

ACÀMICA

---

# ¡Bienvenidos/as a Data Science!



# Agenda

---

Equipo docente

¿Cómo anduvieron?

Actividad: Carta a mi yo del futuro

Proyecto 1: EDA

Repaso: ¿Qué es programar?

Explicación: tipos de datos

Hands-on training

Break

Listas, Loops y Condicionales

Hands-on training

Cierre



# Equipo Docente



# ¿Cómo anduvieron?



ACÁMICA



**¿Alguna duda con estos canales?**

# Actividad: Carta a mi yo del futuro



# ¿Dónde estamos?





# Cronograma

bloque

entrega

tiempo

## ADQUISICIÓN Y EXPLORACIÓN

## MODELADO

## DEPLOY

### Exploración de datos

### Feature Engineering

### Regresión

### Optimización de parámetros

### Procesam. del lenguaje natural

### Sistema de recomendación

### Publicación de modelos

SEM 1

SEM 5

SEM 8

SEM 12

SEM 13

SEM 18

SEM 23

SEM 2

SEM 6

SEM 9

SEM 14

SEM 19

SEM 24

SEM 3

SEM 7

SEM 10

SEM 15

SEM 20

SEM 4

SEM 11

SEM 16

SEM 21

SEM 17

SEM 22

APRENDIZAJE SUPERVISADO

APRENDIZAJE NO SUPERVISADO

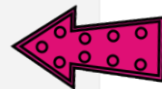


# BLOQUE 1

## Exploración de datos

Semana 1

Introducción a Data Science  
Python



Semana 2

Numpy  
Probabilidad y Estadística – Pandas

Semana 3

Probabilidad y Estadística – Pandas  
Matplotlib

Semana 4

Seaborn  
Práctica integradora para el proyecto

## Feature Engineering

Semana 5

Práctica integradora para el proyecto  
Transformación de datos con Pandas

Semana 6

Clases y objetos  
Scikit-Learn

Semana 7

¡DEMOS! (presentación de los trabajos de este  
BLOQUE 1)



# Repaso: ¿Qué es programar?



*“Programar es darle instrucciones a la computadora para que realice una función específica.”*

*“Programar es darle instrucciones a la computadora para que realice una función específica.”*



**¡ESTO NO ES UNA  
CARRERA DE  
PROGRAMACIÓN!**

**¿Y cómo lo  
vamos a hacer  
en esta carrera?**





General purpose and high level programming language.



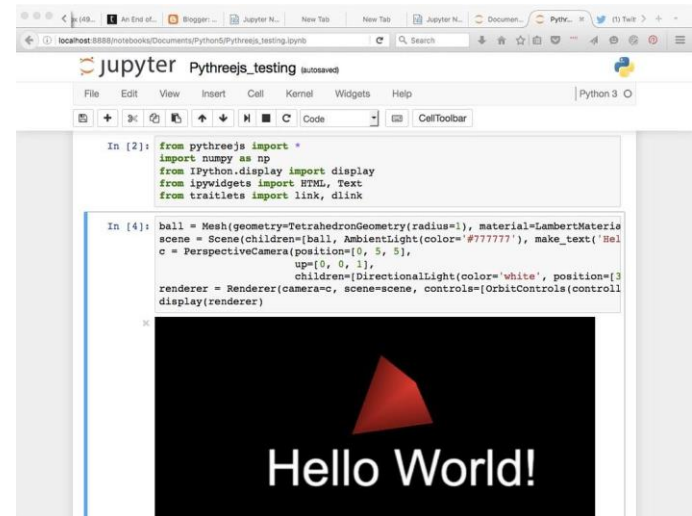
- Fácil de usar
- Rápido y eficiente
- Gran comunidad online
- Ampla cantidad de **librerías** específicas (¡pronto veremos qué son!)



# 1. Instalar Python

## ¿ALGUNA DUDA O PROBLEMA?

```
esteban@esteban-Lenovo-ideapad-FLEX-5-1570: ~  
esteban@esteban-Lenovo-ideapad-FLEX-5-1570:~$ source activate datascience  
(datascience) esteban@esteban-Lenovo-ideapad-FLEX-5-1570:~$
```



¡EMPEZAMOS!

## 1. Instalar Python

Vamos a instalar una distribución particular: [Miniconda](#).

Si ya tienen instalado Anaconda está perfecto. Si tienen otra distribución, instalar Miniconda preferiblemente.

1) Ir al link y descargar la versión correspondiente a su sistema operativo. **Importante:** Python 3

[Miniconda](#) 🔗

	Windows	Mac OS X	Linux
Python 3.7	<a href="#">64-bit (exe installer)</a>	<a href="#">64-bit (bash installer)</a>	<a href="#">64-bit (bash installer)</a>
	<a href="#">32-bit (exe installer)</a>	<a href="#">64-bit (.pkg installer)</a>	<a href="#">32-bit (bash installer)</a>

2) ¡Instalar!

3) Les va a preguntar si **COMPLETAR**



¡EMPEZAMOS!

## 2. Comprobar instalación

Vamos a instalar una distribución particular: [Miniconda](#).

1) Abrir una terminal (¡¿Qué es eso?!)

A screenshot of a terminal window. The title bar shows window control buttons and the text 'tele@locadelosgatos: ~'. The terminal content shows the prompt 'tele@locadelosgatos:~\$' followed by the command 'python' and a cursor. The background of the terminal is dark purple.

```
tele@locadelosgatos: ~  
tele@locadelosgatos:~$ python
```

2) Tipear “python”

3) Fijarse qué versión de Python les aparece


4) Poner “2+5”



# Recomendaciones para programar

---

- 1) Comentar el código en voz alta ayuda a aprender y a entender lo que estás haciendo.
- 2) No tengas miedo de hacer, romper y arreglar.
- 3) La frustración es una buena señal (“Get things done”).
- 4) Pedir la opinión de tus compañeros/as y mentores/as sobre tu código.
- 5) Busca crecer en comunidad (Medium, Github, Slack Stackoverflow, etc).
- 6) Pide ayuda a tu mejor amigo:



Google Search

I'm Feeling Lucky

Search by voice

# PRIMEROS PASOS CON PYTHON



## 1. Vimos que podemos crear variables y asignarles "texto" o "números"

```
[ ]: nombre = 'Esteban'
```

```
[ ]: edad = 31
```

# PRIMEROS PASOS CON PYTHON



## 1. Vimos que podemos crear variables y asignarles "texto" o "números"

```
[ ]: nombre = 'Esteban'
```

```
[ ]: edad = 31
```

## 2. También podemos hacer listas de cosas

```
[ ]: primeros_10 = [0,1,2,3,4,5,6,7,8,9]
```



## 1. Vimos que podemos crear variables y asignarles “texto” o “números”

```
[ ]: nombre = 'Esteban'
```

```
[ ]: edad = 31
```

## 2. También podemos hacer listas de cosas

```
[ ]: primeros_10 = [0,1,2,3,4,5,6,7,8,9]
```

## 3. Y podemos imprimir en pantalla texto y el valor de las variables

```
[ ]: print(nombre, edad)
```

```
[ ]: print('Mi nombre es', nombre, '. Mi edad es ', edad, 'años')
```

```
[ ]: print('Mi nombre es {}. Mi edad es {} años'.format(nombre, edad))
```

## PRIMEROS PASOS CON PYTHON



### UN LENGUAJE DE PROGRAMACIÓN VIENE CON...

#### **tipos de datos**

Números, texto, variables de verdad (bool), etc.

#### **estructuras de datos**

Podemos hacer “conjuntos” de cosas y agruparlas de formas específicas. ¡Y vienen con funcionalidades propias! Ejemplo: listas.

#### **funciones propias**

Ejemplo: print(), type(), etc.

Vimos, además, que podemos definir **Variables**.





## VARIABLES

En un lenguaje de programación, a los datos se los guarda en forma de variables. A cada variable debemos darle un nombre único que la identifique:

```
In [ ]: a = 5
```

```
In [ ]: un_nombre_cualquiera = 12.7
```

```
In [ ]: b = 'Hola!'
```

```
In [ ]: nueva_variable = True
```

A estas **variables** pueden se le pueden asignar distintos **tipos de datos**.



## VARIABLES y TIPOS DE DATOS

Python identifica automáticamente el **tipo de dato** de cada variable. Esto resulta muy cómodo para trabajar.

```
In [ ]: a = 5
```

→ Entero

```
In [ ]: un_nombre_cualquiera = 12.7
```

→ Float

```
In [ ]: b = 'Hola!'
```

→ String

```
In [ ]: nueva_variable = True
```

→ Boolean

Pero debemos ser cuidadosos, **a veces** el tipo asignado automáticamente **no es el que esperamos ...**



## TIPOS DE DATOS

¿Podemos pasar de un **tipo de dato a otro**?

**¡Sí!** La **solución** es ser explícitos si deseamos que nuestra variable sea de algún tipo en particular.

```
In [6]: numero_entero = int(2.0)  
        type(numero_entero)
```

```
Out[6]: int
```



Podemos especificar el tipo de dato que queremos

```
In [8]: numero_en_texto = '5'  
        type(numero_en_texto)
```

```
Out[8]: str
```



Podemos consultar el tipo de dato de una variable

# PRIMEROS PASOS CON PYTHON



## TIPOS DE DATOS

Enteros	Floats	Strings	Booleanos
Son los números que usamos para contar, el 0 y los negativos	Son los números "con coma"  Se introducen usando puntos	Texto  Se introducen entre comillas dobles, "", o simples, ' '.	Variables de "verdad": verdadero o Falso
-1 0 1 2	5.1 -1.3 1.0 10.0	"Hola Mundo" "A" 'Mi nombre es Esteban'	True False 1 == 2 1 == 1
[1]: <code>type(3)</code>  [1]: <code>int</code>	[1]: <code>type(3.0)</code>  [1]: <code>float</code>	[1]: <code>type("Hola")</code>  [1]: <code>str</code>	[1]: <code>type(2==2)</code>  [1]: <code>bool</code>

# Operaciones con variables

Distintos **tipos de datos** permiten realizar distintas **tipos de operaciones**.

```
In [11]: b = 'Hola!'
c = ' Como estas?'
print(b + c)

Hola! Como estas?
```

```
In [12]: x = 5
y = 7
print(x+y)

12
```

```
In [13]: variable_1 = True
variable_2 = False
print(variable_1 or variable_2)

True
```

---

El resultado de estas **operaciones** dependen del **tipo de variable**:

```
In [14]: x = '5'
y = '7'
print(x+y)

57
```

```
In [17]: b = 'Hola!'
c = 8
print(b + c)

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-17-0721abbbb84d> in <module>()
      1 b = 'Hola!'
      2 c = 8
----> 3 print(b + c)

TypeError: must be str, not int
```

# Operaciones básicas entre ENTEROS y FLOATS

```
In [42]: x = 3  
        y = 1.5  
        print(x/y)  
  
2.0
```

```
In [43]: x = 2  
        y = 3  
        print(x**y)  
  
8
```

```
In [44]: x = 10  
        y = 3  
        print(x%y)  
  
1
```

Operación	Operador	Ejemplo
Suma	+	3 + 5.5 = 8.5
Resta	-	4 - 1 = 3
Multiplicación	*	3 * 6 = 18
Potencia	**	3 ** 2 = 9
División (cociente)	/	15.0 / 2.0 = 7.5
División (parte entera)	//	15.0 // 2.0 = 7
División (resto)	%	7 % 2 = 1

# Operaciones básicas con STRINGS

```
In [32]: txt_1 = 'Los textos'
         txt_2 = ' se concatenan.'
         print(txt_1 + txt_2)
```

Los textos se concatenan.

```
In [33]: b = 'Los textos'
         c = ' no se restan.'
         print(b - c)
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-33-def6c3f4c5e8> in <module>()
      1 b = 'Los textos'
      2 c = ' no se restan.'
```

```
----> 3 print(b - c)
```

TypeError: unsupported operand type(s) for -: 'str' and 'str'

```
In [34]: txt_3 = 'Los textos se multiplican. '
         print(txt_3 * 2)
```

Los textos se multiplican. Los textos se multiplican.

# Operaciones lógicas

Un tipo importante de operación en programación son las **operaciones lógicas**. Estas pueden realizarse sobre **variables booleanas**.

```
In [27]: variable_1 = True  
         variable_2 = False  
         print(variable_1 or variable_2)
```

True

```
In [28]: print(not(variable_1))
```

False

El resultado es también una **variable booleana**.

A	B	A & B
False	False	False
False	True	False
True	False	False
True	True	True

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

A	A!
False	True
True	False



# Hands-on training



# DS\_Clase\_2\_Python.ipynb

## Sección 1



A close-up photograph of a white ceramic cup filled with a latte. The coffee has a thick layer of white foam on top, decorated with intricate brown latte art that resembles a stylized leaf or feather pattern. The cup is placed on a matching white saucer. In the background, a blurred white napkin with a silver fork and knife is visible on a dark blue surface.

**¡BREAK!**

---

# Listas



# Definición

Una estructura de dato muy importante en Python son las **listas**.

Una lista consiste en una serie de elementos ordenados:

```
In [47]: lista_1 = [2, 4.7, True, 'Texto']  
         type(lista_1)
```

```
Out[47]: list
```

```
In [49]: lista_2 = [0, lista_1, 'Mas texto']  
         print(lista_2)  
[0, [2, 4.7, True, 'Texto'], 'Mas texto']
```

→ Los elementos pueden ser de distintos tipos.

→ Incluso puede haber listas dentro de listas.

Las **listas** se definen con corchetes [ ]

# Operaciones con LISTAS

Las listas se pueden **sumar** entre sí (se **concatenan**). También se les puede agregar un elemento nuevo mediante el método **'append()'**

```
In [52]: lista_1 = [2, 4.7, True, 'Texto']  
        lista_2 = [42, 42]  
        lista_1 + lista_2
```

```
Out[52]: [2, 4.7, True, 'Texto', 42, 42]
```

```
In [53]: lista_1 = [2, 4.7, True, 'Texto']  
        lista_1.append('Un nuevo elemento')  
        lista_1
```

```
Out[53]: [2, 4.7, True, 'Texto', 'Un nuevo elemento']
```

# Operaciones con LISTAS

```
In [55]: lista_1 = [2, 4.7, True, 'Texto']  
len(lista_1)
```

```
Out[55]: 4
```

```
In [56]: lista_2 = [0, lista_1, 'Mas texto']  
len(lista_2)
```

```
Out[56]: 3
```

```
In [59]: lista_vacia = []  
len(lista_vacia)
```

```
Out[59]: 0
```

```
In [60]: lista_vacia.append(42)  
lista_vacia.append('un segundo item')  
print(lista_vacia)
```

```
[42, 'un segundo item']
```

Las listas tienen un largo determinado por su cantidad de elementos. Se consulta mediante la función **len()**.

Se pueden generar listas vacías y luego ir agregándole elementos a medida que una lo precise.

# Loops





# LOOPS - For

Los **Loops** en programación son bloques de código que, dadas ciertas condiciones, se repiten una cierta cantidad de veces.

El **For** es un tipo de **Loop** que repite un bloque de código tantas veces como elementos haya en una **lista** dada:

```
In [62]: lista_1 = [0, 1, 2, 3]
         for item in lista_1:
             print('Hola.')
```

```
Hola.
Hola.
Hola.
Hola.
```

Lo que está dentro del cuerpo del loop se identifica por su **indentación**. En este caso se repite 4 veces, porque la lista tiene 4 elementos

# LOOPS - For

En cada repetición, la variable **item** (podría tener cualquier nombre) va tomando el valor de cada uno de los elementos de la lista dada.

```
In [64]: lista_1 = [10, 20, 30,]
```

```
for item in lista_1:  
    doble = 2*item  
    print(doble)
```

```
20  
40  
60
```

→ Todo lo que está indentado se repite 3 veces, pero en cada repetición el valor de **item** va cambiando

# LOOPS - For

Las listas puede contener texto. Veamos un ejemplo donde creamos una nueva lista.

```
In [66]: lista_nombres = ['Ernesto', 'Camilo', 'Violeta']
nueva_lista = []

for item in lista_nombres:
    oracion = 'Mi nombre es ' + item
    nueva_lista.append(oracion)

print(nueva_lista)

['Mi nombre es Ernesto', 'Mi nombre es Camilo', 'Mi nombre es Violeta']
```

Estas dos líneas se repiten 3 veces. Le agregamos texto a cada elemento y lo agregamos a una nueva lista.

# Condicionales



# CONDICIONALES - if

Los **condicionales** son bloques de código que se ejecutan únicamente si se cumple una condición. El resultado de esta condición debe ser un **Booleano** (True o False). Esto se logra mediante el condicional **if**.

```
[10]: valor = 5
      if valor > 10:
          print('El valor es mayor que 10')
```

5 > 10

False

No se cumple la condición.

```
[11]: valor = 15
      if valor > 10:
          print('El valor es mayor que 10')
```

15 > 10

True

El valor es mayor que 10

Se cumple la condición.

# CONDICIONALES - if / else

Además uno puede agregar un código que se ejecute si la condición no se cumple. Para esto se utiliza el condicional **else**.

```
In [77]: nombre = 'Pedro'

if nombre == 'Juan':
    print('Esta persona se llama Juan')
else:
    print('Esta persona NO se llama Juan')
```

Esta persona NO se llama Juan

```
'Pedro' == 'Pedro'
```

True

```
'Juan' == 'Pedro'
```

False

La comparación entre strings también genera un booleano.

*Nota: Para condicionales usamos doble igual ==, ya que nos reservamos el igual simple = para la asignación de variables.*

## CONDICIONALES - if / elif / else

Además del **if** y el **else**, uno puede agregar más condiciones a través de condicional **elif** (else if). De esta forma se puede agregar un número arbitrario de condiciones.

```
In [80]: edad = 20

if edad < 18:
    print('Esta persona tiene menos de 18 años')
elif edad > 18:
    print('Esta persona tiene mas de 18 años')
else:
    print('Esta persona tiene justo 18 años')
```

Esta persona tiene mas de 18 años

# Combinando LOOPS y CONDICIONALES

Los distintos loops y condicionales que vimos se pueden combinar para generar procedimientos más complejos.

```
In [81]: lista_de_edades = [4,20,15,29,11,42,10,18]
        lista_mayores = []

        # Queremos armar una lista solo con las edades mayores o iguales a 18
        for edad in lista_de_edades:
            if edad >= 18:
                # Agremos a la lista de mayores
                lista_mayores.append(edad)

        print(lista_mayores)

[20, 29, 42, 18]
```



# Hands-on training



DS\_Clase\_2\_Python.ipynb

Sección 2



# Recursos



# Intro a Python

<https://learnxinyminutes.com/docs/python3/>



# ¡RECUERDEN!

- 1) No podrás entregar un proyecto si no entregaste el anterior.
- 2) Agenda las fechas de entrega en tu calendario para no atrasarte en la clase.
- 3) Descarga los “Checklist” antes de comenzar tus proyectos para tener claridad sobre qué esperamos que entregues.
- 4) Los “Checklist” son la base. ¡Recomendamos ejercitar tu creatividad y personalizar los proyectos con tu impronta!

# Para la próxima

---

1. Ejercicio: a) Googlear qué es un diccionario de Python y crear uno.  
b) ¿Qué hace la función *range*? Crear un ejemplo.
2. Ver los videos de la plataforma “Biblioteca: NumPy” y, si tienen tiempo, “Biblioteca: Pandas”.
3. Completar el notebook de hoy.



ACÀMICA