:  
    creds = json.load(f)  
  
fred_key = creds['fred_key']  
  
BASE_FRED_API = 'https://api.stlouisfed.org/fred/'  
FRED_SERIES_METADATA = BASE_FRED_API + 'series?series_id={serie_id}&api_key='  
&api_key&file_type=json'  
FRED_SERIES_DATA = BASE_FRED_API + 'series/observations?series_id='  
&serie_id&api_key&file_type=json'  
  
gdp_meta_call = FRED_SERIES_METADATA.format(  
    serie_id='GNPCA', api_key=fred_key  
)  
  
gdp_metadata = requests.get(gdp_meta_call).json()  
gdp_metadata
```

```

{'realtime_start': '2020-09-15',
'realtime_end': '2020-09-15',
'serieses': [{'id': 'GNPCA',
'realtime_start': '2020-09-15',
'realtime_end': '2020-09-15',
'title': 'Real Gross National Product',
'observation_start': '1929-01-01',
'observation_end': '2019-01-01',
'frequency': 'Annual',
'frequency_short': 'A',
'units': 'Billions of Chained 2012 Dollars',
'units_short': 'Bil. of Chn. 2012 $',
'seasonal_adjustment': 'Not Seasonally Adjusted',
'seasonal_adjustment_short': 'NSA',
'last_updated': '2020-07-30 07:57:33-05',
'popularity': 18,
'notes': 'BEA Account Code: A001RX\\n\\n'}]}

```

```

gdp_data_call = FRED_SERIES_DATA.format(
    serie_id='GNPCA', api_key=fred_key
)

```

```

gdp_data = requests.get(gdp_data_call).json()
gdp_data

```

```

df_gdp = pd.DataFrame(gdp_data['observations'])
df_gdp

```

	realtime_start	realtime_end	date	value
0	2020-09-15	2020-09-15	1929-01-01	1120.076
1	2020-09-15	2020-09-15	1930-01-01	1025.091
2	2020-09-15	2020-09-15	1931-01-01	958.378
3	2020-09-15	2020-09-15	1932-01-01	834.291
4	2020-09-15	2020-09-15	1933-01-01	823.156
...	...	...	...	...
86	2020-09-15	2020-09-15	2015-01-01	17647.607
87	2020-09-15	2020-09-15	2016-01-01	17955.437
88	2020-09-15	2020-09-15	2017-01-01	18421.034
89	2020-09-15	2020-09-15	2018-01-01	18951.897
90	2020-09-15	2020-09-15	2019-01-01	19338.371

91 rows × 4 columns

**Ejercicio para la casa:** busca el id de la serie de inflacion en la pagina de FRED, y genera un dataframe de series de tiempo que tenga la siguiente estructura:

- Las fechas (date) deben usarse para construir un indice de tiempo (el index de la tabla).
- Las unicas columnas que debe haber en el dataframe son aquellas que corresponden a cada serie: `pbi_usa`, `ipc_usa`.

## 36.4. API Quandl

<https://www.quandl.com/tools/api>

Quandl ofrece una API de series de tiempo gratuita que contiene numerosos datasets publicos y privados (estos ultimos pagos). Entre ellos, la base completa de FRED es accesible desde la API de Quandl.

La interfaz de la API de Series de Tiempo de Argentina tomo varias ideas de la de Quandl.

```

BASE_QUANDL_API = 'https://www.quandl.com/api/v3/datasets/'

```

```

fred_gdp_call = BASE_QUANDL_API + 'FRED/GDP.csv'
wiki_aapl_call = BASE_QUANDL_API + 'WIKI/AAPL.csv'

```

```
df_fred_gdp = pd.read_csv(fred_gdp_call)
df_fred_gdp
```

	Date	Value
0	2020-04-01	19486.509
1	2020-01-01	21561.139
2	2019-10-01	21747.394
3	2019-07-01	21540.325
4	2019-04-01	21329.877
...	...	...
289	1948-01-01	265.742
290	1947-10-01	259.745
291	1947-07-01	249.585
292	1947-04-01	245.968
293	1947-01-01	243.164

294 rows × 2 columns

```
df_wiki_aapl = pd.read_csv(wiki_aapl_call)
df_wiki_aapl
```

	Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open
0	2018-03-27	173.68	175.15	166.92	168.340	38962839.0	0.0	1.0	173.680000
1	2018-03-26	168.07	173.10	166.44	172.770	36272617.0	0.0	1.0	168.070000
2	2018-03-23	168.39	169.92	164.94	164.940	40248954.0	0.0	1.0	168.390000
3	2018-03-22	170.00	172.68	168.60	168.845	41051076.0	0.0	1.0	170.000000
4	2018-03-21	175.04	175.09	171.26	171.270	35247358.0	0.0	1.0	175.040000
...	...	...	...	...	...	...	...	...	...
9395	1980-12-18	26.63	26.75	26.63	26.630	327900.0	0.0	1.0	0.391536
9396	1980-12-17	25.87	26.00	25.87	25.870	385900.0	0.0	1.0	0.380362
9397	1980-12-16	25.37	25.37	25.25	25.250	472000.0	0.0	1.0	0.373010
9398	1980-12-15	27.38	27.38	27.25	27.250	785200.0	0.0	1.0	0.402563
9399	1980-12-12	28.75	28.87	28.75	28.750	2093900.0	0.0	1.0	0.422706

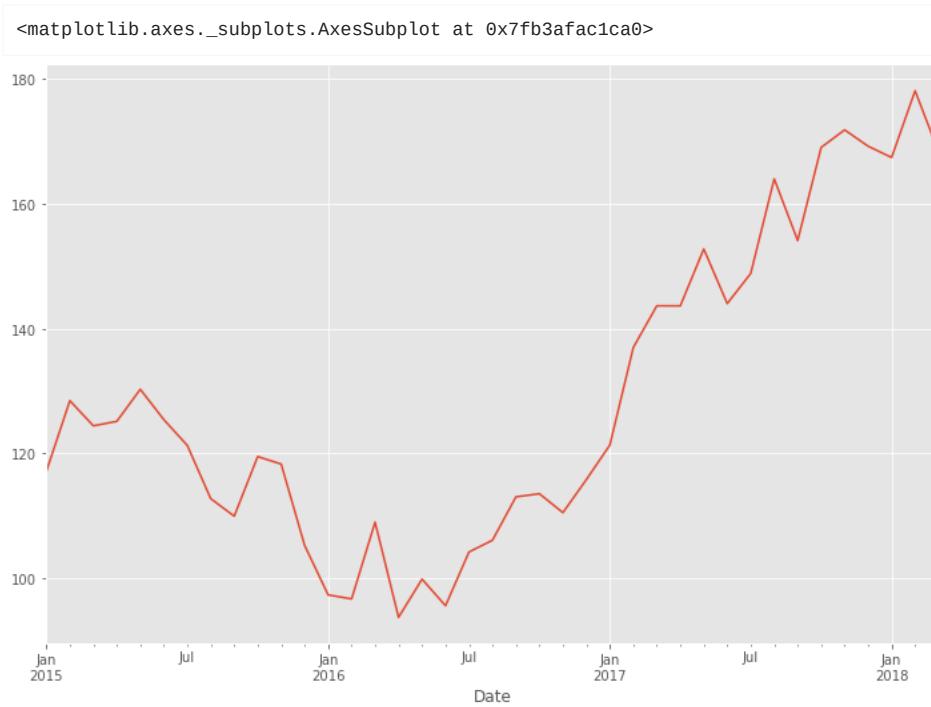
9400 rows × 13 columns

Algunas de las funcionalidades de filtro y transformacion te van a resultar conocidas!

```
wiki_aapl_call_transf = BASE_QUANDL_API + 'WIKI/AAPL.csv?
column_index=4&start_date=2015-01-01&collapse=monthly'
df_wiki_aapl_transf = pd.read_csv(wiki_aapl_call_transf)
df_wiki_aapl_transf
```

	Date	Close
<b>0</b>	2018-03-31	168.340
<b>1</b>	2018-02-28	178.120
<b>2</b>	2018-01-31	167.430
<b>3</b>	2017-12-31	169.230
<b>4</b>	2017-11-30	171.850
<b>5</b>	2017-10-31	169.040
<b>6</b>	2017-09-30	154.120
<b>7</b>	2017-08-31	164.000
<b>8</b>	2017-07-31	148.850
<b>9</b>	2017-06-30	144.020
<b>10</b>	2017-05-31	152.760
<b>11</b>	2017-04-30	143.650
<b>12</b>	2017-03-31	143.660
<b>13</b>	2017-02-28	136.990
<b>14</b>	2017-01-31	121.350
<b>15</b>	2016-12-31	115.820
<b>16</b>	2016-11-30	110.520
<b>17</b>	2016-10-31	113.540
<b>18</b>	2016-09-30	113.050
<b>19</b>	2016-08-31	106.100
<b>20</b>	2016-07-31	104.210
<b>21</b>	2016-06-30	95.600
<b>22</b>	2016-05-31	99.860
<b>23</b>	2016-04-30	93.740
<b>24</b>	2016-03-31	108.990
<b>25</b>	2016-02-29	96.690
<b>26</b>	2016-01-31	97.340
<b>27</b>	2015-12-31	105.260
<b>28</b>	2015-11-30	118.300
<b>29</b>	2015-10-31	119.500
<b>30</b>	2015-09-30	109.950
<b>31</b>	2015-08-31	112.760
<b>32</b>	2015-07-31	121.300
<b>33</b>	2015-06-30	125.425
<b>34</b>	2015-05-31	130.280
<b>35</b>	2015-04-30	125.150
<b>36</b>	2015-03-31	124.430
<b>37</b>	2015-02-28	128.460
<b>38</b>	2015-01-31	117.160

```
df_wiki_aapl_transf.index = pd.to_datetime(df_wiki_aapl_transf.Date)
df_wiki_aapl_transf.Close.plot()
```



**Ejercicio para la casa:** generar un dataframe de series de tiempo que tenga las acciones de Apple, Microsoft, Google y Amazon.

**Ejercicio para la casa:** generar un dataframe de series de tiempo con las proyecciones de crecimiento del producto bruto elaboradas por el FMI, para una muestra de paises de Sudamerica. (Dataset del FMI: <https://www.quandl.com/data/ODA-IMF-Cross-Country-Macroeconomic-Statistics>)

### 36.5. API Banco Mundial

Las APIs del Banco Mundial exponen cientos de indicadores de desarrollo de todos los paises del mundo de frecuencia anual, trimestral y mensual.

- Documentacion APIs: <https://datahelpdesk.worldbank.org/knowledgebase/topics/125589>
- Documentacion API indicadores: <https://datahelpdesk.worldbank.org/knowledgebase/articles/898581-api-basic-call-structures>
- Buscar indicadores: <https://datos.bancomundial.org/indicator>

Ahora vamos a descargar las series por pais de un indicador.

```
BASE_BM_API =
'http://api.worldbank.org/v2/es/country/{pais}/indicator/{indicador}?
format=json&per_page=20000&date=2000:2020'
```

Notar que la API del BM soporta varios idiomas: en este caso estamos pidiendo los datos en español `v2/es/country`.

Usamos un valor de `per_page` alto para obtener una gran cantidad de resultados sin tener que iterar entre paginas.

Pedimos los resultados para un rango de fechas determinado con `date`, para limitar la cantidad de resultados que se retornan.

Como siempre, fijamos en un template BASE los parametros que siempre vamos a usar para llamar a la API, y dejamos como variables aquellos que vamos a ir cambiando.

```
# esta llamada pide las exportaciones a USD constantes de 2010, para todos los
# paises
exportaciones_wb = requests.get(
    BASE_BM_API.format(
        pais='all',
        indicador='NE.EXP.GNFS.KD'
    )
).json()
```

```
exportaciones_wb
```

Hay que conseguir leer la estructura de respuesta JSON de forma tal que pueda construirse un DataFrame.

```
df_expo = pd.DataFrame(exportaciones_wb[1])
```

```
df_expo
```

	indicator	country	countryiso3code	date	value	unit	obs_
0	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': '1A', 'value': 'El mundo árabe'}			ARB 2019	1.387774e+12		
1	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': '1A', 'value': 'El mundo árabe'}			ARB 2018	1.418474e+12		
2	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': '1A', 'value': 'El mundo árabe'}			ARB 2017	1.303973e+12		
3	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': '1A', 'value': 'El mundo árabe'}			ARB 2016	1.273703e+12		
4	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': '1A', 'value': 'El mundo árabe'}			ARB 2015	1.242632e+12		
...	...	...	...	...	...	...	...
5275	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': 'ZW', 'value': 'Zimbabwe'}			ZWE 2004	4.238208e+09		
5276	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': 'ZW', 'value': 'Zimbabwe'}			ZWE 2003	4.327190e+09		
5277	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': 'ZW', 'value': 'Zimbabwe'}			ZWE 2002	5.438714e+09		
5278	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': 'ZW', 'value': 'Zimbabwe'}			ZWE 2001	6.549891e+09		
5279	{'id': 'NE.EXP.GNFS.KD', 'value': 'Exportacion...'}{'id': 'ZW', 'value': 'Zimbabwe'}			ZWE 2000	6.342110e+09		

5280 rows × 8 columns

La estructura anidada es un problema en algunos casos... Afortunadamente pandas tiene un metodo para normalizar (aplanar) la estructura anidada de un JSON.

```
df_expo = pd.json_normalize(exportaciones_wb[1])
```

```
df_expo
```

	countryiso3code	date	value	unit	obs_status	decimal	indicator.i
0	ARB	2019	1.387774e+12			0	NE.EXP.GNFS.K
1	ARB	2018	1.418474e+12			0	NE.EXP.GNFS.K
2	ARB	2017	1.303973e+12			0	NE.EXP.GNFS.K
3	ARB	2016	1.273703e+12			0	NE.EXP.GNFS.K
4	ARB	2015	1.242632e+12			0	NE.EXP.GNFS.K
...	...	...	...	...	...	...	...
5275	ZWE	2004	4.238208e+09			0	NE.EXP.GNFS.K
5276	ZWE	2003	4.327190e+09			0	NE.EXP.GNFS.K
5277	ZWE	2002	5.438714e+09			0	NE.EXP.GNFS.K
5278	ZWE	2001	6.549891e+09			0	NE.EXP.GNFS.K
5279	ZWE	2000	6.342110e+09			0	NE.EXP.GNFS.K

5280 rows × 10 columns

Una vez leída la estructura, podemos transformarla a series de tiempo usando `pivot_table`.

```
df_expo_series = df_expo.pivot_table(
    index='date',
    columns='country.value',
    values='value'
)
df_expo_series
```

country.value	Ingreso mediano	Afganistán	Albania	Alemania	América Latina y el Caribe
date					
2000	NaN	NaN	9.679770e+08	8.857566e+11	8.544018e+11
2001	NaN	NaN	1.085927e+09	9.359548e+11	8.660872e+11
2002	NaN	NaN	1.161242e+09	9.755377e+11	8.757786e+11
2003	3.757331e+12	NaN	1.333395e+09	9.938898e+11	9.004039e+11
2004	4.230782e+12	NaN	1.561785e+09	1.108500e+12	1.026605e+12
2005	4.613180e+12	NaN	1.899999e+09	1.182628e+12	1.104833e+12
2006	4.986643e+12	NaN	2.018742e+09	1.328005e+12	1.129454e+12
2007	5.247727e+12	NaN	2.444847e+09	1.446034e+12	1.164076e+12
2008	5.478234e+12	NaN	2.426776e+09	1.474102e+12	1.171799e+12
2009	5.006375e+12	NaN	2.671156e+09	1.263593e+12	1.066679e+12
2010	5.499933e+12	2.566011e+09	3.337086e+09	1.445674e+12	1.165024e+12
2011	5.842524e+12	NaN	3.559806e+09	1.566582e+12	1.236437e+12
2012	6.031301e+12	NaN	3.570257e+09	1.611922e+12	1.264077e+12
2013	6.143712e+12	NaN	3.626988e+09	1.628115e+12	1.273759e+12
2014	6.284015e+12	NaN	3.668800e+09	1.706201e+12	1.288153e+12
2015	NaN	NaN	3.704364e+09	1.799221e+12	1.347295e+12
2016	NaN	NaN	4.124461e+09	1.842402e+12	1.380054e+12
2017	NaN	NaN	4.670682e+09	1.932364e+12	1.432290e+12
2018	NaN	NaN	4.862305e+09	1.973746e+12	1.490922e+12
2019	NaN	NaN	5.156592e+09	1.993177e+12	1.500973e+12

20 rows × 233 columns

Hay otros recursos de la API del BM que sirven para pedir otras cosas, como `country` que devuelve la lista de países y algunos de sus atributos.

```

países = requests.get(
    'http://api.worldbank.org/v2/es/country?format=json&per_page=1000'
).json()

df_paises = pd.json_normalize(países[1])
df_paises

```

	<b>id</b>	<b>iso2Code</b>	<b>name</b>	<b>capitalCity</b>	<b>longitude</b>	<b>latitude</b>	<b>region.id</b>	<b>region.iso2</b>
0	ABW	AW	Aruba	Oranjestad	-70.0167	12.5167	LCN	
1	AFG	AF	Afganistán	Kabul	69.1761	34.5228	SAS	
2	AFR	A9					NA	
3	AGO	AO	Angola	Luanda	13.242	-8.81155	SSF	
4	ALB	AL	Albania	Tirana	19.8172	41.3317	ECS	
...	...	...	...	...	...	...	...	...
299	XZN	A5					NA	
300	YEM	YE	Yemen, Rep. del	Saná	44.2075	15.352	MEA	
301	ZAF	ZA	Sudáfrica	Ciudad del Cabo	28.1871	-25.746	SSF	
302	ZMB	ZM	Zambia	Lusaka	28.2937	-15.3982	SSF	
303	ZWE	ZW	Zimbabwe	Harare	31.0672	-17.8312	SSF	

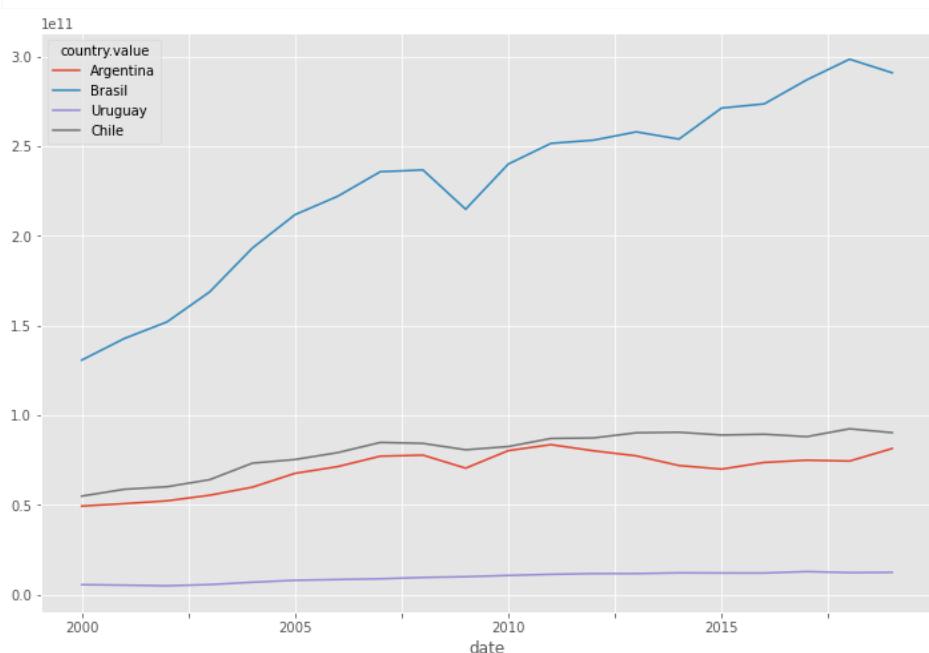
304 rows × 18 columns

Esto puede ser util para buscar los ids de los paises de interes, si queremos armar una muestra.

```
df_paises[df_paises['region.id'] == "LCN"]
```

```
df_expo_series[['Argentina', 'Brasil', 'Uruguay', 'Chile']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3a0475670>
```



Tal vez comparar la evolucion absoluta de exportaciones entre paises no tiene mucho sentido (tienen tamaños distintos). Podemos pedir mas de un indicador para esta muestra de paises, y calcular las exportaciones per capita.

Se pueden solicitar varios paises o varios indicadores, separando con ":".

```
expo_poblacion = requests.get(
    BASE_BM_API.format(
        pais='arg;bra;ury;chl',
        indicador='SP.POP.TOTL;NE.EXP.GNFS.KD'
    ) + '&source=2'
).json()

df_expo_pobl = pd.json_normalize(expo_poblacion[1])
```

Necesitamos quedarnos con dos columnas que podamos dividir, y asi calcular nuestro propio indicador de exportaciones per capita.

```
df_expo_pobl_pivot = df_expo_pobl.pivot_table(
    index=['country.id', 'date'],
    columns='indicator.id',
    values='value'
)
df_expo_pobl_pivot
```

		indicator.id	NE.EXP.GNFS.KD	SP.POP.TOTL
country.id	date			
AR	2000	4.928934e+10	36870787.0	
	2001	5.064036e+10	37275652.0	
	2002	5.220664e+10	37681749.0	
	2003	5.533662e+10	38087868.0	
	2004	5.982672e+10	38491972.0	
...				
UY	2015	1.196635e+10	3412009.0	
	2016	1.194755e+10	3424132.0	
	2017	1.277231e+10	3436646.0	
	2018	1.215714e+10	3449299.0	
	2019	1.230619e+10	3461734.0	

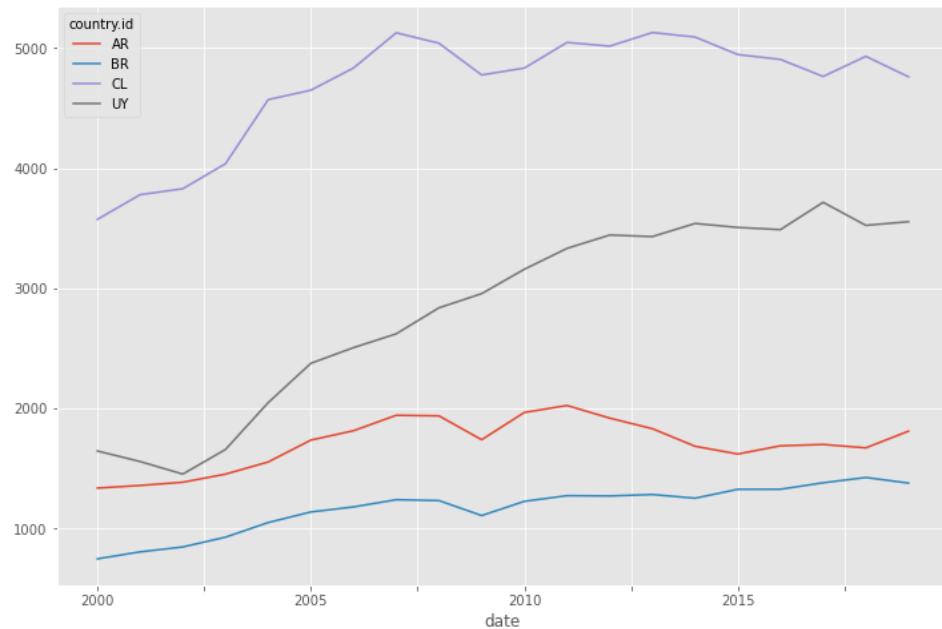
80 rows × 2 columns

```
df_expo_pobl_pivot['exportaciones_per_capita'] =
df_expo_pobl_pivot['NE.EXP.GNFS.KD'] / df_expo_pobl_pivot['SP.POP.TOTL']
```

Ahora si! Podemos generar las series de tiempo por pais de exportaciones per capita, y graficarlas.

```
df_expo_pobl_pivot.pivot_table(
    index='date',
    columns='country.id',
    values='exportaciones_per_capita'
).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3a0564a90>
```



Open in Colab

Recordá abrir en una nueva pestaña

## 37. APIs Series de Tiempo: Ejercicios

```
import pandas as pd
```

### 37.1. Ejercicio 1: API de Series de Tiempo de Argentina

- Genera una tabla y grafica la evolucion de los tipos de cambio ARS/USD de todas las entidades financieras (canal electronico, venta, 15hs).
- Genera un reporte automatico en HTML que diga las ultimas temperaturas diarias y el promedio de los ultimos 30 dias, para 3 ciudades de Argentina
- Grafica la relacion entre el nivel de precios (nucleo) y la base monetaria. Podes buscar la base monetaria en [este dataset](#) bajo el nombre de "Saldo de la base monetaria" y la serie de nivel de precios [es esta](#). Algunos scatter a probar:
  - IPC vs. base monetaria
  - IPC promedio 6 meses vs. base monetaria promedio 6 meses (Pista: usa rolling() y mean()).
  - IPC promedio 6 meses (variacion porcentual) vs. base monetaria promedio 6 meses (variacion porcentual) (Pista: agregale pct\_change(1) al anterior).
  - IPC promedio 6 meses (variacion porcentual) vs. base monetaria promedio 6 meses (variacion porcentual) de hace 3 meses -rezago de 3 meses- (Pista: agregale shift(3) a una de las variables).

Que otras variables se podrian incorporar para explicar o controlar esta relacion? Nivel de actividad? Tipo de cambio? Tasa de interes?

### 37.2. Ejercicio 2: API de Quandl

- Grafica las tasas de interes de los bonos de Estados Unidos, a partir del dataset de FRED disponible en Quandl (Pista: podes arrancar a buscar por aca: <https://www.quandl.com/data/FRED-Federal-Reserve-Economic-Data?keyword=10 years treasury>)

### 37.3. Ejercicio 3: API de Banco Mundial

- Grafica la evolucion de las emisiones per capita de CO2 para por lo menos 8 paises de Sudamerica desde 1960 (o el primer año con datos).

## 38. Generando contenido

Vamos a usar Google Colaboratory para mantenernos sincronizados y aprovechar la portabilidad que ofrece.

La notebook debe ser autosuficiente, sirviendo como material para exposición, con ejercicios prácticos orientados al dominio al que se apunta, complementado por una notebook de ejercicios y soluciones.

### 38.1. Estilo de las clases

- Partes separadas con títulos grandes y subsecciones jerárquicas con #
- Explicaciones en texto plano (markdown)
- Comentarios solo para especificar parámetros, argumentos específicos en ejemplos u otras cuestiones técnicas.
- Usar HTML para hipervínculos.
- Indentación, negrita o indentacion con formato código son preferibles cuando se hable de términos de software como "booleano", *keywords* de Python
- Respetar las indicaciones de [PEP 8](#) como manual de estilo.

Las clases se componen de una o dos (como mucho) notebooks. Cada notebook incluye ejercicios a ser completados. Salvando que sean casos demasiado fáciles (como las primeras clases de intro a Python) se pide que haya una notebook sin ejercicios completados y una con los ejercicios completados, para ayudar al docente. El sufijo -solucion indica en el nombre si incluye la solución o no. Además se debe incluir una carpeta ejercicio/ donde se incluyen ejercicios para realizar fuera de la clase. También debe incluirse la solución.

Ejemplo:

### 38.2. Operaciones con variables

#### 38.3. Variables

##### 38.3.1. Booleanos

Las variables de tipo *bool* aceptan ciertos operadores lógicos o relacionales:

- **&, and** : simbolizan la operación conocida de "y", ^, o intersección...
- ...

Aparecen en distintos contextos blabla... como las sentencias condicionales o *ifs*...

```
if (10 > 5) & ('palabras' in ['palabras', 'algo', 'cosas']) and True:  
    print('Verdadero!')
```

```
# Declaramos un bool con valor verdadero, o sea True o 1  
esta_soleado = 1  
hay_cuarentena = 1  
  
if esta_soleado & hay_cuarentena:  
    print('Uhh qué lástima!')
```

```
Uhh qué lástima!
```

Los ejercicios:

- Deben formularse en infinitivo
- Pueden tener secciones con títulos
- Las consignas deben estar escritas en Markdown con texto plano (no en comentarios, o todo con ##)
- Deben estar todos juntos al final de la notebook, o en una distinta, y las soluciones en otra ipynb

### 38.4. Operaciones numéricas

Calcular la raiz cuadrada de 5 y redondear a 3 decimales

```
round(5 ** (1/2), 3)
```

```
2.236
```

### 38.5. Principales convenciones de PEP8

```
# Recommended naming convention for variables, functions and class methods
under_score_naming_convention = 'default'
```

```
# Recommended
class Foo:
    def _private_method(self):
        return 'private'

    def public_method(self):
        return 'public'
```

```
# Recommended
total = (first_variable
         + second_variable
         - third_variable)
```

Indentation following line breaks

```
# Recommended
def function(arg_one, arg_two,
            arg_three, arg_four):
    return arg_one
```

Otros:

- <https://google.github.io/styleguide/pyguide.html>
- <https://refactoring.guru/>
- Black para formato
- pylint para detectar conflictos
- isort para ordenar imports

---

By Matías Grinberg

© Copyright 2021.