Programación para el análisis de datos

Federico Pousa
fpousa@udesa.edu.ar

Objetivo

Este curso presenta los conceptos y técnicas de programación generales esenciales para una audiencia de análisis de datos sin experiencia previa en programación.

El objetivo es equipar a los estudiantes con las habilidades de programación necesarias para tener éxito en los otros cursos del programa.

Vamos a separar en 2 la materia

1era parte: Python

2da parte: R (Con Daniel Fraiman)

Programa

1. Introducción a la computación. Introducción a la programación en Python. Entorno de ejecución (python, ipython, jupyter notebook). Instaladores de paquetes (pip). Variables, estructuras de control (condicionales, ciclos). Tipos de datos. Estructuras de datos en Python (diccionarios, listas por comprensión). Clases. Funciones. Manejo de errores, excepciones. Manejo de archivos. debugging
2. Bibliotecas numéricas (numpy)
3. Bibliotecas para estructuración de datos (pandas)
4. Bibliotecas de herramientas estadísticas (scipy)
5. Bibliotecas para graficar (matplotlib, seaborn)

Cronograma

2022-03-03	Jueves	Python intro I
2022-03-05	Sábado	Python intro II
2022-03-08	Martes	Vectorial: numpy+scipy
2022-03-10	Jueves	Dataframes: Pandas
2022-03-12	Sábado	Graficación: matplotlib + seaborn + plotly
2022-03-15	Martes	Terminar guías en clase y dudas | Presentar TP, datos y validar hipótesis
2022-03-19	Sábado	TP en clase

Evaluación

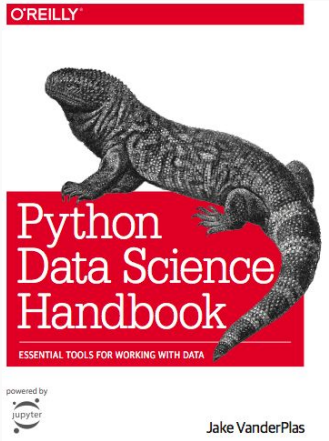
- Trabajo práctico en grupos de 3 alumnos
- Resolución de consignas
- Creación de hipótesis, validación y respuesta
- Informe

Bibliografía

- Infinidad de fuentes en internet

- “Libro”: VanderPlas, Jake. Python data science handbook: Essential tools for working with data. " O'Reilly Media, Inc.", 2016.

  https://jakevdp.github.io/PythonDataScienceHandbook/



Comunicación

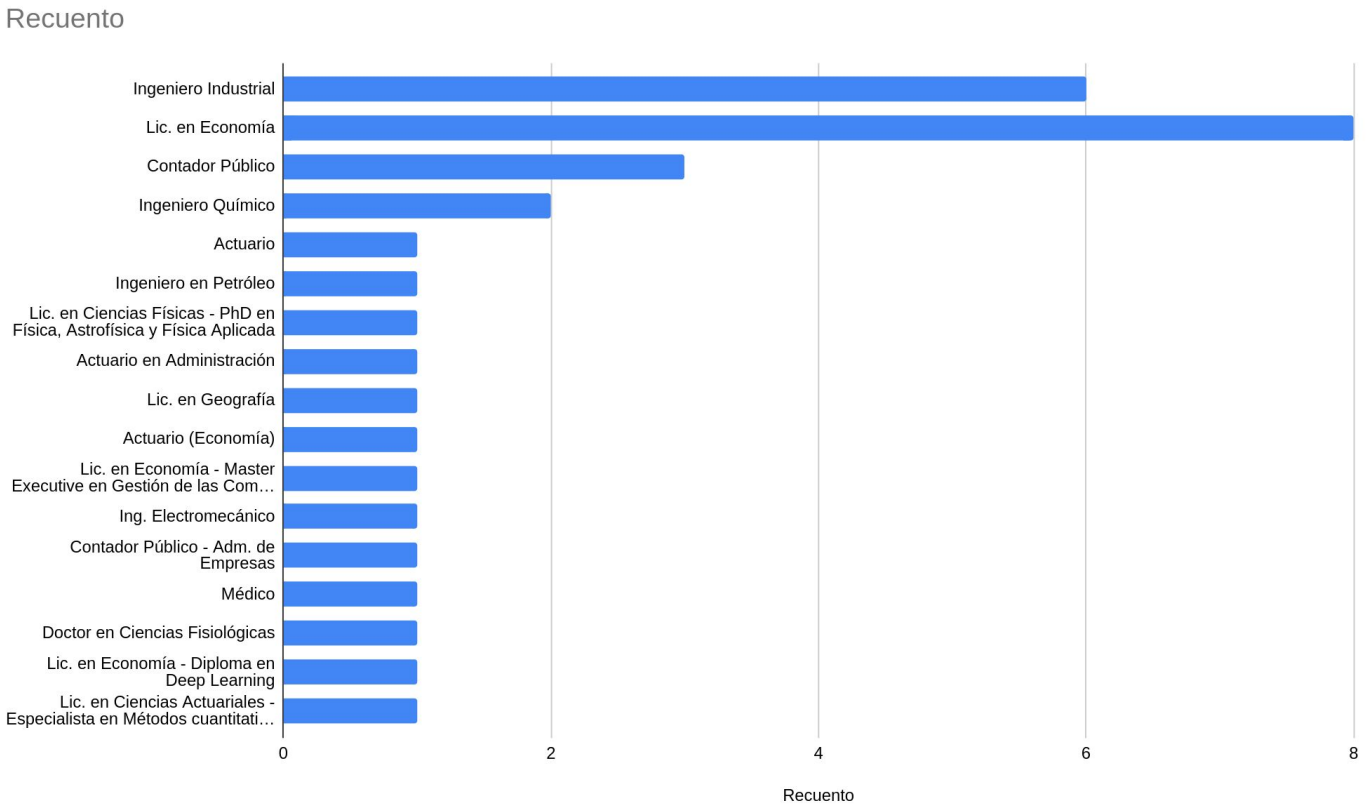
- Email: fpousa@udesa.edu.ar
- Campus: https://campusvirtual.udesa.edu.ar/
- Dictado de clases:
- Clase taller:
  - Servidor de Discord en el campus

I me mine

- Doctor en Ciencias de la Computación (UBA), Área Investigación Operativa.
- Docente en Ciencias de la Computación (UBA), Maestría en Data Science (UDESA), Maestría en Management+Analytics (UTDT).
- Socio y Data Science & Operations Research en Eryx.
- https://www.linkedin.com/in/federico-pousa/

Ustedes

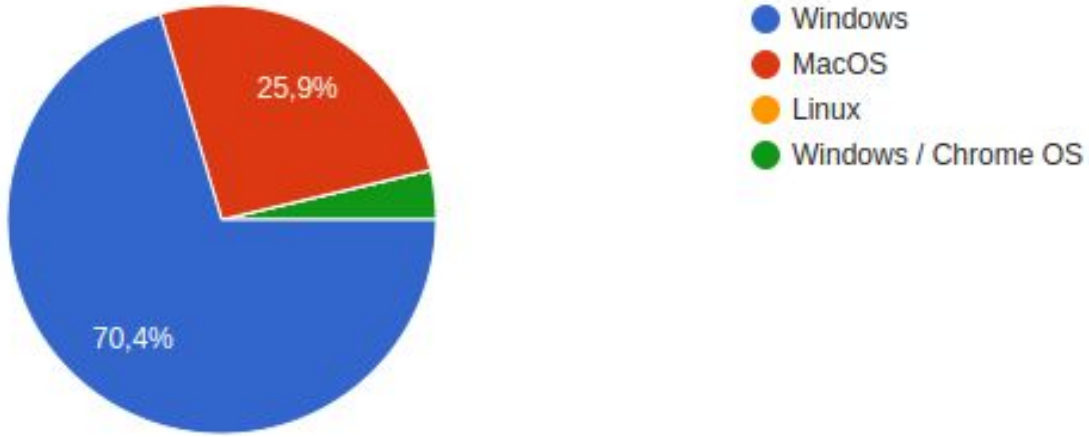
Ustedes


Recuento
Ingeniero Industrial
Lic. en Economía
Contador Público
Ingeniero Químico
Actuario
Ingeniero en Petróleo
Lic. en Ciencias Físicas - PhD en Física, Astrofísica y Física Aplicada
Actuario en Administración
Lic. en Geografía
Actuario (Economía)
Lic. en Economía - Master Executive en Gestión de las Com…
Ing. Electromecánico
Contador Público - Adm. de Empresas
Médico
Doctor en Ciencias Fisiológicas
Lic. en Economía - Diploma en Deep Learning
Lic. en Ciencias Actuariales - Especialista en Métodos cuantitati…
0
2
4
6
8
Recuento

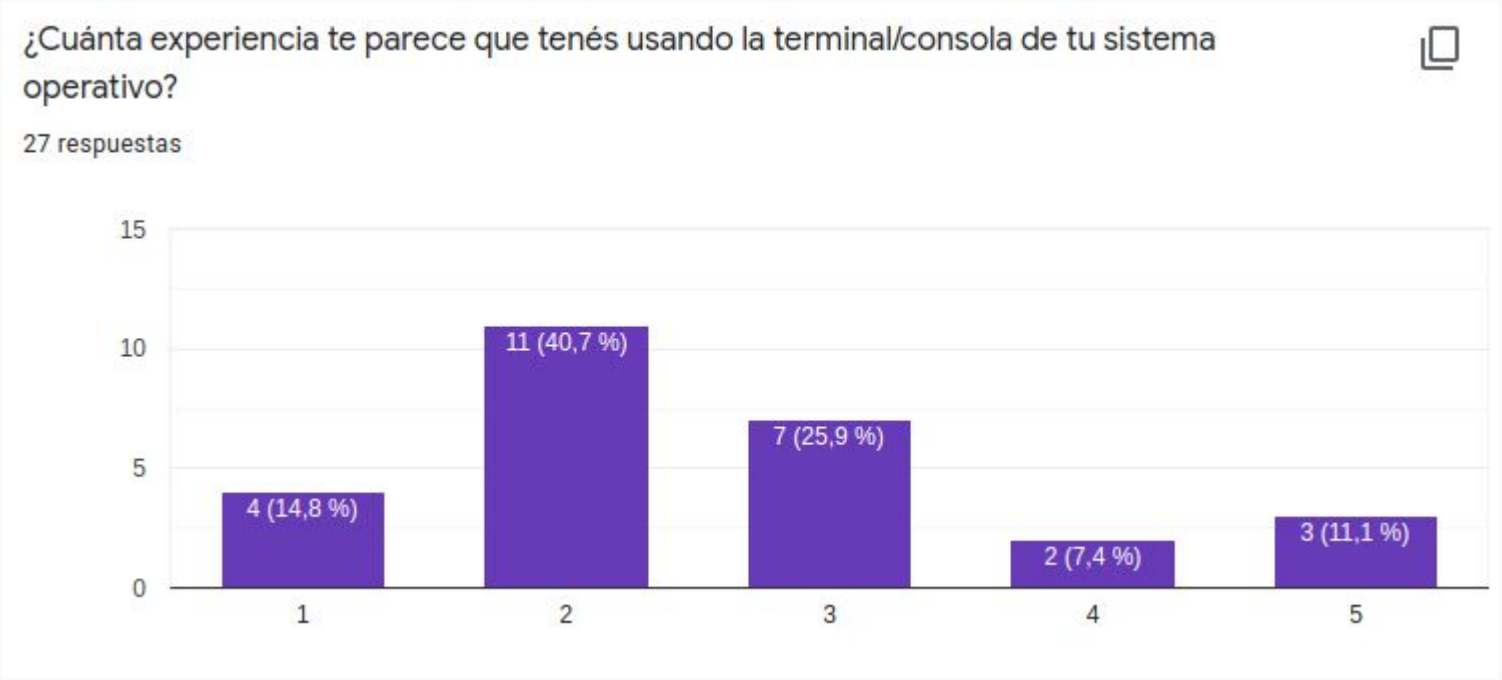
Ustedes

¿Qué sistema operativo usás?

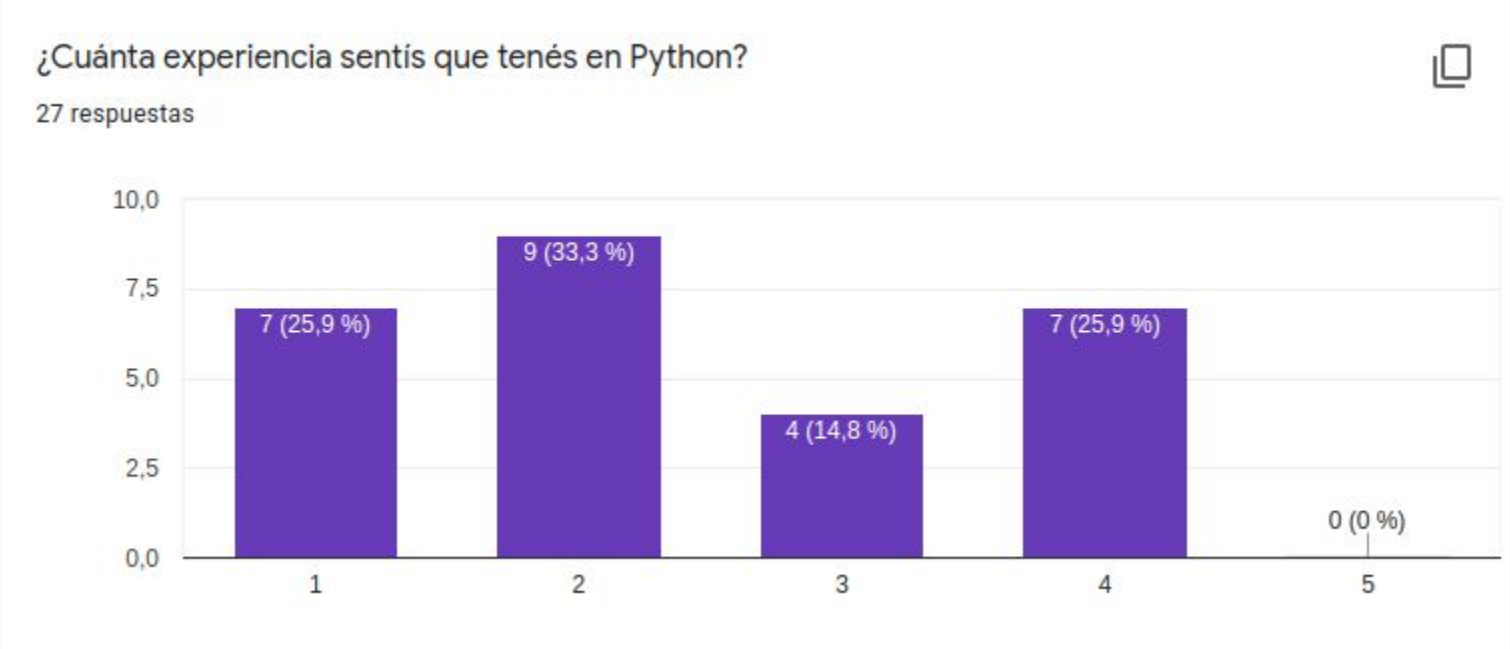
27 respuestas



Ustedes


¿Cuánta experiencia te parece que tenés usando la terminal/consola de tu sistema operativo?
27 respuestas
15
10
5
0
4 (14,8 %)
11 (40,7 %)
7 (25,9 %)
2 (7,4 %)
3 (11,1 %)
1
2
3
4
5

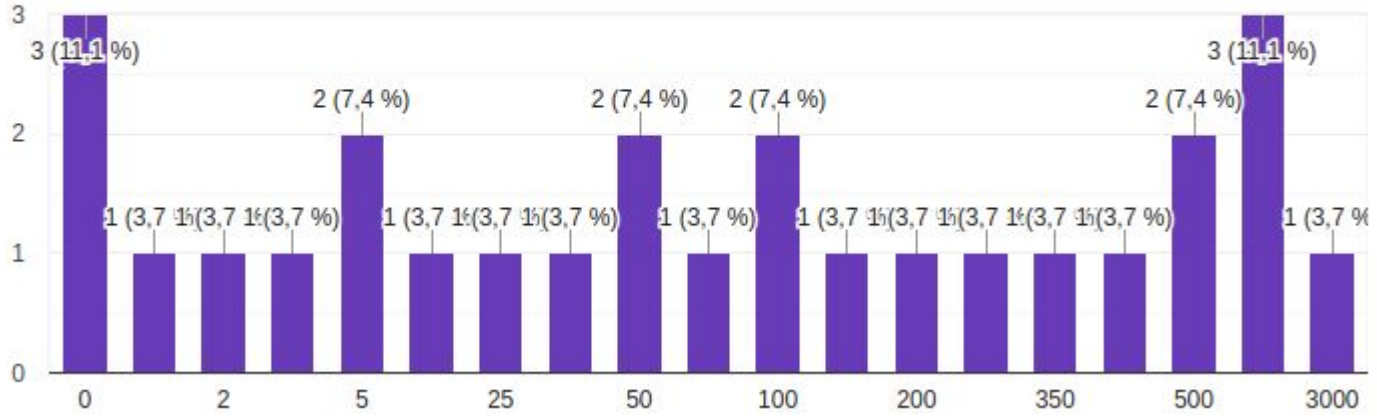
Ustedes


¿Cuánta experiencia sentís que tenés en Python?
27 respuestas
10,0
7,5
5,0
2,5
0,0
7 (25,9 %)
9 (33,3 %)
4 (14,8 %)
7 (25,9 %)
0 (0 %)
1
2
3
4
5

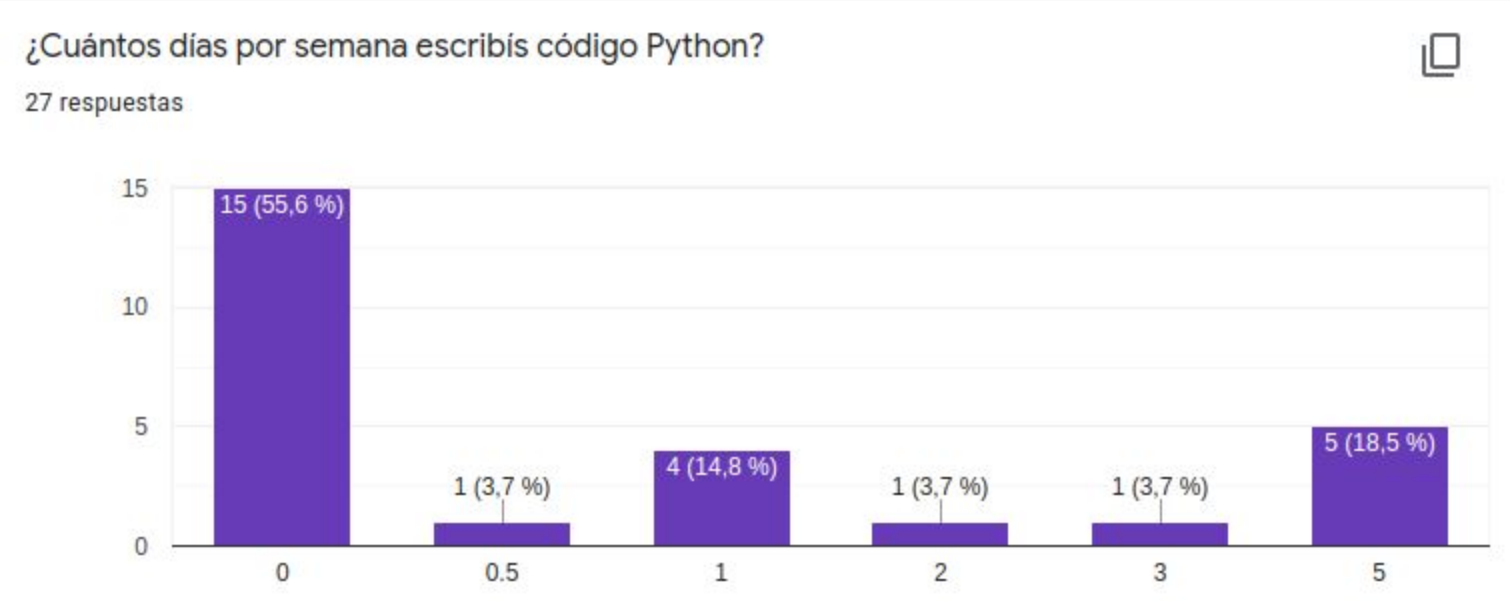
Ustedes

¿Cuántas líneas tiene el código más largo que escribiste?

27 respuestas



Ustedes


¿Cuántos días por semana escribís código Python?
27 respuestas
15
10
5
0
15 (55,6 %)
1 (3,7 %)
4 (14,8 %)
1 (3,7 %)
1 (3,7 %)
5 (18,5 %)
0
0.5
1
2
3
5

Ustedes


basic
html
java
javascript
logo
mathematica
matlab
nada
pascal
sql
stata
vba
visual
wolfram

¿Ciencia de Datos?

¿Ciencias de datos?

Dan Ariely

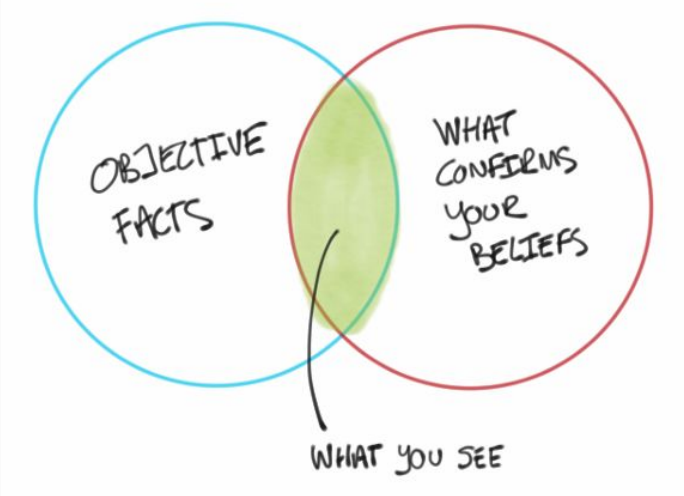
January 6, 2013 at 6:17pm ·

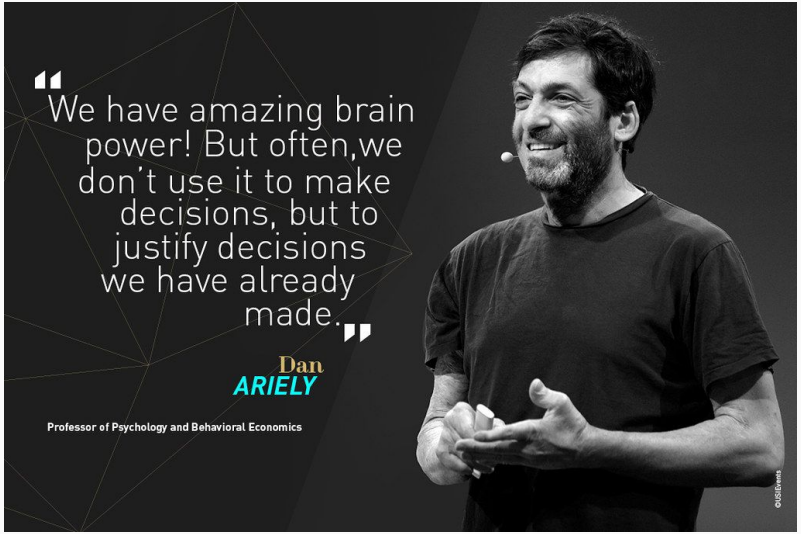
Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it...

Like | Comment | Share

¿Ciencias de datos?

- Poder hacer las preguntas correctas
- Poder abordar estas preguntas con los métodos correctos
- Saber transmitir lo que encontramos





¿Ciencias de datos?

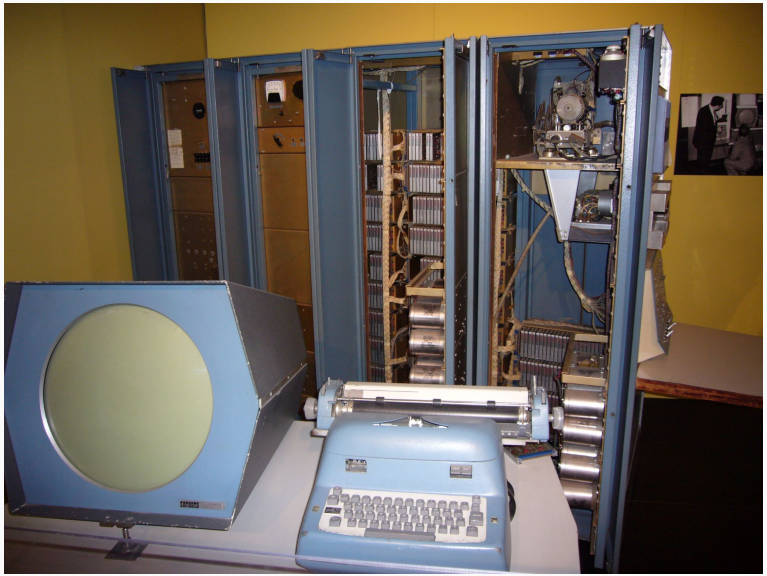
- Obtener datos
- Organizar y limpiar datos
- Interpretar los datos
- Modelar los datos
- Mostrar los resultados

Vamos a tener que saber programar ...

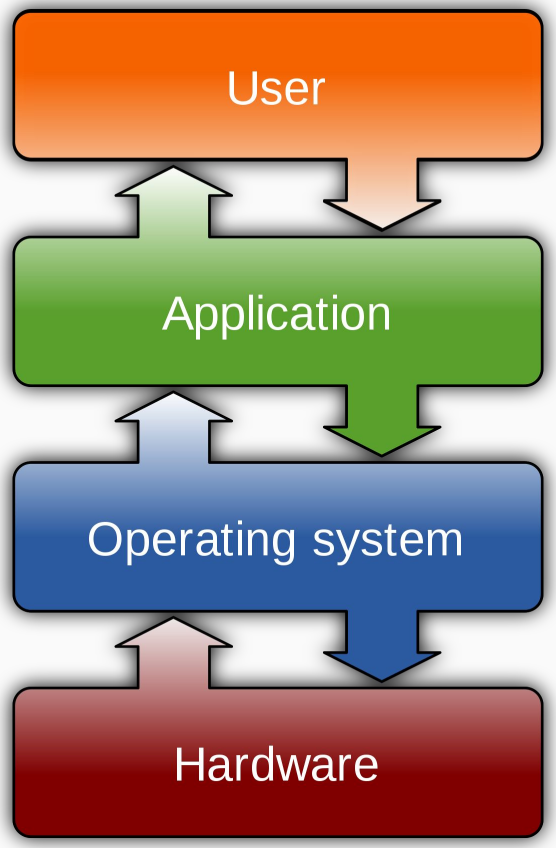
Computación

Computadoras

- Hasta **1945** hubieron máquinas que computaban
- Transición hacia válvulas
- **1955** Primeros transistores
- **1965** Primeros circuitos integrados
- **1980** Primeras computadoras personales



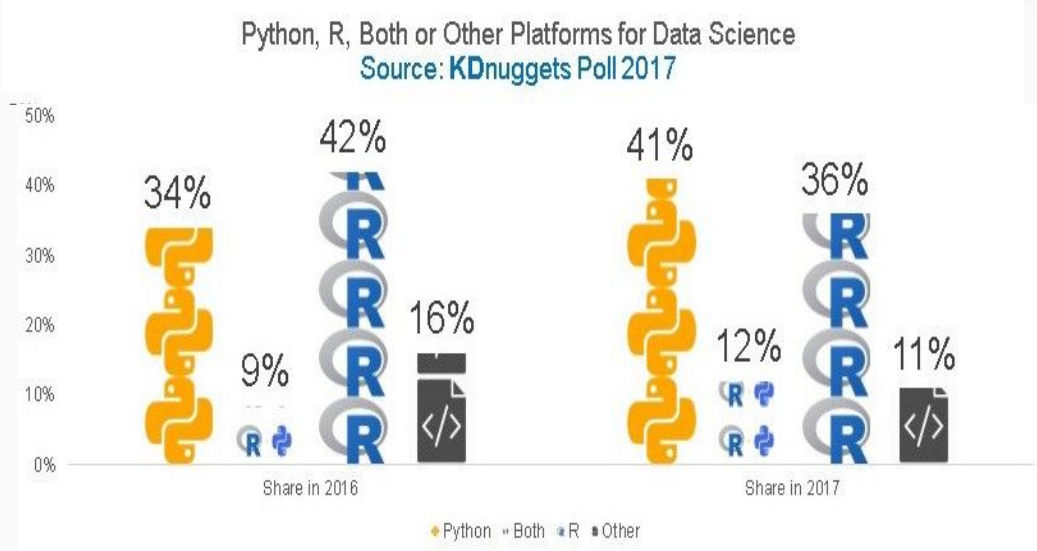
Sistema operativo

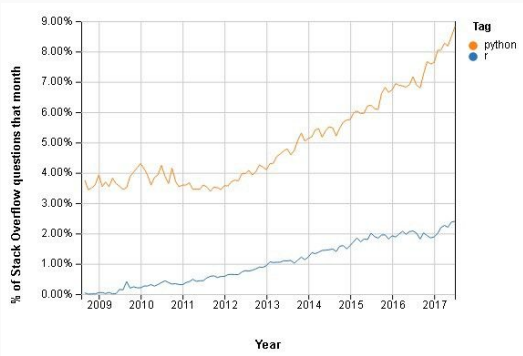


Wikipedia contributors. (2020, January 5). Memory management. In *Wikipedia, The Free Encyclopedia*. Retrieved 14:13, February 10, 2020, from https://en.wikipedia.org/w/index.php?title=Memory_management&oldid=934247639

¿Por qué Python?

- Lenguaje de programación de alto nivel
- Interpretado
- Orientado a objetos
- Tipado dinámico
- Fácil interacción con otros lenguajes
- Muy bien soportado
- Version >= 3
- Multi-plataforma (Linux, macOS, Windows, etc)





Link interesante! https://www.dataschool.io/python-or-r-for-data-science/

Python: cómo, dónde

- ¿Cómo podemos escribir Python?:
  - Consola
  - Script
  - Jupyter notebooks
  - Colab
  - IDEs

- ¿Dónde lo podemos ejecutar?:
  - Local
  - Nube

¿Cómo empezamos?

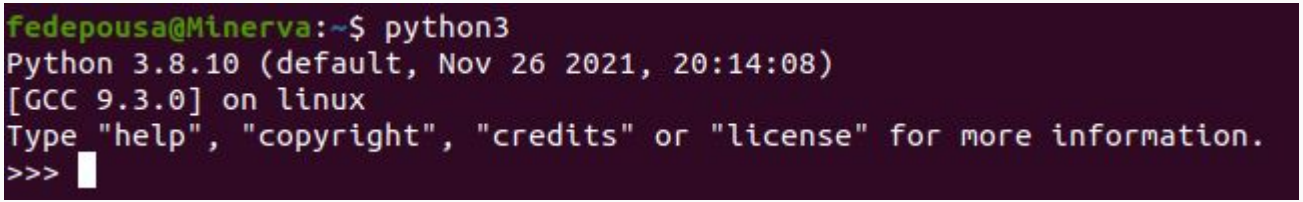
https://en.wikipedia.org/wiki/The_Matrix_(franchise)

Python: Instalación

1. Descargamos Python: https://www.python.org
2. Instalamos Python
3. Esto nos va a permitir poder usar el intérprete de Python

   MacOS/Linux: terminal
   Windows: cmd

4. python3



Python: Instalación pip

¿**Qué** es PIP?

- PIP es un instalador de paquetes de python
- Multi-plataforma

Instalar *cualquier* cosa (desde una consola), por ejemplo:

- pip install numpy
- pip install pandas
- pip install nltk

¿**Cómo** instalamos PIP? (si no lo instalamos previamente)

- Bajamos get-pip.py (https://bootstrap.pypa.io/get-pip.py)
- Lo ejecutamos
- Ya esta :D

Python: Uso, entorno, jupyter

Por ahora, para simplificar vamos a usar **Jupyter**

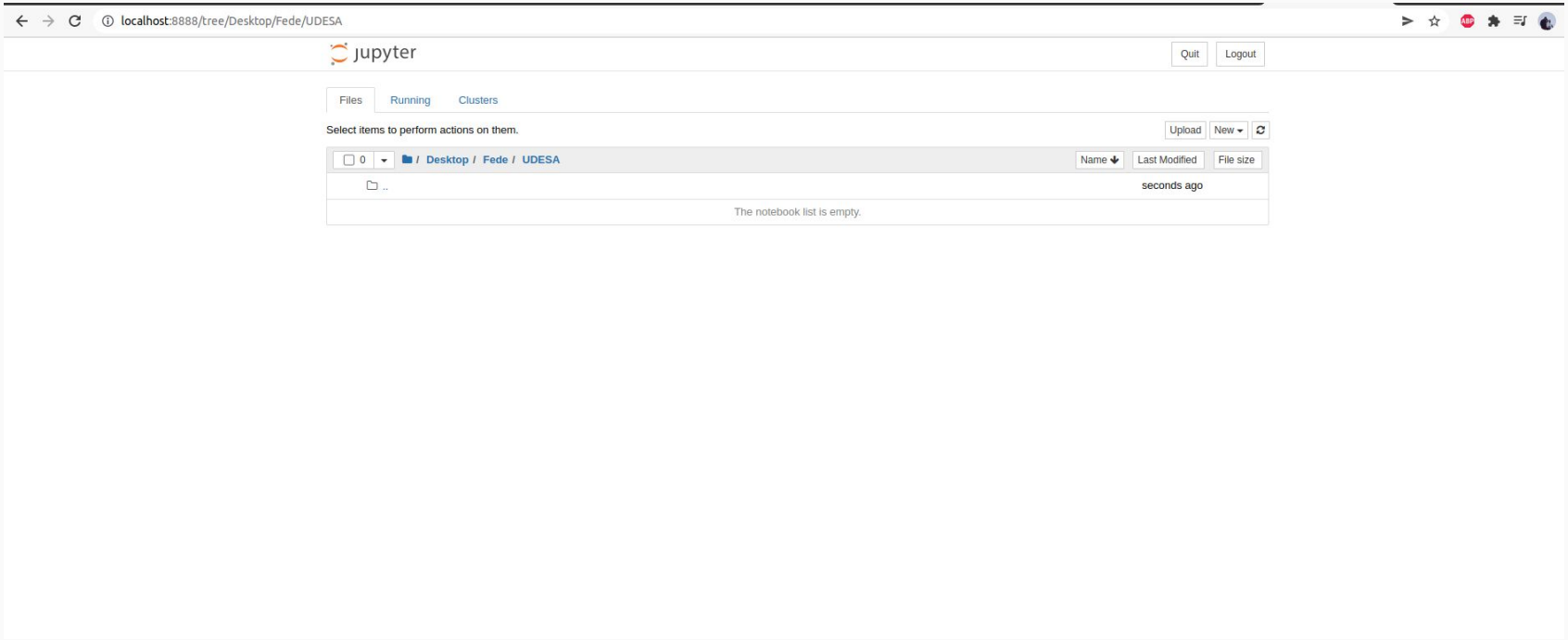
```
pip install jupyter
```

Corremos jupyter desde una consola/terminal

```
jupyter notebook
```

Link referencias: https://jupyter.org/index.html

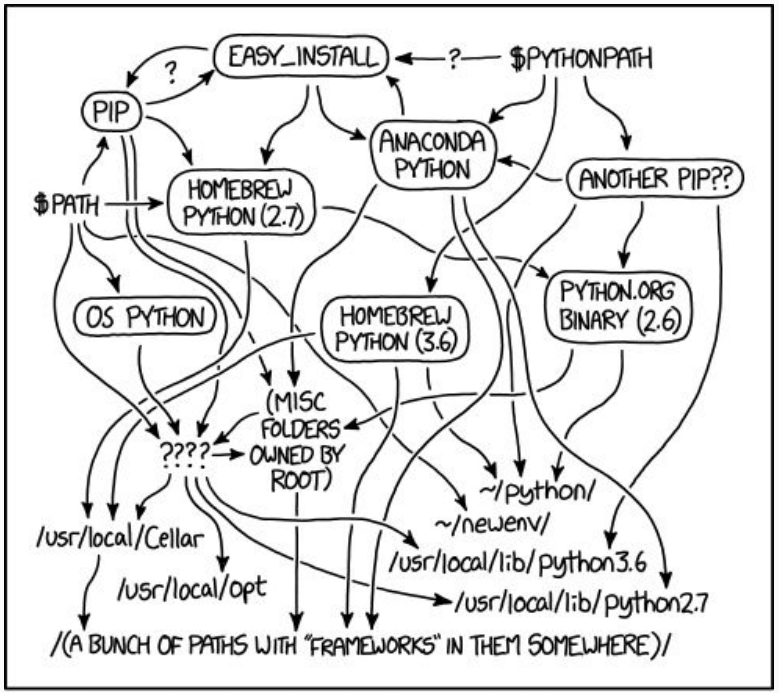
Python: Uso, entorno, jupyter



Ambientes en Python

- Cuando desarrollamos, nos interesa que lo que hacemos sea replicable en otra máquina.
- Además, al tener Python tantas herramientas disponibles, es común instalar muchas librerías que pueden generar incompatibilidades.
- Hay varios “sabores” de ambientes virtuales para subsanar estos problemas.

Una clásica instalación



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

VirtualEnvs

- La idea de un ambiente virtual es que las librerías que instalemos no se instalen a nivel global del sistema operativo, sino solamente dentro de un ambiente virtual controlado.
- Podemos tener muchos ambientes virtuales en una máquina, cada uno con sus librerías y sus respectivas versiones (que pueden ser todas distintas).
- Hay muchas versiones distintas de ambientes virtuales: virtualenv, virtualenvwrapper, conda.

VirtualEnvWrapper

- Un ejemplo de manejador de ambientes virtuales es virtualenvwrapper. Funciona muy parecido a virtualenv pero los ambientes se definen en una carpeta a nivel sistema.
- Se instala corriendo *sudo apt install virtualenvwrapper*.
- Se crea un nuevo ambiente corriendo *mkvirtualenv nombreambiente*.
- Para acceder al ambiente se corre con *workon nombreambiente*.
- Para salir del ambiente se corre *deactivate*.

Conda

- Otra herramienta muy utilizada es Conda. Es tanto un package manager (pip), como un manejador de versiones (virtualenv).
- Con *conda create* se crean ambientes.
- Con *conda activate* se ingresa al ambiente.
- Con *conda deactivate* se sale del ambiente.
- Con *conda install* se instalan librerías.

requirements.txt

- En los ambientes es muy común tener un archivo requirements.txt que indica todas las librerías necesarias en dicho proyecto.
- Con *pip install -r requirements.txt* se pueden instalar todas las librerías necesarias juntas.
- Para armar el archivo de requerimientos se puede correr *pip freeze > requirements.txt*.
- El anterior comando primero lista todas las librerías actualmente instaladas y luego las vuelca a un archivo de texto.

PyEnv

- Además de las herramientas para ambientes virtuales, existe Pyenv que se encarga de que podamos utilizar diferentes versiones de python en una misma computadora.
- Tambien existen pyenv-virtualenv y pyenv-virtualenvwrapper para trabajar conjuntamente.
- https://github.com/pyenv/pyenv
- https://github.com/pyenv/pyenv-virtualenv
- https://github.com/pyenv/pyenv-virtualenvwrapper

Demo de jupyter!

Demo de colab!

Python: Ejercicio 0

1. Instalar python, pip, jupyter
2. Correr jupyter
3. Crear un archivo primera_prueba.py (usar Text File)
4. Escribir un código que imprima en pantalla “Hola Mundo!”
5. Ejecutar el programa en una consola de Python3
6. Modificar el programa anterior para que luego de “Hola Mundo!” imprima en otra línea “Chau Mundo!”
7. Celebre el progreso :D

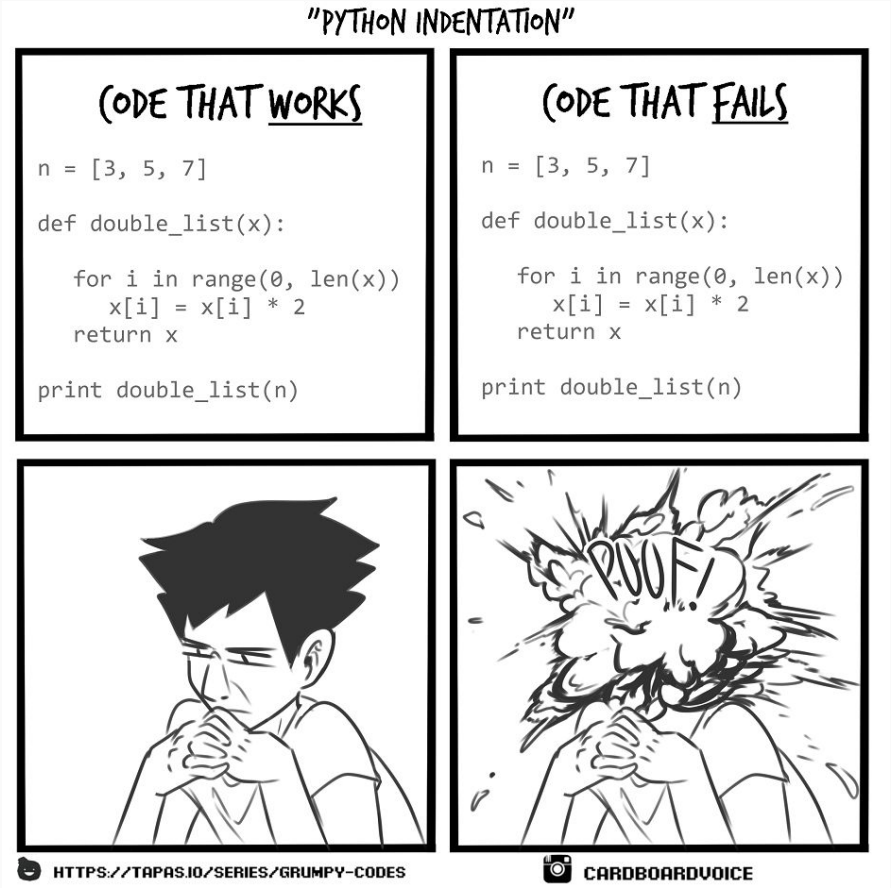
Python

Python

Vamos a aprender la sintaxis de Python de a poco, hoy vamos a ver y ejercitar

- Indentación
- Asignación
- Comentarios
- Tipos de datos:
  - Númerico
  - String
  - Bool
  - Listas diccionarios

Python: Indentación



Python: Asignación

Operación para modificar el valor de una variable

Sintaxis:

nombre_de_la_variable = expresion

¿Qué hace?

1. Evalúa la expresión de la derecha
2. Se guarda en algún lugar de la memoria el resultado de la expresión que puede ser llamada y usada mediante el *nombre_de_la_variable*

Ejemplos:

```
1.  x = 5
2.  y = x
3.  w = y+x+1
4.  res = funcion1(4,5,w)
```

Python: Comentarios

Los comentarios son textos que no se ejecutan en el programa, sirven para hacer más claro el código. En el contexto de esta materia son **obligatorios**

Sintaxis:

esto es un comentario

¿Qué hace?

1. Nada, solo anota el código para que sea más entendible

Ejemplos:

```
1.  # Guardo un x en la variable x
2.  x = 5
3.
4.  # Guardo en w, la suma de las variable x mas un 3
5.  w = x+3
6.
7.  # Ejecuto la funcion1 con los parametros 4,5 y w, guardo este valor en la variable res
8.  res = funcion1(4,5,w)
```

Python: Tipos de datos

En Python todos los valores tienen un **tipo de dato asociado.** Las variables **no** porque funcionan como etiquetas de los valores

Python provee algunos tipos de datos y las librerías que incluiremos agregaran más, también podremos crear nuestros propios tipos de datos.

- Numeros: enteros (int), decimales (float)
- Texto: Se denotan entre comillas
- Valores de verdad: True o False
- Listas y diccionarios

Python: Tipos de datos / Números

En Python todos los valores tienen un **tipo de dato asociado.** Las variables **no** porque funcionan como etiquetas de los valores

Python provee algunos tipos de datos y las librerías que incluiremos agregaran más, también podremos crear nuestros propios tipos de datos.

- Numeros: enteros (int), decimales (float)

Programa	Output

```
1.  # Creo dos variables a una le asigno un entero y a otro un
    decimal
2.  un_entero = 4
3.  un_decimal = 4.2
4.
5.  # Printeo los valores
6.  print('El valor de un_entero es:', un_entero)
7.  print('El valor de un_decimal es:', un_decimal)
8.  # Printeo los tipos de las dos variables, uso la funcion type
9.  print('El tipo de la variable un_entero es:', type(un_entero) )
10. print('El tipo de la variable un_decimal es:', type(un_decimal) )
```

```
El valor de un_entero es: 4
El valor de un_decimal es: 4.2
El tipo de la variable un_entero es: <class 'int'>
El tipo de la variable un_decimal es: <class 'float'>
```

Python: Operaciones sobre números

```
# Sumas (+)
4+9
```

```
13
```

```
# Restas (-)
9-2
```

```
7
```

```
# Multiplicacion (*)
4*2
```

```
8
```

```
# Division (/)
7/2
```

```
3.5
```

```
# Division entera (//)
7 // 2
```

```
3
```

```
# Resto o modulo (%)
7 % 2
```

```
1
```

```
# Potencia (**)
2**10
```

```
1024
```

Python: Tipos de datos / String o texto

- Texto (str) o strings: Se denotan entre comillas

Hay un montón de funciones sobre **strings,** por ejemplo, algunas:

- len: Cuenta la longitud
- .capitalize: Devuelve un string igual pero que empieza con mayúscula la primera letra
- .lower: Devuelve un string igual pero todo en minuscula
- + : toma dos string y devuelve uno nuevo concatenándolos

Hay un monton, googlearlas!

```
[7]: unString = 'esto es un string'

[8]: print('El tamano de unString es:',len(unString))
     El tamano de unString es: 17

[9]: #Contateno mas texto
     otroString = unString+', si!'

[10]: print(otroString)
      esto es un string, si!

[11]: print(len(otroString))
      22

[12]: print(unString.capitalize())
      Esto es un string
```

Python: Tipos de datos / Bool

- Valores de verdad: True o False

```
# Estas 2 variables son de tipo texto (str)
esto_es_un_texto = "Hola!"
esto_tambien = 'Chau'

# esto es una variable de tipo booleana (para modelar valor de verdad)
var_b = False
print('Imprimo valor y tipo de la variable var_b, valor:', var_b,' | tipo:', type(var_b))


x = 3
print('Imprimo valor y tipo de la variable x, valor:', x,' | tipo:', type(x))

x = 'abcd'
print('Imprimo valor y tipo de la variable x, valor:', x,' | tipo:', type(x))
```

Imprimo valor y tipo de la variable var_b, valor: False | tipo: <class 'bool'>
Imprimo valor y tipo de la variable x, valor: 3 | tipo: <class 'int'>
Imprimo valor y tipo de la variable x, valor: abcd | tipo: <class 'str'>

Python: Operaciones con resultados bool

```
# and (se cumplen ambas)
```

```
a = True
```

```
a and a
```

```
True
```

```
a and False
```

```
False
```

```
False and False
```

```
False
```

```
# or (se cumple alguna)
```

```
a = False
```

```
True or True
```

```
True
```

```
False or True
```

```
True
```

```
False or False
```

```
False
```

```
# not (niego)
```

```
not True
```

```
False
```

```
not False
```

```
True
```

Python: Operaciones con resultados bool

```
# == (igualdad)
```

```
3==3
```

```
True
```

```
3==2+1
```

```
True
```

```
"a"=="a"
```

```
True
```

```
4==3
```

```
False
```

```
# <, > (menor, mayor),
# <=, >=, menor igual,
# mayor igual
3<4
```

```
True
```

```
3<3
```

```
False
```

```
3<=4
```

```
True
```

```
3<=3
```

```
True
```

Python: Tipos de datos / Listas

list en Python, son listas de elementos, tienen orden y pueden tener elementos de muchos tipos distintos dentro.

Sintaxis:

[<elemento0>, <elemento1>, <elemento2>, ..., <elementoN>]

Por ejemplo:

```
unaLista = [ 1, 2, 10]
otraLista = [True, 1, False, 'hola', 'Chau']
y_otraLista = [ [1] ,2 , 3, [4,5] ]
```

Existen diversas funciones sobre listas, algunas son:

- Concatenar (+): Toma dos listas y las une
- Longitud (len): Cuenta la cantidad de elementos
- I-esimo ([i]): Toma el i-ésimo elemento de la lista
- Agregar al final (.append): agrega un elemento a la lista

Python: Tipos de datos / Listas

```
In [115]: v1 = [2,4,1,40,True,'hola',3,4]

In [116]: v1[0] # es el primer elemento de la lista
Out[116]: 2

In [117]: v2[-1] # el ultimo elemento del a lista
Out[117]: 6

In [118]: v1 = [2,4,1,40,True,'hola',3,4]

In [119]: v1[0] # es el primer elemento de la lista
Out[119]: 2

In [120]: v1[-1] # el ultimo elemento del a lista
Out[120]: 4

In [121]: v1[-2] # el ante-ultimo elemento del a lista
Out[121]: 3

In [122]: len(v1) # la longitud de la lista
Out[122]: 8

In [123]: v1[1:3] # devuelvo el intervalo [1,3)

Out[123]: [4, 1]
```

Python: Tipos de datos / Listas

list.append(x): Add an item to the end of the list; equivalent to a[len(a):] = [x].

list.extend(L): Extend the list by appending all the items in the given list; equivalent to a[len(a):] = L.

list.insert(i, x): Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x): Remove the first item from the list whose value is x. It is an error if there is no such item.

list.pop([i]): Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.

list.index(x):
Return the index in the list of the first item whose value is x. It is an error if there is no such item.

list.count(x):
Return the number of times x appears in the list.

list.sort(cmp=None, key=None, reverse=False): Sort the items of the list in place (the arguments can be used for sort customization, see sorted() for their
explanation).

list.reverse(): Reverse the elements of the list, in place.

Mas: https://docs.python.org/3/tutorial/datastructures.html

Python: Tipos de datos / Diccionarios

Los diccionarios son estructuras de datos que almacenan relaciones. Funciona como un diccionario común. Se tienen claves y valores.

Los valores pueden ser de cualquier tipo, las claves tienen que tener ciertas propiedades.

Las claves siempre son únicas! (no puede haber dos definiciones distintas para la misma key)

Sintaxis:

- un_dic = {key1:value1, key2:value2,...}

Python: Tipos de datos / Diccionarios

```
In [132]: un_dic = {} # esto es un diccionario vacio


In [133]: un_dic2 = { 'Juan':1, 'Maria':2, 'Laura':3} # esto es un dic inicializado con datos

In [134]: un_dic2['Juan'] # busco en el diccionario por la clave Juan y obtengo su valor
Out[134]: 1

In [135]: un_dic2['Facundo'] # si la clave no esta en el diccionario obtengo un error
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-135-b649f7338f79> in <module>
----> 1 un_dic2['Facundo'] # si la clave no esta en el diccionario obtengo un error

KeyError: 'Facundo'

In [136]: un_dic2
Out[136]: {'Juan': 1, 'Maria': 2, 'Laura': 3}

In [137]: un_dic2['Facundo'] = 'Hola!'

In [138]: un_dic2
Out[138]: {'Juan': 1, 'Maria': 2, 'Laura': 3, 'Facundo': 'Hola!'}
```

Python: Tipos de datos / Diccionarios

```
In [139]: un_dic2['Facundo'] # Ahora si deberia andar!
Out[139]: 'Hola!'

In [140]: un_dic2['Facundo'] = 'Chau' # Aca estoy pisando el valor la clave ya definida

In [141]: un_dic2
Out[141]: {'Juan': 1, 'Maria': 2, 'Laura': 3, 'Facundo': 'Chau'}


In [142]: del un_dic2['Facundo'] # Esto borra la entrada Facundo

In [143]: un_dic2

Out[143]: {'Juan': 1, 'Maria': 2, 'Laura': 3}


In [144]: un_dic2.clear() # Esto borra todo el diccionario

In [145]: un_dic2

Out[145]: {}
```

Pista: del tambien funciona para listas!

```
In [150]: l = [5,4,9]
In [151]: del l[1] # Borro el elemento en la posición 1
In [152]: l
Out[152]: [5,9]
```

Python:

Ya pueden terminar la práctica I

Link Colab Guía 1