

# Prácticas de Aprendizaje Automático

## Clase 1: Introducción a Python

Pablo Mesejo y Salvador García

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

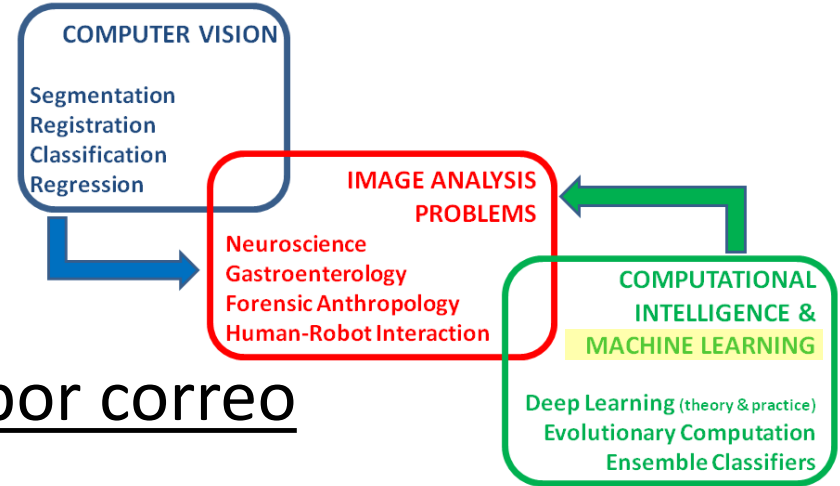


UNIVERSIDAD  
DE GRANADA



# Profesor

- Pablo Mesejo Santiago
- Email: [pmesejo@go.ugr.es](mailto:pmesejo@go.ugr.es)
- Tutorías:
  - Concertar cita previamente por correo
  - Oficialmente
    - Martes y Miércoles: 10:00-13:00
  - Extraoficialmente
    - Cualquier día a cualquier hora



# Profesor

- Salvador García López
- Email: [salvagl@decsai.ugr.es](mailto:salvagl@decsai.ugr.es)
- Tutorías:
  - Concertar cita previamente por correo
  - Oficialmente
    - Miércoles y Viernes: 10:00-13:00
  - Extraoficialmente
    - Cualquier día a cualquier hora

<https://scholar.google.com/citations?hl=en&user=vIC06a0AAAAJ>

# Materiales de que disponéis

- 2 presentaciones de introducción a Python, NumPy, Matplotlib, y Scikit-learn (dos primeras semanas de clase).
- 1 guía sobre cómo redactar y estructurar un informe académico-científico.
- 1 Notebook de Google Colab con la información de estas presentaciones (y más cosas).
- 1 presentación con algunos ejercicios, preguntas, y detalles técnicos a los que prestar atención.

# Objetivo

- **QUE APRENDÁIS (Y APROBÉIS)**
  - No quiero que nadie apruebe sin aprender
  - Pero tampoco quiero que nadie aprenda y suspenda
- El objetivo es doble: aprender y aprobar
  - Lo primero suele implicar lo segundo

# Objetivo (y 2)

- Mi objetivo es mejorar como profesor
  - No dudéis en darme ***feedback*** para mejorar
    - Tengo muy en cuenta vuestros comentarios y consejos, tanto a nivel de impartir las clases como de organizar las asignaturas
  - ¡Acordaos de cubrir las **encuestas de evaluación de la calidad docente!**

# Dudas

- No dudéis en preguntar y solicitar toda la información que necesitéis.

**Todos somos ignorantes. Lo que pasa que ignoramos cosas diferentes.**

- ¡Aprovechad las horas de clase!
- Fuera de horas de clase:
  - emplead el email indicado ([pmesejo@go.ugr.es](mailto:pmesejo@go.ugr.es) / [salvagl@decsai.ugr.es](mailto:salvagl@decsai.ugr.es)) para consultarme dudas offline
  - emplead el email indicado ([pmesejo@go.ugr.es](mailto:pmesejo@go.ugr.es) / [salvagl@decsai.ugr.es](mailto:salvagl@decsai.ugr.es)) para solicitarme tutorías (**recomendable enviar también las dudas por email, porque en ocasiones es muy rápido resolverlas y no tenéis que esperar a clase o tutoría**)
- Dudas de teoría al profesor/a de teoría (y de prácticas a vuestro/a profesor/a de prácticas).
  - Es quien os evaluará de esa parte, y preguntarle directamente será más efectivo.

# Evaluación Continua

- No dudéis tampoco en solicitar **feedback**, en cualquier momento del curso, **para conocer si vais alcanzando** progresiva y adecuadamente **las competencias esperadas**.
- Las calificaciones intentaré ir dándolas a lo largo del curso (junto con el feedback asociado)
  - Pero debido a mi carga docente no creo que me dé tiempo a dar ninguna nota antes de mediados-finales de Mayo.
  - La 1ª entrega sería, en cualquier caso, a mediados de Abril.



# Sobre este curso (1)

- Esto no es un curso sobre Python
  - No tenéis que convertirlos en expertos en Python ni dominar todas las funcionalidades
  - No se evaluará la elegancia del código
    - Al margen de que el código entregado tiene que poder ejecutarse y resolver el problema indicado
- Python se considera una mera herramienta para resolver problemas de Aprendizaje Automático

# Sobre este curso (y 2)

- Asistencia a prácticas no obligatoria
- Podéis ir al grupo de prácticas que queráis, pero...
  - ... al estar apuntados a este grupo, seré yo quien evalúe vuestros trabajos

# Temporización Prácticas

## Calendario Aproximado

- *Sesión 1: Semana 26 de Febrero (sin clase Miércoles)*
  - Seminario1. Introducción a Python
- *Sesión 2: Semana 04 de Marzo*
  - Seminario2. Introducción a NumPy, Matplotlib y Scikit-learn
- *Sesión 3: Semana 11 de Marzo*
  - Presentación Práctica 1 - Experimentación con **clasificadores y regresores**
- *Sesión 4: Semana 18 de Marzo*
  - Práctica 1 - Seguimiento/Dudas
- *SEMANA SANTA*
- *Sesión 5: Semana 01 de Abril (sin clase Lunes)*
  - Práctica 1 - Seguimiento/Dudas
- *Sesión 6: Semana 08 de Abril*
  - Práctica 1 - Seguimiento/Dudas


**Fecha límite de entrega de la P1:**  
14 de Abril

# Temporización Prácticas

## Calendario Aproximado

- Sesión 7: Semana 15 de Abril
    - Presentación Práctica 2 - Experimentación con algoritmos de **aprendizaje no supervisado**
  - Sesión 8: Semana 22 de Abril
    - Práctica 2 - Seguimiento/Dudas
  - Sesión 9: Semana 29 de Abril (**sin clase Miércoles**)
    - Práctica 2 - Seguimiento/Dudas
  - Sesión 10: Semana 06 de Mayo
    - Presentación Práctica 3 - Experimentación con algoritmos de **aprendizaje profundo**
  - Sesión 11: Semana 13 de Mayo
    - Práctica 3 / Proyecto Final - Seguimiento/Dudas
  - Sesión 12: Semana 20 de Mayo
    - Práctica 3 / Proyecto Final - Seguimiento/Dudas
  - Sesión 13: Semana 27 de Mayo (**sin clase Viernes**)
    - Práctica 3 / Proyecto Final - Seguimiento/Dudas
  - Sesión 14: Semana 03 de Junio (**sin clase Miércoles y Viernes**)
    - Práctica 3 / Proyecto Final - Seguimiento/Dudas
- Fecha límite de entrega de la P2: 12 de Mayo**
- Fecha límite de entrega de la P3: 04 de Junio**
- Entrega Proyecto Final: Junio**

# Evaluación de las prácticas

- Con cada trabajo/práctica se entrega un **Notebook que integra código, resultados y discusión.**
- **La discusión de los resultados, y el análisis y descripción del trabajo realizado, resultan fundamentales** 
  - Es importante que los resultados sean correctos, pero todavía lo es más demostrar que se entiende lo que se está haciendo.
- Sistema de evaluación:
  - Revisar materiales en PRADO proporcionados por el coordinador de la asignatura

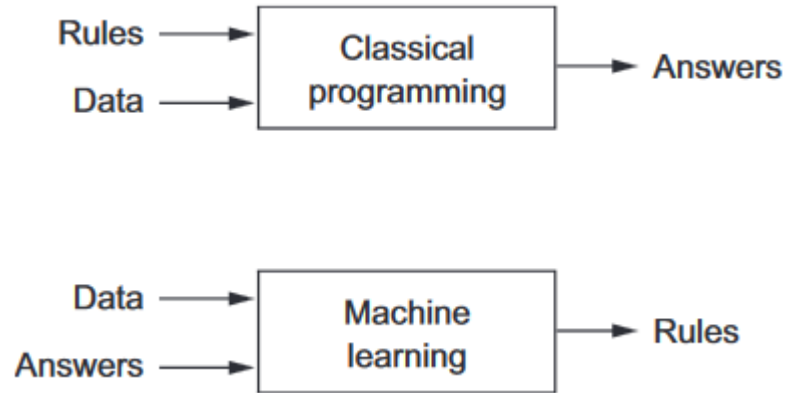
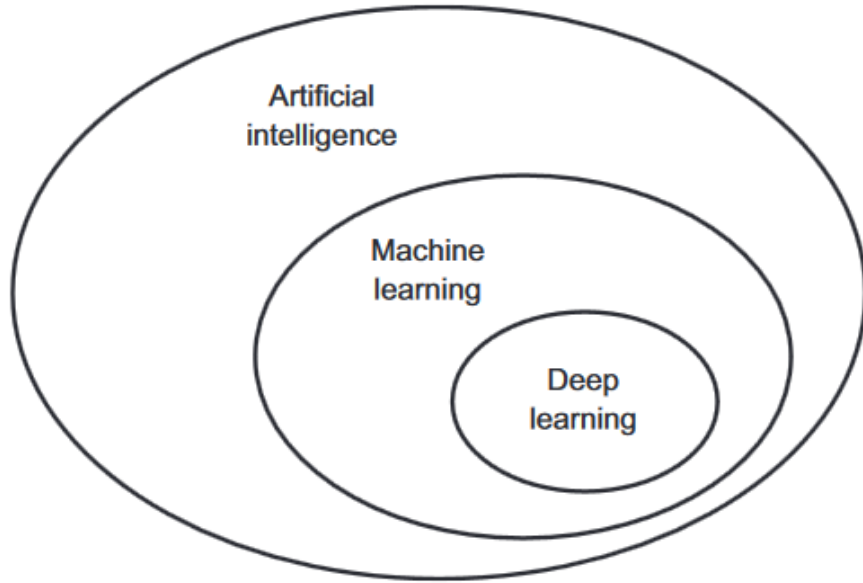
# Índice

1. ¿Qué es el Aprendizaje Automático?
2. ¿Por qué usamos Python y Scikit-learn en las prácticas?
3. Uso básico de Google Colab
4. Primeros pasos con Python
5. Listas, tuplas y diccionarios
6. Indexado
7. Estructuras condicionales y bucles
8. Funciones
9. Clases

# ¿Qué es el Aprendizaje Automático?

- Que las máquinas aprendan a partir de los propios datos:  
datos + algoritmos de aprendizaje = aprendizaje automático
  - “Usar un conjunto de observaciones para descubrir un proceso subyacente”  
(Learning From Data, Abu-Mostafa et al., 2012)
    - Existe un patrón
    - No podemos describirlo matemáticamente
    - Tenemos datos
- Ejemplo:** recomendador de películas
- Nuestros gustos no son arbitrarios, sino que siguen ciertos patrones.
  - No podemos definir matemáticamente porqué nos gusta lo que nos gusta.
  - Tenemos ejemplos de otras películas que nos han gustado.

# ¿Qué es el Aprendizaje Automático? (y 2)



Figuras extraídas de “Deep Learning with Python” (F. Chollet, 2018)



# ¿Por qué estudiar Aprendizaje Automático?

- Rama de la Inteligencia Artificial más pujante actualmente
- Muchísimas aplicaciones
  - Visión por computador y procesamiento de imágenes
  - Procesado de señales y reconocimiento del habla
  - Traducción automática
  - Conducción autónoma
  - Juegos y videojuegos
  - Supera las capacidades humanas en la realización de numerosas actividades complejas (p.ej. predicción de la estructura de proteínas)
  - ...



# ¿Por qué estudiar Aprendizaje Automático? (y 2)

Por todo ello,

mucho trabajo (y, generalmente, bien pagado)

Entre los trabajos mejor pagados: data scientist, AI expert,...

<https://www.edix.com/es/instituto/trabajos-mejor-pagados/>

<https://www.puromarketing.com/12/35699/profesionales-dats-entre-mejores-pagados-sector-ittelco.html>

<https://www.xataka.com/robotica-e-ia/estos-trabajos-futuro-linkedin-especialistas-inteligencia-artificial-cobran-140-000-dolares-anuales>

Estados Unidos

**\$112,289 / year** ▾

Avg. Base Salary (USD)



The average salary for a Machine Learning Engineer is \$112,289

**Base Salary** ⓘ

\$77k - \$154k

**Bonus**

\$3k - \$21k

**Profit Sharing**

\$1k - \$38k

**Total Pay** ⓘ

\$75k - \$165k

Based on 1,306 salary profiles (last updated Jan 12 2022)

Alemania

**€53,481 / year** ▾

Avg. Base Salary (EUR)



The average salary for a Machine Learning Engineer is €53,481

**Base Salary** ⓘ

€41k - €75k

**Bonus**

€995 - €10k

**Profit Sharing**

€4 - €40k

**Total Pay** ⓘ

€40k - €78k

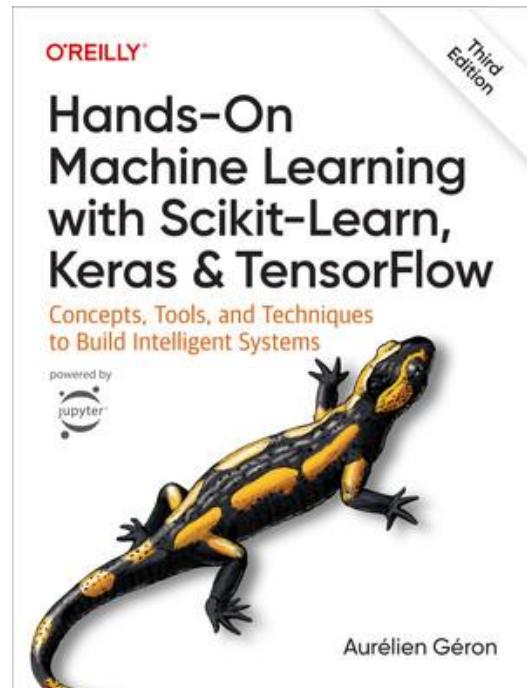
Based on 249 salary profiles (last updated Jan 07 2022)

# Sobre este curso de Aprendizaje Automático

Géron, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2022.

- Notebooks disponibles online:

<https://github.com/ageron/handson-ml3>



# Otra referencia de posible interés

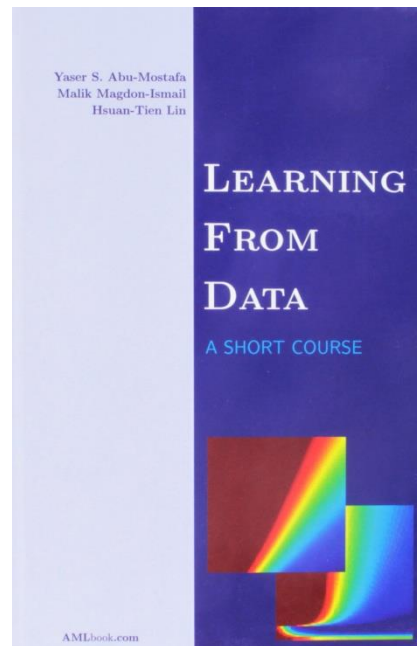
Abu-Mostafa, Yaser S., Malik Magdon-Ismael, and Hsuan-Tien Lin. *Learning from data*. Vol. 4. New York: AMLBook, 2012.

- Diapositivas y vídeos disponibles online:

<https://work.caltech.edu/lectures.html>

- **Lecture 1 (The Learning Problem)**  
[Lecture](#) (some audio drops, sorry!) - [Q&A](#) - [Slides](#)
- **Lecture 2 (Is Learning Feasible?)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 3 (The Linear Model I)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 4 (Error and Noise)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 5 (Training versus Testing)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 6 (Theory of Generalization)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 7 (The VC Dimension)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 8 (Bias-Variance Tradeoff)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 9 (The Linear Model II)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)

- **Lecture 10 (Neural Networks)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 11 (Overfitting)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 12 (Regularization)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 13 (Validation)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 14 (Support Vector Machines)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 15 (Kernel Methods)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 16 (Radial Basis Functions)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 17 (Three Learning Principles)**  
[Review](#) - [Lecture](#) - [Q&A](#) - [Slides](#)
- **Lecture 18 (Epilogue)**  
[Review](#) - [Lecture](#) - [Acknowledgment](#) - [Slides](#)



# Un poquito de Historia

- [McCulloch, W. S., & Pitts, W.](#) (1943). *“A logical calculus of the ideas immanent in nervous activity”*. The bulletin of mathematical biophysics, 5, 115-133.
  - Primer modelo neuronal: solo permitía **entradas y salidas binarias**, empleaba **función de activación umbral/escalón** (*threshold step*), y **no incorporaba pesos en las entradas**.
- [Samuel, Arthur L.](#) (1959). *“Some Studies in Machine Learning Using the Game of Checkers”*. IBM Journal of Research and Development. 44: 206–226.
  - Introducción del término “machine learning” (*self-teaching computers*)
- [Rosenblatt, F.](#) (1962). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan books.
  - Discusión y extension del concepto de **Perceptron** (inicialmente desarrollado por Rosenblatt en 1957-58)
- Otros modelos, como [Adaline](#):
  - Widrow, B. (1960). *An adaptive “ADALINE” neuron using chemical “memistors”*. Tech.Report Nº 1553-2. Office of Naval Research.
  - El Adaline tiene una regla de aprendizaje diferente del Perceptron y emplea una función de activación distinta (lineal).

# ¿Por qué Python? (1)

- + **Lenguaje de propósito general de alto nivel** creado a finales de los '80 por Guido Van Rossum (actualmente *Distinguished Engineer at Microsoft*)



- + Cálculo científico de modo **similar a Matlab/Octave**

<https://numpy.org/doc/stable/user/numpy-for-matlab-users.html>

<https://bastibe.de/2013-01-20-a-python-primer-for-matlab-users.html>

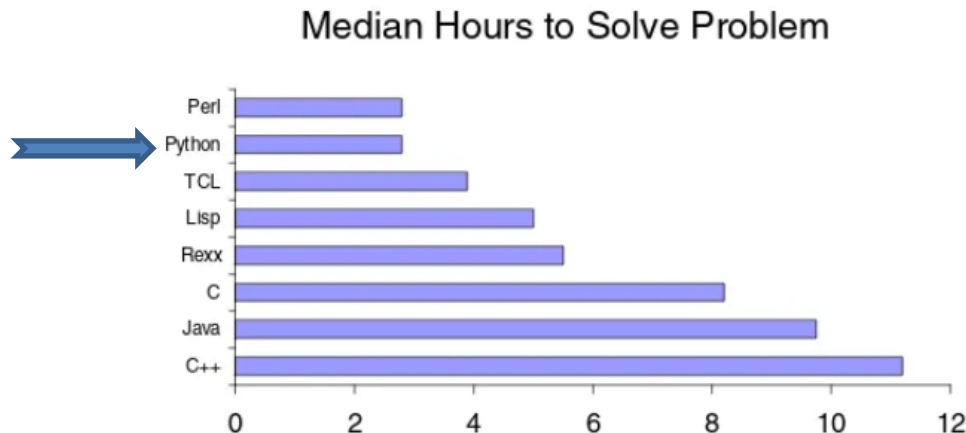
- + **Software libre** (Python Software Foundation License) **y gratuito** (*free and open-source software*)

# ¿Por qué Python? (2)

+ Lenguaje interpretado que soporta programación orientada a objetos y que cuenta con una sintaxis simple e intuitiva

➤ **Resulta fácil comenzar a trabajar con Python**

➤ **Permite el desarrollo rápido de aplicaciones**

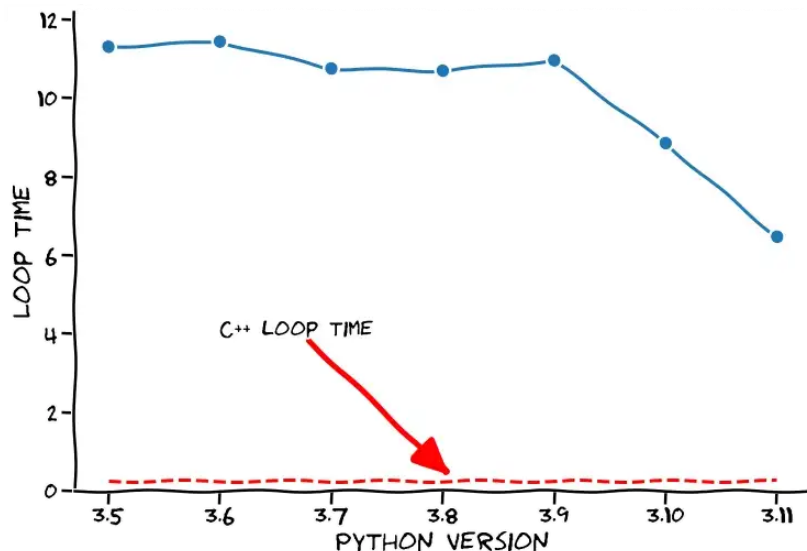


<https://medium.com/pyslackers/yes-python-is-slow-and-i-dont-care-13763980b5a1>

# ¿Por qué Python? (3)

+ Aunque más lento que otros lenguajes, en numerosas aplicaciones, es **suficiente y razonablemente rápido** (y cada vez lo es más)

➤ Hay librerías de Python que sí ofrecen una ejecución rápida (e.g. NumPy)



<https://towardsdatascience.com/python-3-14-will-be-faster-than-c-a97edd01d65d>

<https://towardsdatascience.com/why-is-python-so-slow-and-how-to-speed-it-up-485b5a84154e>



# ¿Por qué Python? (y 4)

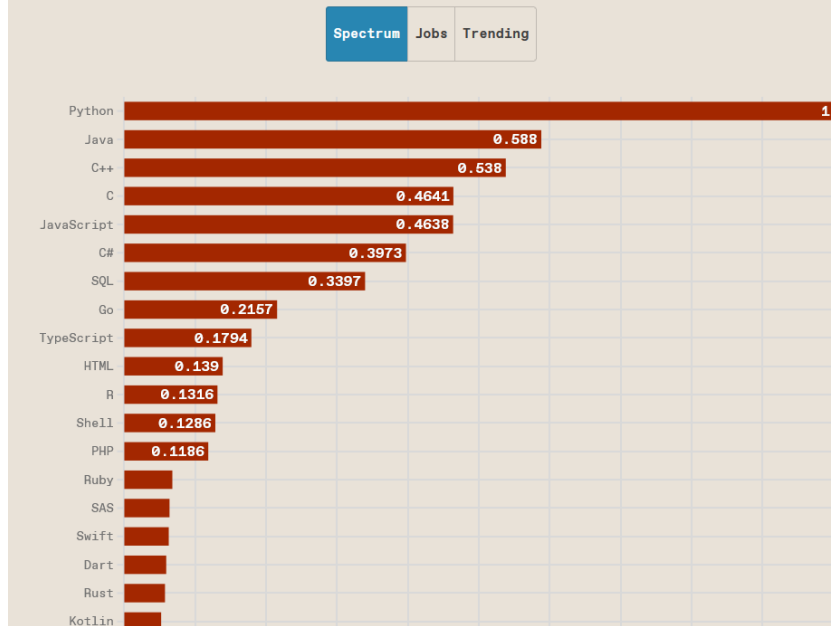
- + Utilizado por los principales frameworks de **deep learning** (PyTorch, Theano, Keras, TensorFlow)
- + Debido a todo lo anterior
  - **ampliamente usado**
  - **mucha ayuda disponible** (foros, documentación)

**Python** claimed **#1 in the Job Postings** category and **#3 in Average Salary** (#1 C# and #2 C++). It excels in the areas of data science tools, AI and machine learning, cybersecurity, and DevOps engineering.

<https://www.codeplatoon.org/the-highest-paying-and-most-in-demand-programming-languages-in-2023/>

## Top Programming Languages 2023

Click a button to see a differently weighted ranking



<https://spectrum.ieee.org/the-top-programming-languages-2023>

# ¿Por qué Scikit-learn?

## + Paquete/librería de Python para aprendizaje automático

Gran cantidad de algoritmos y funciones para el tratamiento y análisis de datos

Originalmente desarrollada por David Cournapeau en 2007

## + Construido sobre las librerías NumPy, SciPy y Matplotlib

## + Software libre (licencia BSD)

## + Muchas librerías compatibles (incluyendo para Deep Learning).



# ¿Por qué Scikit-learn?

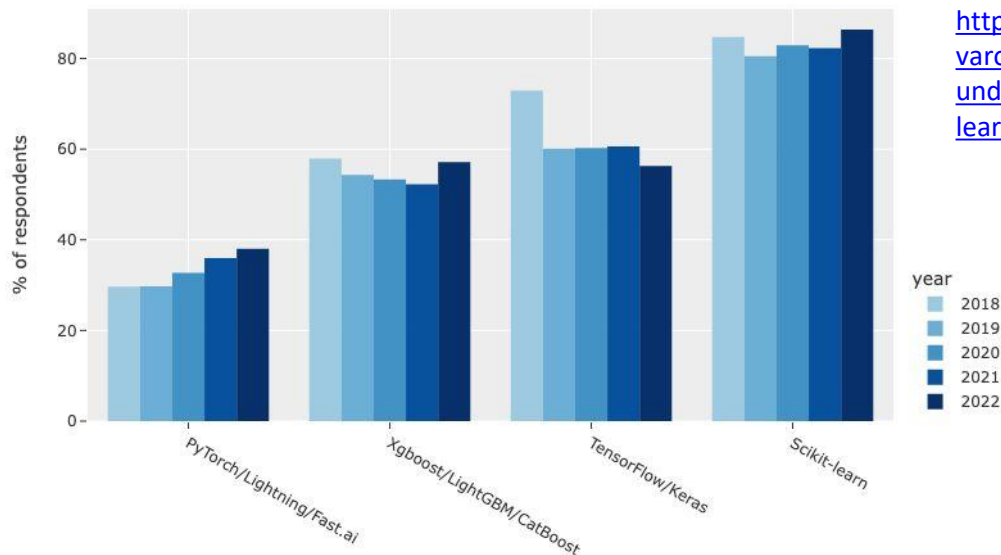
## + Paquete/librería de Python para aprendizaje automático

Encuesta realizada por Kaggle en 2022

(<https://www.kaggle.com/kaggle-survey-2022>):

“Scikit-learn is the most popular ML framework while PyTorch has been growing steadily year-over-year”

Most popular consolidated machine learning frameworks (2018-2022)



<https://gael-varoquaux.info/programming/people-underestimate-how-impactful-scikit-learn-continues-to-be.html>



**François Chollet**  @fchollet · Nov 22

People underestimate how impactful Scikit-Learn continues to be.

# De cara a realizar las prácticas: Notebooks de Google Colab

<https://colab.research.google.com/>

- Con cada práctica entregaréis un fichero *.ipynb* de Colab.
- En dicho fichero se incorpora código, resultados, y discusión y análisis de los mismos. Es decir, un fichero de Colab integra implementación y memoria.
- **El uso de Colab no significa que podáis realizar una memoria de menor calidad o menor grado de detalle que la que redactaríais en un PDF independiente.**

# ¿Qué es Colab?

- Servicio web de Google que permite ejecutar código Python en el navegador
  - No requiere que instales o configures nada en tu equipo
  - Permite combinar código, resultados y texto de forma cómoda → permite integrar memoria, resultados y código en un único fichero
  - Permite emplear los recursos hardware (GPU) de Google

# ¿Qué es Colab?

- Se recomienda revisar con atención la documentación oficial:
  - <https://colab.research.google.com/notebooks/intro.ipynb>



The screenshot displays the Google Colaboratory (Colab) web interface. At the top, a welcome message reads "Te damos la bienvenida a Colaboratory". Below this, a navigation bar includes links for "Archivo", "Editar", "Ver", "Insertar", "Entorno de ejecución", "Herramientas", and "Ayuda". On the left side, there is a sidebar with a search icon and a list of items: "Índice", "Primeros pasos", "Ciencia de datos", "Aprendizaje automático", "Más recursos", "Ejemplos destacados", and "Sección". The main content area features a large video player with a thumbnail showing a man and the text "3 Cool Google Colab Features". Below the video, the text "¿Qué es Colaboratory?" is followed by a description: "Colab, también conocido como 'Colaboratory', te permite programar y ejecutar Python en tu navegador con las siguientes ventajas:". A bulleted list of advantages follows: "No requiere configuración", "Da acceso gratuito a GPUs", and "Permite compartir contenido fácilmente".

Te damos la bienvenida a Colab

Si ya conoces Colab, echa un vistazo a este vídeo para obtener información sobre las tablas interactivas, la vista del historial de código ejecutado y la paleta de comandos.

3 Cool Google Colab Features

¿Qué es Colaboratory?

Colab, también conocido como "Colaboratory", te permite programar y ejecutar Python en tu navegador con las siguientes ventajas:

- No requiere configuración
- Da acceso gratuito a GPUs
- Permite compartir contenido fácilmente

# Primeros pasos

Accedéis a la URL <https://colab.research.google.com/>

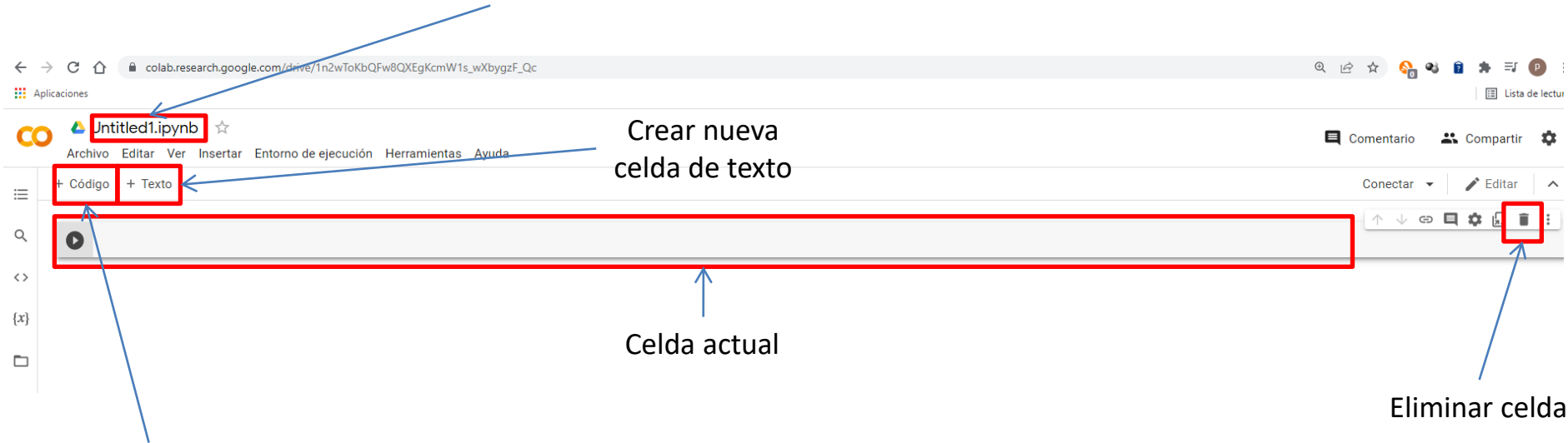
The screenshot shows the Google Colaboratory web interface. At the top, the browser address bar displays [colab.research.google.com](https://colab.research.google.com/), which is highlighted with a red box. Below the browser, the Colaboratory interface is visible. On the left, there is a sidebar with a search bar and a list of notebooks under the 'Índice' (Index) section. The main area displays a modal window with the 'Recientes' (Recent) tab selected. This tab shows a list of notebooks with columns for 'Ejemplos' (Examples), 'Recientes' (Recent), 'Google Drive', 'GitHub', and 'Subir' (Upload). The list includes notebooks like 'Te damos la bienvenida a Colaboratory', 'Index.ipynb', 'ml\_models\_scikit-learn.ipynb', '05.02-Introducing-Scikit-Learn.ipynb', and '2020-10-19-ScikitLearn-tutorial-part2.ipynb'. At the bottom right of the modal, the 'Nuevo cuaderno' (New notebook) button is highlighted with a red box.

Ejemplos	Recientes	Google Drive	GitHub	Subir
Filtrar cuadernos				
Título		Abierto por última vez	Abierto por primera vez	
Te damos la bienvenida a Colaboratory	15:58	17 sept 2020		
Index.ipynb	10 de febrero	10 de febrero		
ml_models_scikit-learn.ipynb	10 de febrero	10 de febrero		
05.02-Introducing-Scikit-Learn.ipynb	10 de febrero	10 de febrero		
2020-10-19-ScikitLearn-tutorial-part2.ipynb	10 de febrero	10 de febrero		

# Primeros pasos

Al crear un nuevo cuaderno (Notebook) veremos lo siguiente

Nombre del fichero. Extensión .ipynb (interactive **python** **notebook**)



The screenshot shows the Google Colab web interface. The browser address bar displays the URL: `colab.research.google.com/drive/1n2wToKbQFw8QXEgKcmW1s_wXbygzF_Qc`. The file name `Untitled1.ipynb` is highlighted with a red box and an arrow pointing to the text 'Nombre del fichero. Extensión .ipynb (interactive **python** **notebook**)'. The menu bar includes 'Archivo', 'Editar', 'Ver', 'Insertar', 'Entorno de ejecución', 'Herramientas', and 'Ayuda'. Below the menu, the '+ Código' and '+ Texto' buttons are highlighted with red boxes and an arrow pointing to the text 'Crear nueva celda de texto'. The main workspace contains a single code cell, which is highlighted with a large red rectangle and an arrow pointing to the text 'Celda actual'. On the right side of the cell, a toolbar contains icons for running, saving, and deleting. The delete icon (trash can) is highlighted with a red box and an arrow pointing to the text 'Eliminar celda'.

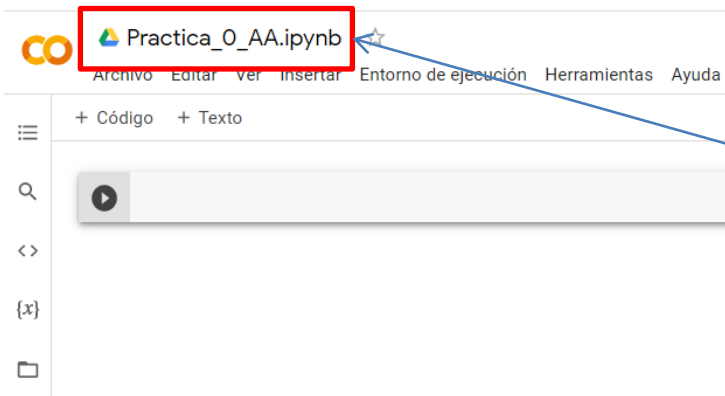
Crear nueva  
celda de código

Este notebook está en nuestro Google Drive.  
Podremos acceder desde allí también a él, y lo  
mismo con los datos que queramos emplear.





# Primeros pasos



En cuanto cambiamos el nombre del fichero, este se actualiza automáticamente en nuestro Google Drive.



# Primeros pasos



The screenshot displays the JupyterLab environment. At the top, the browser address bar shows 'Practica\_O\_AA.ipynb'. The main interface is divided into two panes. The left pane is in 'Edit' mode, showing a text cell with the following content: `<h1><center><b> Práctica 0 de Aprendizaje Automático </b></center></h1>`, `Autor: Pepito Grillo`, and `DNI: 12345678A`. The right pane shows the rendered output of the text cell, which is a centered bold heading 'Práctica 0 de Aprendizaje Automático', followed by 'Autor: Pepito Grillo' and 'DNI: 12345678A'. The interface includes various toolbars for file management, editing, and viewing system resources like RAM and disk space.

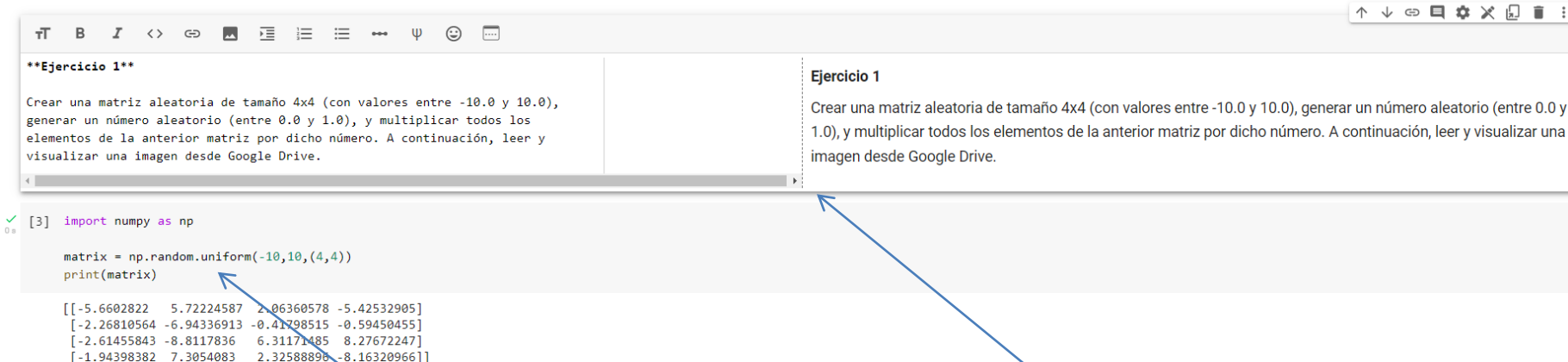
En una celda de texto, tendréis el texto/código (Markdown) a la izquierda, y la visualización del mismo a la derecha. Permite gran flexibilidad desde el punto de vista del formateo.

# Primeros pasos

## Práctica 0 de Aprendizaje Automático

Autor: Pepito Grillo

DNI: 12345678A



The screenshot shows a Jupyter Notebook interface. At the top, there is a toolbar with various icons for text formatting and editing. Below the toolbar, the notebook is divided into two main sections. The left section contains a text cell with the heading **\*\*Ejercicio 1\*\*** and a description: "Crear una matriz aleatoria de tamaño 4x4 (con valores entre -10.0 y 10.0), generar un número aleatorio (entre 0.0 y 1.0), y multiplicar todos los elementos de la anterior matriz por dicho número. A continuación, leer y visualizar una imagen desde Google Drive." The right section contains a code cell with the heading **Ejercicio 1** and the same description. Below the code cell, the output of the code is displayed, showing a 4x4 matrix of random values. Two blue arrows point from the text in the right section to the code and output in the left section.

**\*\*Ejercicio 1\*\***

Crear una matriz aleatoria de tamaño 4x4 (con valores entre -10.0 y 10.0), generar un número aleatorio (entre 0.0 y 1.0), y multiplicar todos los elementos de la anterior matriz por dicho número. A continuación, leer y visualizar una imagen desde Google Drive.

**Ejercicio 1**

Crear una matriz aleatoria de tamaño 4x4 (con valores entre -10.0 y 10.0), generar un número aleatorio (entre 0.0 y 1.0), y multiplicar todos los elementos de la anterior matriz por dicho número. A continuación, leer y visualizar una imagen desde Google Drive.


```
[3] import numpy as np

matrix = np.random.uniform(-10,10,(4,4))
print(matrix)
```

```
[[-5.6602822  5.72224587  2.06360578 -5.42532905]
 [-2.26810564 -6.94336913 -0.41798515 -0.59450455]
 [-2.61455843 -8.8117836  6.31171485  8.27672247]
 [-1.94398382  7.3054083  2.32588896 -8.16320966]]
```

Podéis ir intercalando **celdas de texto** con **celdas de código**, ejecutarlas, y ver los **resultados obtenidos**.

# Primeros pasos



```
import numpy as np

matrix = np.random.uniform(-10,10,(4,4))
print(matrix)
```

Para ejecutar una celda de código, solo tenéis que hacer click en el símbolo de “Play” (o hacer Ctrl+Enter)

Y el resultado de la ejecución, si no hay errores, aparecerá debajo

```
[3] import numpy as np

matrix = np.random.uniform(-10,10,(4,4))
print(matrix)
```

```
[[-5.6602822  5.72224587  2.06360578 -5.42532905]
 [-2.26810564 -6.94336913 -0.41798515 -0.59450455]
 [-2.61455843 -8.8117836  6.31171485  8.27672247]
 [-1.94398382  7.3054083  2.32588896 -8.16320966]]
```

```
[5] number = np.random.uniform()
print(number)
```

```
0.9444134094519612
```

```
[6] result = np.dot(matrix,number)
print(result)
```

```
[[-5.34564641  5.40416573  1.94889697 -5.1237535 ]
 [-2.14202939 -6.55741092 -0.39475078 -0.56145806]
 [-2.46922404 -8.3219666  5.96086814  7.81664769]
 [-1.83592438  6.89932556  2.19660072 -7.70944467]]
```

```
[[-5.6602822  5.72224587  2.06360578 -5.42532905]
 [-2.26810564 -6.94336913 -0.41798515 -0.59450455]
 [-2.61455843 -8.8117836  6.31171485  8.27672247]
 [-1.94398382  7.3054083  2.32588896 -8.16320966]]
```

# Primeros pasos

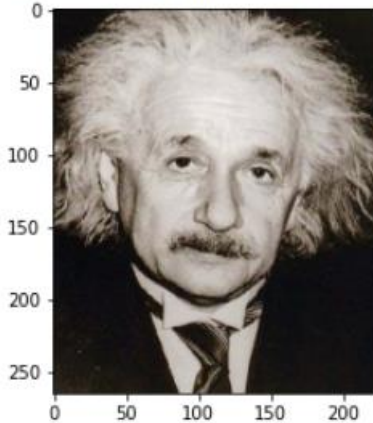
```
[8] from google.colab import drive  
drive.mount('/content/drive')
```

Para acceder a los ficheros de nuestro Drive debemos montarlo

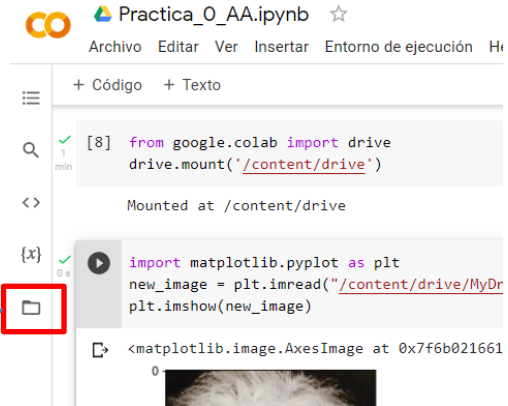
Esto nos permite, por ejemplo, visualizar una imagen

```
import matplotlib.pyplot as plt  
new_image = plt.imread("/content/drive/MyDrive/Colab Notebooks/images/einstein.bmp")  
plt.imshow(new_image)
```

<matplotlib.image.AxesImage at 0x7f6b00450650>

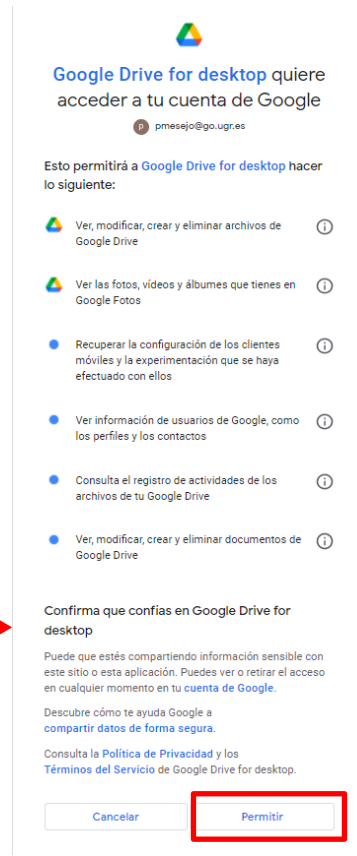
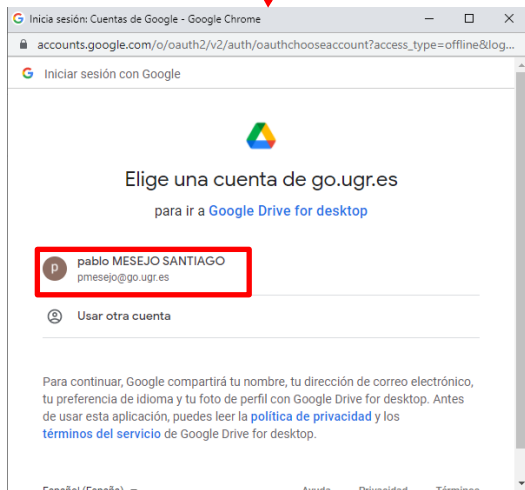
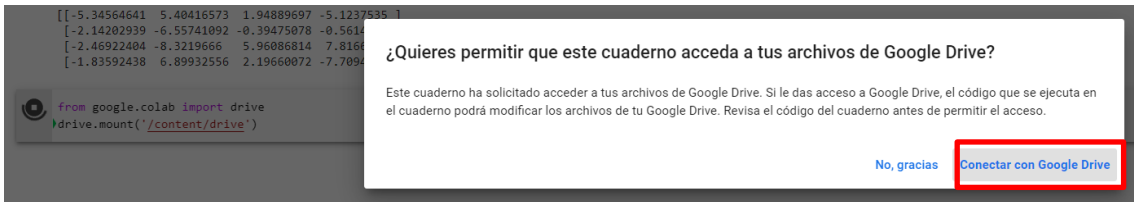


IMPORTANTE: aseguraos de que la ruta es adecuada. Para ello, listad los ficheros (empleando `!ls /content/`), o emplead el gestor de archivos a la izquierda (y dadle a “Copiar Ruta” del fichero de interés)



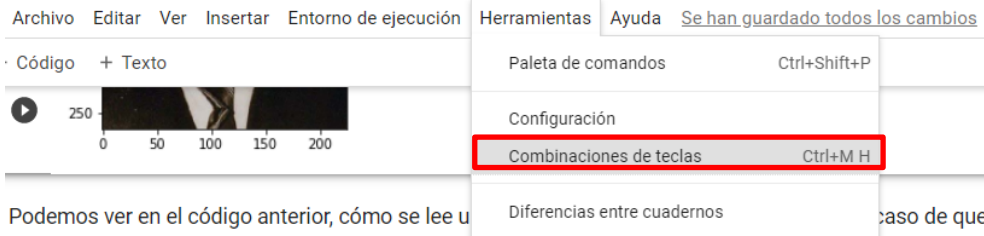
# Primeros pasos

## Sobre montar el Drive:

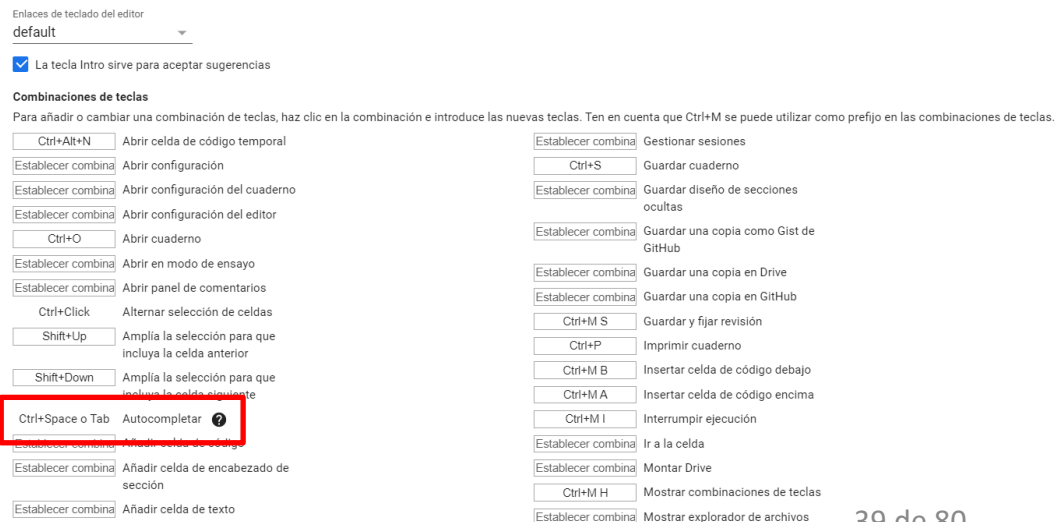


# Algunas ideas útiles

- Autocompletado:



Podemos ver en el código anterior, cómo se lee u



# Algunas ideas útiles



- Se pueden (y deben) emplear **índices** para estructurar el contenido del cuaderno:

Si hacéis doble click en la celda, abríreis el editor para poder modificarla.

[https://colab.research.google.com/notebooks/markdown\\_guide.ipynb](https://colab.research.google.com/notebooks/markdown_guide.ipynb)

Headings are rendered as titles.



# Algunas ideas útiles

- Inspector de variables.

The screenshot displays the Jupyter Notebook interface. On the left, the 'Variables' inspector is open, showing a table of variables. A red box highlights the '{x}' icon in the left sidebar. The table lists variables: list\_a (list, 4 items), list\_b (list, 2 items), w (ndarray, (2, 2)), and w2 (ndarray, (2, 2)). Below the table, there is a search bar and a dropdown menu for 'Mostrar valores en el editor de código'.

Nombre	Tipo	Forma	Valor
list_a	list	4 items	[1, 2, 3, 4]
list_b	list	2 items	[2, 3]
w	ndarray	(2, 2)	array([[4.74836781e-31
w2	ndarray	(2, 2)	array([[100., 100.], [100.

On the right, the code editor shows the following Python code:

```
import numpy as np
w = np.empty([2, 2])
print(w)

w2 = w + 100
print(w2)

list_a = [*range(1,5)]
list_b = [*range(2,4)]

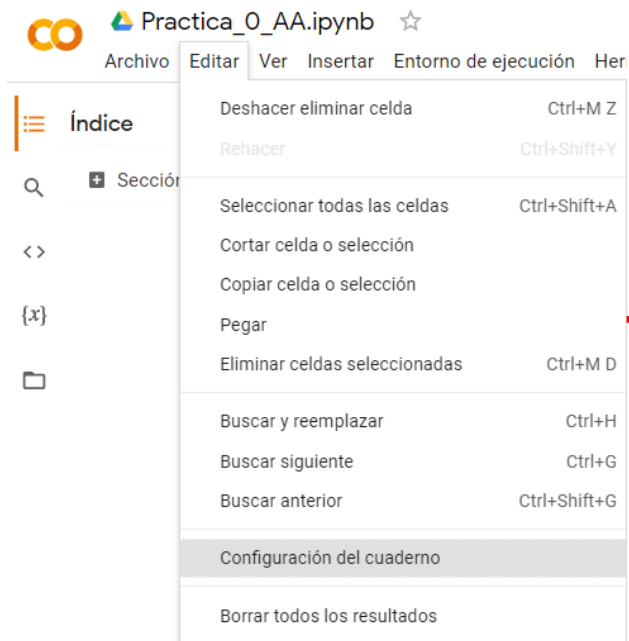
[a*b for a,b in zip(list_a, list_b)]
```

The output of the code is displayed below the code editor:

```
[[4.74836781e-310 0.00000000e+000]
 [6.79497456e-310 6.79497456e-310]]
[[100. 100.]
 [100. 100.]]
[2, 6]
```

# Algunas ideas útiles

- Podéis utilizar hardware de Google para acelerar la ejecución de experimentos. Por ejemplo, GPUs.



## Configuración del cuaderno

Acelerador por hardware

None

None

GPU

TPU

¿Quieres que tu cuaderno siga ejecutándose incluso después de cerrar el navegador?

[Pasarse a Colab Pro+](#)

☐ Omitir resultado de las celdas de código al guardar este cuaderno

Cancelar

Guardar

# Algunas ideas útiles

- Pero... usad los recursos con criterio.
  - De lo contrario, pueden surgir problemas de RAM en Colab
    - Más probable en tareas y campos en donde el consumo de memoria es intensivo (deep learning y computer vision).
  - Si tenéis problemas con la memoria RAM, **¡¡¡no paguéis por la versión Colab Pro!!!**
  - Si se programa bien, y se realizan experimentos razonables de modo ordenado, no debe haber ningún problema.

# Algunas ideas útiles

- No obstante, si tenéis problemas con la RAM de Colab
  - 1) Modificad los servicios de Google Colab. Por ejemplo, aumentad la RAM disponible en Colab (<https://analyticsindiamag.com/5-google-colab-hacks-one-should-be-aware-of/>).
  - 2) Optimizad el código. Se podría optimizar el tipo de dato empleado (p.ej. <https://stackoverflow.com/questions/62977311/how-can-i-stop-my-colab-notebook-from-crashing-while-normalising-my-images>). También es recomendable eliminar objetos innecesarios que puedan estar en memoria (comando `del`) y/o utilizar el *garbage collector* para liberar memoria (<https://stackoverflow.com/questions/61188185/how-to-free-memory-in-colab>).
  - 3) En último término, hablad con el profesor sobre la posibilidad de dividir el Notebook en varios ficheros a ejecutar de modo independiente. Al reiniciar el *runtime* no debería haber problema.

# Referencias de interés

- Overview of Colaboratory Features:  
[https://colab.research.google.com/notebooks/basic\\_features\\_overview.ipynb](https://colab.research.google.com/notebooks/basic_features_overview.ipynb)
- Introducción a NumPy, Matplotlib y Scikit-learn en Google Colab:  
<https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/Index.ipynb>
- Tutorial sobre Machine Learning with Scikit-learn en Colab:  
[https://colab.research.google.com/github/astg606/py\\_materials/blob/master/machine\\_learning/ml\\_models\\_scikit-learn.ipynb](https://colab.research.google.com/github/astg606/py_materials/blob/master/machine_learning/ml_models_scikit-learn.ipynb)
- LaTeX y Colab:  
[https://colab.research.google.com/github/bebi103a/bebi103a.github.io/blob/master/lessons/00/intro\\_to\\_latex.ipynb](https://colab.research.google.com/github/bebi103a/bebi103a.github.io/blob/master/lessons/00/intro_to_latex.ipynb)
- Markdown y Colab:  
[https://colab.research.google.com/notebooks/markdown\\_guide.ipynb](https://colab.research.google.com/notebooks/markdown_guide.ipynb)

# Primeros pasos con Python

# Primeros pasos (1)

- Abrir un Notebook en Colab (o la guía que os proporcionamos)
- En una celda de código (Colab) o en la línea de comandos/terminal (Spyder) escribir código

```
In [1]: 2*5+10**2  
Out[1]: 110
```

- Se puede realizar asignaciones, importar paquetes, etc.

- **Indentación** obligatoria

**Nota:** muchos de los ejemplos visuales han sido ejecutados en el IDE Spyder. El comportamiento en Colab es el mismo, solo que, en lugar de mostrarse el resultado en el terminal, sale en una celda del cuaderno.

Importar un módulo usando un alias

```
In [2]: import numpy as np
```

```
In [3]: z = np.zeros((5,2), np.float32)
```

```
In [4]: z.shape  
Out[4]: (5, 2)
```

```
In [5]: z.sum()  
Out[5]: 0.0
```

```
In [6]: z  
Out[6]:  
array([[0., 0.],  
       [0., 0.],  
       [0., 0.],  
       [0., 0.],  
       [0., 0.]], dtype=float32)
```

# Primeros pasos (2)

```
#Variables  
entero1 = 5 #Int  
entero2 = 505 #Int  
flotante = 50.5 #Float  
boolean_t = True #Boolean  
boolean_f = False #Boolean  
string1 = 'String1' #String  
string2 = 'String2' #String
```

```
#Operaciones aritméticas  
suma = entero1 + flotante  
resta = entero1 - flotante  
producto = entero1 * flotante  
print(producto)  
division = entero1 / flotante  
print(division)  
division_entera = entero2 // entero1  
print(division_entera)  
resto = entero2 % entero1  
print(resto)
```

Sin el `print()` no ves el resultado de la operación

```
In [2]: suma = entero1 + flotante  
...: resta = entero1 - flotante  
...: producto = entero1 * flotante  
...: print(producto)  
252.5
```

```
In [3]: division = entero1 / flotante  
...: print(division)  
0.09900990099009901
```

```
In [4]: division_entera = entero2 // entero1  
...: print(division_entera)  
101
```

```
In [5]: resto = entero2 % entero1  
...: print(resto)  
0
```



# Primeros pasos (3)

```
#Operaciones lógicas
igual = entero2==(500 + 5)
no_igual = entero1 != suma
mayor = entero2 > entero1 #>=
menor = entero1 < entero2 # <=
and_logico = igual and mayor
or_logico = igual or no_igual

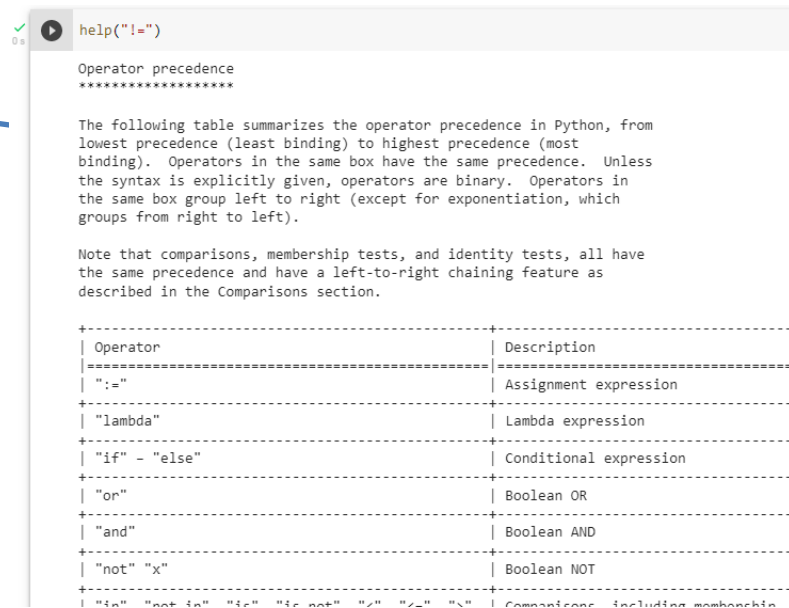
#Cambiar tipos
entero2flotante = float(entero1)
flotante2entero = int(flotante)
astring = str(entero2)
abool = bool(entero1)

#Strings
formatear = 'String con entero %d, flotante %f y string %s' % (entero1,
                                                             flotante,
                                                             string1)

concatenar = string1 + str(entero1)

#Mostrar por pantalla
print('Dos de los strings:', string1, string2)
print('String y entero:', concatenar, entero1)
```

## Ayuda sobre un comando concreto



```
help("!=")
```

Operator precedence  
\*\*\*\*\*

The following table summarizes the operator precedence in Python, from lowest precedence (least binding) to highest precedence (most binding). Operators in the same box have the same precedence. Unless the syntax is explicitly given, operators are binary. Operators in the same box group left to right (except for exponentiation, which groups from right to left).

Note that comparisons, membership tests, and identity tests, all have the same precedence and have a left-to-right chaining feature as described in the Comparisons section.

Operator	Description
":="	Assignment expression
"lambda"	Lambda expression
"if" - "else"	Conditional expression
"or"	Boolean OR
"and"	Boolean AND
"not" "x"	Boolean NOT
"in" "not in" "is" "is not" "<" "<=" ">" ">="	Comparisons, including membership

```
!pip install nbdev
from nbdev.showdoc import *
```

```
help("nombre_función")
print(nombre_función.__doc__)
??"nombre_función"
doc("nombre_función")
```

# Primeros pasos (y 4)

`None`

# Representa la ausencia de algo

`x = None`

# Las variables pueden ser None

`array = [1, 2, None]`

# Las listas pueden contener None

`def func():`

`return None`

# Las funciones pueden devolver None

- No se pueden usar como nombres de variables las siguientes palabras reservadas de Python:

```
from keyword import iskeyword, kwlist
```

```
kwlist
```

Out: ['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

```
iskeyword('try')
```

Out: True

# (Algunos) Tipos de Datos

Tipo	Clase	Notas	Ejemplo
int	Entero	Números enteros con un rango ilimitado	53
float	Real	Coma flotante de doble precisión	3.141592653589793
complex	Complejo	Parte real e imaginaria	(3 + 5j)
bool	Booleano	Valores verdadero o falso	True o False
str	Cadena	Inmutable, contiene texto	"Hola"
list	Secuencia	Mutable, contiene objetos de diverso tipo	[4, "Hola", 3.14]
tuple	Secuencia	Inmutable, contiene objetos de diverso tipo	(4, "Hola", 3.14)
set	Conjunto	Mutable, sin orden, sin duplicados	set([4, "Hola", 3.14])
frozenset	Conjunto	Inmutable, sin orden, sin duplicados	frozenset([4, "Hola", 3.14])
dict	Diccionario	Pares de clave:valor	{"clave1":4, "clave2":"Hola"}

# Más sobre tipos de datos (1)

- Todos los tipos de datos son objetos en python: id, type, valor
  - **type**(dato), devuelve el tipo
  - **id**(dato), devuelve su identificador (un entero único)
- Se pueden forzar los tipos para obtener el resultado deseado:
  - **bool**(dato), **int**(dato), **float**(dato), **str**(dato),...

Python es un lenguaje:

- **Fuertemente tipado:**  $1 + '1' \rightarrow \text{Error!}$
- **Dinámicamente tipado:**  $\text{foo} = [1,2,3] \dots \rightarrow \dots \text{foo} = \text{'hello!'}$

<https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language>

# Más sobre tipos de datos (y 2)

- Tipado fuerte:
  - Los objetos/valores obedecen reglas estrictas sobre cómo pueden interactuar.
  - Ejemplo de tipado débil (PHP):  
`$x = 1 + "1"; // x es 2`
- Tipado dinámico:
  - Son los datos quienes tienen tipo, y no las variables. Es decir, el tipo de dato que un objeto puede almacenar es mutable.
  - Ejemplo de tipado estático (C#):  
`int i = 5;  
i = "5"; // Error! i solo puede contener enteros!`

# Operaciones y Precedencia

Operación	Símbolo	Ejemplo	Resultado	Precedencia
Suma	+	$7 + 2$	9	4
Resta	-	$7 - 2$	5	4
Multiplicación	*	$7 * 2$	14	2
División Real	/	$7 / 2$	3.5	2
División Entera	//	$7 // 2$	3	2
Módulo	%	$7 \% 2$	1	3
Potenciación	**	$7 ** 2$	49	1

# Más sobre operaciones

Al realizar ciertas operaciones, los resultados pueden presentar errores de redondeo:

```
>>> 100 / 3  
33.333333333333336
```

Debido a estos errores, dos operaciones que debieran dar el mismo resultado pueden dar resultados diferentes:

```
>>> 4 * 3 / 5  
2.4  
>>> 4 / 5 * 3  
2.4000000000000004
```

Y viceversa:

```
>>> round(3.45, 1)  
3.5  
>>> round(3.55, 1)  
3.5
```

Este error se debe a que **Python almacena los números decimales en binario** (con 53 bits de precisión, norma IEEE-754), **y pasar de decimal a binario provoca errores de redondeo.**

Cuando se pide un cálculo con números decimales, Python convierte esos números decimales a binario, realiza la operación en binario y convierte el resultado de nuevo a decimal para mostrárselo al usuario.

En la mayor parte de los casos, esto no es especialmente problemático. No obstante, si se necesita total exactitud, se pueden utilizar bibliotecas específicas: **decimal**, **fractions** o **mpmath**

# Listas, tuplas y diccionarios (1)

```
In [1]: tupla = (5, 't1', True, 0.5)
...: print(tupla)
(5, 't1', True, 0.5)
```

```
In [2]: lista = [5, 't1', True, 0.5]
...: print(lista)
[5, 't1', True, 0.5]
```

```
In [3]: l_tupla = len(tupla)
...: l_lista = len(lista)
...: print(l_tupla)
...: print(l_lista)
```

4  
4

```
In [4]: print(tupla[2])
...: print(lista[2])
```

True  
True

```
In [5]: lista[2] = 1000
...: print(lista)
[5, 't1', 1000, 0.5]
```

**Tuplas:** Almacenan datos de cualquier tipo y se acceden mediante índices enteros.

- **No pueden modificarse, solo consultarse (son estáticas).**

**Nota importante:**

Python indexa de 0 a N-1

```
In [6]: print(tupla[4])
Traceback (most recent call last):
```

```
File "<ipython-input-6-09eb472720a7>", line 1, in <module>
    print(tupla[4])
```

**IndexError:** tuple index out of range



# Listas, tuplas y diccionarios (2)

**Listas:** Almacenan datos de cualquier tipo y se acceden mediante índices enteros.

- **Son dinámicas, es decir, pueden modificarse sus elementos, añadir, eliminar,...**

```
#Añadir elemento
lista.append(False) #Al final
position = 1
lista.insert(position, 't21') #En una posición concreta
```

```
#Eliminar elemento
lista.remove('t1') #Buscando por el propio elemento
lista.pop() #Al final
lista.pop(1) #En la posición 1
```

```
#Concatenar
lista2 = ['a', 'b', 'c']
lista_combinada = lista + lista2 #Pone lista2 al final
                                # de lista
```

```
#Copiar
lista_copia = lista.copy()
```

```
In [3]: lista.append(False)
...: print(lista)
[5, 't1', 1000, 0.5, False]
```

```
In [4]: position = 1
...: lista.insert(position, 't21')
...: print(lista)
[5, 't21', 't1', 1000, 0.5, False]
```

```
In [5]: lista.remove('t1')
...: print(lista)
[5, 't21', 1000, 0.5, False]
```

```
In [6]: lista.pop()
...: print(lista)
[5, 't21', 1000, 0.5]
```

```
In [7]: lista.pop(1)
...: print(lista)
[5, 1000, 0.5]
```

```
In [8]: lista2 = ['a', 'b', 'c']
...: lista_combinada = lista + lista2
...: print(lista_combinada)
[5, 1000, 0.5, 'a', 'b', 'c']
```

```
In [9]: lista_copia = lista.copy()
```

```
In [10]: print(lista_copia)
[5, 1000, 0.5]
```

# Listas, tuplas y diccionarios (y 3)

**Diccionarios:** Almacenan datos (*values*) de cualquier tipo y se acceden mediante palabras clave (*keys*).

- Pueden modificarse sus elementos, añadir, eliminar, ...

```
#Declarar  
diccionario = {'a': 1, 'b': 2.0}
```

```
#Añadir elemento  
diccionario['c'] = False
```

```
#Mostrar por pantalla  
print(diccionario)
```

```
#Eliminar elemento  
del diccionario['c']  
print(diccionario)
```

```
#Keys  
diccionario.keys()
```

```
#Values  
diccionario.values()
```

```
In [29]: diccionario = {'a': 1, 'b': 2.0}  
....:
```

```
.... #Añadir elemento  
.... diccionario['c'] = False  
....:
```

```
.... #Mostrar por pantalla  
.... print(diccionario)
```

```
{'a': 1, 'b': 2.0, 'c': False}
```

```
In [30]: del diccionario['c']  
.... print(diccionario)
```

```
{'a': 1, 'b': 2.0}
```

```
In [31]: print(diccionario.keys())  
dict_keys(['a', 'b'])
```

```
In [32]: print(diccionario.values())  
dict_values([1, 2.0])
```

# Indexado (1)

- `lista[inicio:fin:passo]`

- Toda la lista: `lista[:]`
- Desde inicio: `lista[inicio:]`
- Hasta fin: `lista[:fin]` ← Sin incluir fin!!!
- Solo pares: `lista[::2]`

```
In [33]: lista = [5, 't1', True, 0.5]
```

```
In [34]: lista[:]
Out[34]: [5, 't1', True, 0.5]
```

```
In [35]: lista[0:2]
Out[35]: [5, 't1']
```

```
In [36]: lista[3:]
Out[36]: [0.5]
```

```
In [37]: lista[::2]
Out[37]: [5, True]
```

- Podemos usar índices negativos, estos comenzarán a contar desde el final de la lista.

```
In [38]: lista[-1]
Out[38]: 0.5
```

- La posición -1 es la del último elemento de la lista.



Posiciones	0	1	2	3	4	5	6	7	8	→
Posiciones	-9	-8	-7	-6	-5	-4	-3	-2	-1	←

# Indexado (y 2)

```
lista = [0, 1, 2, 3, 4]
```

```
print(lista[0:15:1])
```

```
[0, 1, 2, 3, 4]
```

Da igual si nos pasamos del tamaño máximo real de la lista.

```
print(lista[0:15:2])
```

```
[0, 2, 4]
```

```
print(lista[0:4:2])
```

```
[0, 2]
```

```
print(lista[0:-1:2])
```

```
[0, 2]
```

```
print(lista[::-1])
```

Esto se podría leer como “Devuélveme/recorre la lista al revés”

```
[4, 3, 2, 1, 0]
```

```
print(lista[3:0:-1])
```

```
[3, 2, 1]
```

```
print(lista[3::-1])
```

```
[3, 2, 1, 0]
```

```
print(lista[0:3:-1])
```

```
[]
```

Esto se podría leer como “Vete, hacia atrás, de la posición 0 de la lista a la posición 3-1”. Y, claro,... eso no es posible. De ahí la lista vacía resultante.

# Condicionales y bucles

```
#Condicional
if condicion:
    #Hacer algo
elif otra_condicion:
    #Hacer algo
else:
    #Hacer algo

#Bucle for
for i in range(inicio, fin, paso):
    #Hacer algo
    print(i)

for elemento in lista:
    #Hacer algo

#Bucle while
while condicion:
    #Hacer algo
```

```
In [33]: for x, y in [(1,10), (2,20), (3,30)]:
...:     print (x, y)
1 10
2 20
3 30
```

La indentación es importante!!!

```
nota_obtenida = 6.8
if nota_obtenida < 5:
    print('Suspenso')
elif 5 <= nota_obtenida < 7:
    print('Aprobado')
elif 7 <= nota_obtenida < 8.5:
    print('Notable')
else:
    print('Sobresaliente')
```

print('Suspenso')

IndentationError: expected an indented block

# Comandos útiles (Zip)

- **Zip**: permite iterar sobre varias listas/arrays en paralelo

```
In [1]: list_a = [1,2,3,4]
...: list_b = [2,3,4,5]
...:
...: list_a*list_b
```

Traceback (most recent call last):

```
File "<ipython-input-1-89f73911f1be>", line 4, in <module>
    list_a*list_b
```

TypeError: can't multiply sequence by non-int of type 'list'

In [2]:

```
In [2]: [a*b for a,b in zip(list_a,list_b)]
Out[2]: [2, 6, 12, 20]
```

} list\_a y list\_b no tienen por qué ser del mismo tamaño

Pero el número de listas o argumentos sí debe coincidir:

```
list_a = [*range(1,5)] ## es el argument-unpacking operator, que permite desempaquetar el rango.
list_b = [*range(2,6)]
list_c = [10,10,10]
```

```
[a*b for a,b,c in zip(list_a, list_b)]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-3-60a17bee5eb5> in <module>
----> 1 [a*b for a,b,c in zip(list_a, list_b)]
```

```
<ipython-input-3-60a17bee5eb5> in <listcomp>(.0)
----> 1 [a*b for a,b,c in zip(list_a, list_b)]
```

ValueError: not enough values to unpack (expected 3, got 2)

BUSCAR EN STACK OVERFLOW

```
[a*b for a,b,c in zip(list_a, list_b, list_c)]
```

```
[2, 6, 12]
```

# Comandos útiles (Enumerate)

- **Enumerate:**

- permite iterar sobre índices y elementos de una lista

```
In [1]: alist = ['a1', 'a2', 'a3']
...: b1ist = ['b1', 'b2', 'b3']
...:
...: for i, (a, b) in enumerate(zip(alist, b1ist)):
...:     print(i, a, b)
0 a1 b1
1 a2 b2
2 a3 b3
```

```
l1 = ["eat", "sleep", "repeat"]
s1 = "geek"

# creating enumerate objects
obj1 = enumerate(l1)
obj2 = enumerate(s1)

print ("Return type:", type(obj1))
print (list(enumerate(l1)))

# changing start index to 2 from 0
print (list(enumerate(s1, 2)))
```

```
Return type: <class 'enumerate'>
[(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
[(2, 'g'), (3, 'e'), (4, 'e'), (5, 'k')]
```

# Funciones (1)

- Se pueden crear funciones nuevas o emplear aquellas disponibles en los módulos.
- Tres formas principales de importar una librería o módulo:

```
import module as md  
md.fun()
```

```
from module import *  
fun()
```

```
from module import fun as f1  
f1()
```

- Para conocer todas las funciones dentro del módulo importado disponemos de **dir**(module).



# Funciones (2)

- Atención a la hora de importar módulos



```
In [1]: from sklearn import datasets  
  
In [2]: iris = datasets.load_iris()  
...: X = iris.data  
...: y = iris.target
```



```
In [1]: import sklearn as skl
```

```
In [2]: iris = skl.datasets.load_iris()  
Traceback (most recent call last):
```

```
File "<ipython-input-2-2392f01db2c1>", line 1, in <module>  
    iris = skl.datasets.load_iris()
```

```
AttributeError: module 'sklearn' has no attribute 'datasets'
```

```
import sklearn as skl  
dir(skl)
```

```
['_SKLEARN_SETUP__', 'base',  
'__all__', 'clone',  
'__builtins__', 'config_context',  
'__cached__', 'exceptions',  
'__check_build__', 'externals',  
'__doc__', 'get_config',  
'__file__', 'logger',  
'__loader__', 'logging',  
'__name__', 'os',  
'__package__', 'random',  
'__path__', 'set_config',  
'__spec__', 'setup_module',  
'__version__', 'show_versions',  
'__config__', 'sys',  
'__distributor_init__', 'utils']
```

## ¿Por qué?

**datasets** es un sub-paquete de **sklearn**

Quando importas un paquete solamente las variables/funciones/clases en el `__init__.py` de ese paquete son directamente visibles, no los sub-paquetes o módulos.

# Funciones (3)

```
def funcion(a, b=1):  
    c = a + b  
    return c
```

Valor por defecto para el parámetro

```
c = funcion(1,2) #O funcion(a=1, b=2)  
print(c)  
c_def = funcion(1)  
print(c_def)
```

Sin **return**  
el **print** posterior  
imprimiría **None**

**return** fuerza la salida de la función  
y devuelve un valor al **caller**

---

```
In [1]: a = [5,3,2,6,1]
```

```
In [2]: a  
Out[2]: [5, 3, 2, 6, 1]
```

```
In [3]: a.sort()
```

```
In [4]: a  
Out[4]: [1, 2, 3, 5, 6]
```

Al omitir el segundo parámetro (**b**),  
**b** tiene el valor por defecto (1)

Hay métodos que son **in-place**: no devuelven nada, modifican el objeto directamente, y no necesitan asignaciones

# Funciones (4)

Cuidado a la hora de realizar asignaciones!

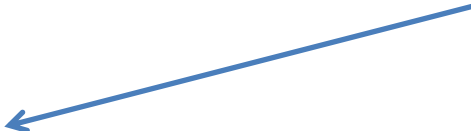
```
In [47]: some_guy = 'Fred'

In [48]: first_names = []
...: first_names.append(some_guy)

In [49]: print(first_names)
['Fred']

In [50]: another_list_of_names = first_names
...: another_list_of_names.append('George')
...: some_guy = 'Bill'

In [51]:
...: print (some_guy, first_names, another_list_of_names)
...:
Bill ['Fred', 'George'] ['Fred', 'George']
```



¿Por qué **first\_names**  
no contiene solamente ['Fred']?

Ambas variables  
(**another\_list\_of\_names** y  
**first\_names**) están ligadas al  
mismo objeto.

El string object (**some\_guy**) y el  
list object (**first\_names**) siguen  
siendo los únicos objetos creados  
por el intérprete de Python.

```
id(first_names)
1758256788808

id(another_list_of_names)
1758256788808
```

Ejemplos extraídos de "Is Python call-by-value or call-by-reference? Binding Names to Objects" de

Jeff Knupp. <https://www.geek-share.com/detail/2606760623.html>

# Funciones (5)

```
In [54]: first_names = ['Fred', 'George', 'Bill']  
...: last_names = ['Smith', 'Jones', 'Williams']  
...: name_tuple = (first_names, last_names)  
  
In [55]: print(name_tuple)  
(['Fred', 'George', 'Bill'], ['Smith', 'Jones', 'Williams'])  
  
In [56]: first_names.append('Igor')  
  
In [57]: print(first_names)  
['Fred', 'George', 'Bill', 'Igor']  
  
In [58]: print(name_tuple)  
(['Fred', 'George', 'Bill', 'Igor'], ['Smith', 'Jones', 'Williams'])
```

Un objeto “inmutable” (como una tupla) puede “mutar” si contiene objetos “mutables” (como listas)

# Funciones (6)

```
In [60]: def foo(bar):  
...:     bar.append(42)  
...:     print(bar)
```

```
In [61]: answer_list = []  
...:     foo(answer_list)  
[42]
```

```
In [62]: print(answer_list)  
[42]
```

Si **bar** se refiere a un **objeto mutable** (como una lista), y **foo** cambia su **valor**, entonces estos cambios serán visibles fuera de la función.

Se asemeja al comportamiento de paso por referencia

```
In [63]: def foo(bar):  
...:     bar = 'new value'  
...:     print (bar)  
...:     # >> 'new value'  
...:  
...:  
...:     answer_list = 'old value'  
...:     foo(answer_list)  
new value
```

```
In [64]: print(answer_list)  
old value
```

Si **bar** se refiere a un **objeto inmutable** (como un string), lo máximo que puede hacer **foo** es crear una variable interna **bar** y ligarla a algún otro objeto.

Se asemeja al comportamiento de paso por valor

Este comportamiento “dual” se llama **“Call by Object Reference”** o **“Call by assignment”** y es MUY IMPORTANTE tenerlo claro para evitar *bugs*.

# Funciones (7)

Si no queremos que cambie el valor de una variable, ¿cómo evitarlo? Es decir, ¿cómo evitar estos problemas y/o efectos indeseables?

Si **a** es una lista o array:



**b = a**



**b = a.copy()**

**Nota:** véase diapositiva 19 del segundo bloque de esta presentación.

# Funciones (y 8)

Un último ejemplo:

```
In [1]: def spam(eggs):  
...:     eggs.append(1)  
...:     eggs = [2, 3]  
...:     print(eggs)
```

```
In [2]: ham = [0]  
...: spam(ham)  
[2, 3]
```

```
In [3]: print(ham)  
[0, 1]
```

Cuando se llama a **spam**,  
**eggs** y **ham** apuntan al mismo valor ([0])

Cuando hacemos **eggs.append(1)**  
→ [0] se convierte en [0, 1]

Cuando hacemos **eggs = [2, 3]**  
→ ahora **eggs apunta a una nueva lista en memoria** que contiene [2, 3]  
Pero **ham** apunta todavía a la lista [0, 1]

# Lambda *functions*

- Permiten emplear funciones de una forma más concisa

```
raiz_cuadrada = lambda x: x**(1/2)  
raiz_cuadrada(4)
```

2.0



```
def raiz_cuadrada2(x):  
    return x**(1/2)  
raiz_cuadrada2(4)
```

2.0

```
suma = lambda x, y: x+y  
suma(2,3)
```

5

```
import numpy as np  
funcion_compuesta = lambda x, func: x + np.sqrt(x) + func(x)  
funcion_compuesta(4, lambda x: x**2)
```

22.0

<https://docs.python.org/3/reference/expressions.html#lambda>

<https://docs.python.org/3/howto/sorting.html>



# Clases (1)

```
class Clase():
```

```
    def __init__(self, a):  
        self.a = a
```

```
    def llamar(self, b):  
        return self.a*b
```

`__init__()` no es un constructor (`__new__()`): es llamado inmediatamente después de que el objeto haya sido creado y se usa para inicializarlo

```
class Clase2(Clase):
```

```
    def __init__(self, a, b=2.0):  
        super().__init__(a)  
        self.b=b
```

```
    def llamar(self, c):  
        return self.a*self.b*c
```

```
    def __call__(self, c):  
        return self.llamar(c)
```

Clase2 hereda los métodos de Clase

Sobrescribe el método `llamar()`

```
In [96]: c = 3
```

```
In [97]: clase2 = Clase2(a=1)
```

```
In [98]: d = clase2.llamar(c)
```

```
In [99]: print(d)  
6.0
```

`self` no es una palabra reservada: podríais usar `this` o `myself`, por ejemplo, aunque no se recomienda por ser una convención fuertemente aceptada.

# Clases (2)

```
In [1]: class Point(object):
...:     def __init__(self, x = 0, y = 0):
...:         self.x = x
...:         self.y = y
...:
...:     def distance(self):
...:         """Find distance from origin"""
...:         return (self.x**2 + self.y**2) ** 0.5
```

```
In [2]: p1 = Point(6,8)
```

```
In [3]: p1.distance()
Out[3]: 10.0
```

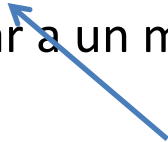
```
In [4]: Point.distance(p1)
Out[4]: 10.0
```

```
In [5]: type(Point.distance)
Out[5]: function
```

```
In [6]: type(p1.distance)
Out[6]: method
```

Si p1 es una instancia de P (i.e. un objeto):

p1.meth(arg) y P.meth(p1, arg) son formas equivalentes de llamar a un método



El **self** se refiere a p1

# Clases (y 3)

In [1]: `class Point(object):`  Si no ponemos `__init__()`, ¿Python pone uno por defecto?

```
...:     def __init__(self, x = 0, y = 0):
...:         self.x =
...:         self.y = y
...:
...:     def distance(self):
...:         """Find distance from origin"""
...:         return (self.x**2 + self.y**2) ** 0.5
```

In [2]: `p1 = Point(6,8)`

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-3-d4ca0747e98e> in <cell line: 6>()
      4         return (self.x**2 + self.y**2) ** 0.5
      5
----> 6 p1 = Point(6,8)
      7
      8 print(p1.distance())
```

`TypeError: Point() takes no arguments`

`__init__()` es necesario siempre y cuando quieras almacenar algún tipo de información del estado del objeto. Si creas objetos que no tienen estado, no te hace falta.

`__init__()` se llama automáticamente cuando un objeto de una clase es creado, y **sirve para inicializar los atributos del objeto e indicar sus valores por defecto.**

```
class A(object):
    def f(self):
        print('Hoolaa')

a = A()
a.f()
```

Si se elimina 'self' obtendríamos un error: `"TypeError: A.f() takes 0 positional arguments but 1 was given"`

# Formateo de salida (1)

```
print('Valor real de 1/3', 1/3, sep=' : ')\nprint('Valor real de 1/3 : ' + str(1/3))\nprint('Valor real de 1/3 : {} '.format(1/3))
```

Valor real de 1/3 : 0.3333333333333333

```
print('Resultado 1 - {} | Resultado 2 - {}'.format(1/3, 1/5))
```

Resultado 1 – 0.3333333333333333 | Resultado2 – 0.2

```
print('Resultado 1 - {0:5.3f} | Resultado 2 - {1:5d}'.format(1/3, 25))
```



Resultado 1 – 0.333 | Resultado 2 – 25

```
print('Resultado 1 - {1:18.2f} | Resultado 2 - {0:7.4f}'.format(1/3, 25))
```

Resultado 1 – 25.00 | Resultado 2 – 0.3333

# Formateo de salida (2)

- También se pueden formatear las salidas para mostrar expresiones matemáticas (en celdas de texto y resultados)

```
from IPython.display import display, Latex, Math
a = r'f(x) = \frac{\exp(-x^2/2)}{\sqrt{2\cdot\pi}}'
display(Math(a))
for i in range(3):
    display(Math(f'$x_{i}$'))
```

"r" (*raw string*): muestra literalmente lo que aparece entre comillas.

"f" es más flexible, y permite el uso de variables.

$$f(x) = \frac{\exp(-x^2/2)}{\sqrt{2 \cdot \pi}}$$

$x_0$

$x_1$

$x_2$

Más información sobre las "r-strings" y las "f-strings" de Python:

<https://stackoverflow.com/questions/52360537/i-know-of-f-strings-but-what-are-r-strings-are-there-others>

# Formateo de salida (y 3)

- También se pueden formatear las salidas para mostrar expresiones matemáticas (en celdas de texto y resultados)

```
from IPython.display import display, Latex, Math
a = r'f(x) = \frac{\exp(-x^2/2)}{\sqrt{2\cdot\pi}}'
display(Math(a))
for i in range(3):
    display(Math(f'$x_{i}$'))
```

$$f(x) = \frac{\exp(-x^2/2)}{\sqrt{2 \cdot \pi}}$$

$x_0$

$x_1$

$x_2$

---

Junto con el manejo de las salidas, Python también posee instrucciones, como `input()`, que permiten la entrada de datos a un programa. `input()` se queda esperando hasta que el usuario introduce algo por teclado.

# Depurando código (1)

- Si os confunde la sintaxis, probad directamente
- Comprobad valores (`print()`), tipos (`type()`), tamaños/formas (`shape()`), atributos/métodos (`dir()`) e identificadores (`id()`)

```
In [1]: import numpy as np
```

```
....: def funcion1(x):
....:     x.append(1)
....:     print(np.shape(x))
....:     print(len(x))
....:     print(type(x))
....:     print(dir(x))
....:     x = (2,3)
....:     print(np.shape(x))
....:     print(len(x))
....:     print(type(x))
....:     print(dir(x))
....:
....: y = [0, 1, 2, 3, 4, 5]
....: funcion1(y)
```

```
(7,)
7
<class 'list'>
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'_ge_', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__',
'_iter_', '__le_', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'_reversed_', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
(2,)
2
<class 'tuple'>
['_add_', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'_getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__',
'_le_', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'_setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
```

```
2 ** 5 / 2
16.0
```

```
2 ** (5 / 2)
5.656854249492381
```

# Depurando código (y 2)

Operaciones que pueden ser de utilidad:

- Verificar si dos arrays son *aproximadamente* iguales

`np.allclose(x, y)` #Se puede especificar la tolerancia

- Verificar si un array es próximo a cero (p.ej. el gradiente)

`np.allclose(x, 0)`

- Seleccionar todos los elementos menores o iguales a 0 en un array

`x[x <= 0]`

```
In [1]: import numpy as np
...: x = np.zeros((1,25))
...: z = np.repeat([1.e-05], 25)
...: np.allclose(x,z,atol=1.e-05)
```

Out[1]: True

```
In [2]: np.allclose(x,z,atol=1.e-06)
```

Out[2]: False

```
In [3]: z = np.repeat([1.e-05, 0, 3], 10)
...: z
```

Out[3]:

```
array([1.e-05, 1.e-05, 1.e-05, 1.e-05, 1.e-05, 1.e-05, 1.e-05, 1.e-05,
       1.e-05, 1.e-05, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00,
       0.e+00, 0.e+00, 0.e+00, 0.e+00, 3.e+00, 3.e+00, 3.e+00, 3.e+00,
       3.e+00, 3.e+00, 3.e+00, 3.e+00, 3.e+00, 3.e+00])
```

```
In [4]: z[z<=0] #Get values
```

Out[4]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

```
In [5]: z<=0 #Get boolean values about the condition
```

Out[5]:

```
array([False, False, False, False, False, False, False, False, False,
       False, True, True, True, True, True, True, True, True, True,
       True, True, False, False, False, False, False, False, False,
       False, False, False])
```

```
In [6]: [i for i in range(len(z)) if z[i] <= 0] #Get indexes
```

Out[6]: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]



# Prácticas de Aprendizaje Automático

## Clase 1: Introducción a Python

Pablo Mesejo y Salvador García

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD  
DE GRANADA

