

ACÀMICA

---

# ¡Bienvenidos/as a Data Science!



# Agenda

---

¿Cómo anduvieron?

Presentaciones Orales

Break

Explicación: Clases

Hands-on training

Consultas Entrega 01 y Lanzamiento Entrega 02

Cierre



# ¿Cómo anduvieron?



# Proyecto 1:

# Análisis Exploratorio

# de Datos (EDA)



# Análisis Exploratorio de Datos (EDA)

fase	ADQUISICIÓN Y EXPLORACIÓN		MODELADO				DEPLOY
	Exploración de datos	Feature Engineering	Regresión	Optimización de parámetros	Procesam. del lenguaje natural	Sistema de recomendación	Publicación de modelos
entrega	SEM 1	SEM 5	SEM 8	SEM 11	SEM 13	SEM 18	SEM 22
	SEM 2	SEM 6	SEM 9	SEM 12	SEM 14	SEM 19	SEM 23
	SEM 3	SEM 7	SEM 10		SEM 15	SEM 20	SEM 24
	SEM 4				SEM 16	SEM 21	
tiempo					SEM 17		



# Proyecto EDA: Hoja de ruta

---

Usted  
Está Aquí

## Entrega 1

### SEM 1 - 4

- Python
- Numpy
- Pandas
- Visualización de datos: Matplotlib y Seaborn
- Estadística

### SEM 5

- Ejercicio/s Integradores
- Programación: Funciones
- Transformación de Datos
- Transformación de Datos con Pandas

## Entrega 2

### SEM 6

- Programación: Clases
- Cómo dar Presentaciones Orales
- Transformación de Datos con Scikit-Learn

### SEM 7

- Transformación de Datos con Scikit-Learn
- Outliers
- Puesta en Común Proyecto 1

### SEM 8

- ¡Arrancamos con Machine Learning!



# DEMO:

## ¿Cómo preparo mi presentación?





# No pienses la DEMO como evaluación.

- Es **practicar** para cuando presentes en el mundo profesional.
- Es **reflexionar** sobre la práctica.
- Es retroalimentar **tu perspectiva con otras**.
- Es **compartir** y abrirse para **mejorar**.

**¡Y es una oportunidad para celebrar los logros!**

# LA GUÍA COMPLETA PARA NO ENTRAR EN PÁNICO



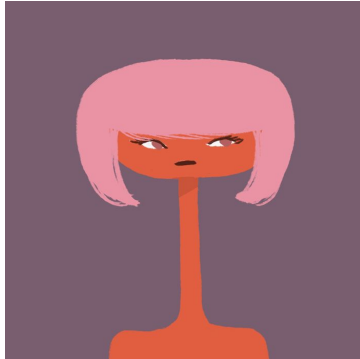


*“Dime lo que puedes  
hacer y muéstramelo.”*

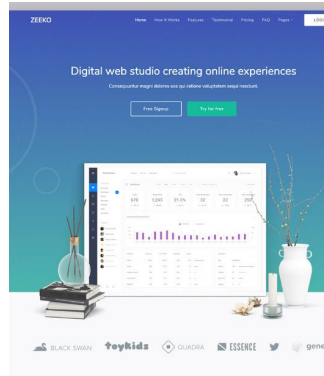
– Dr. Kwegyir Aggrey

# En su presentación, buscamos que:

Nos cuenten



y que sobre todo, ¡NOS MUESTREN!



```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.screen_name, count=10)
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate, '%a %b %d %H:%M:%S %Y')
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past', howlong, 'days'
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', howlong, 'days'
```

# La regla de las 3 Ps:

**1**

**PREPARAR**

**2**

**PRACTICAR**

**3**

**PRESENTAR**

# La regla de las 3 Ps:

**1**

**PREPARAR**

2

PRACTICAR

3

PRESENTAR

# 1

## PREPARAR

Sugerimos que armen un esqueleto de presentación (Word, Excel, etc.) con lo que van a presentar teniendo en cuenta los siguientes puntos:

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

Le presentan a la clase y al equipo docente.

**¿Qué les puede servir a ellos/as de su experiencia en este proyecto?**





# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

### Viaje

¿Con qué obstáculos te encontraste?

¿Qué decisiones tomaste?

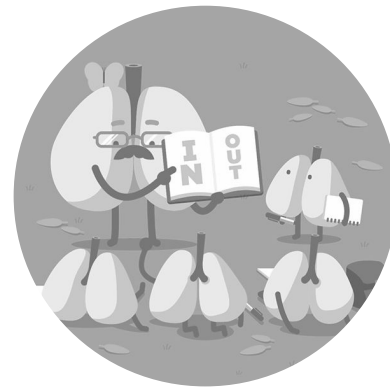
¿Qué quisiste lograr?

### Puerto de llegada

¿Cuáles son los resultados que alcanzaste?

¿Qué te gustaría mejorar?

¿Qué aprendiste?



# 1

## PREPARAR

¿A quién?

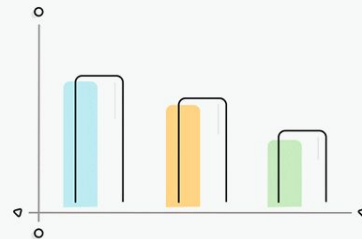
¿Qué?

¿Cómo?

¿Tiempo?

\*TIP: usa fotos, gráficos, diagramas...en fin, elementos visuales.

**¡El 65% de las personas son pensadores puramente visuales!**



# 1

## PREPARAR

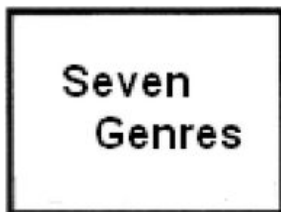
¿A quién?

¿Qué?

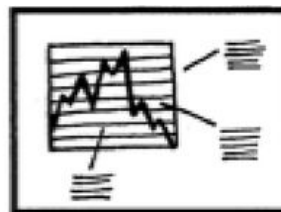
¿Cómo?

¿Tiempo?

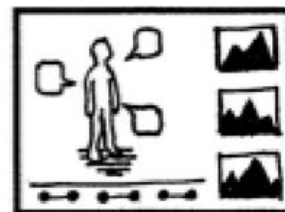
### Los 7 géneros de Edward Segel



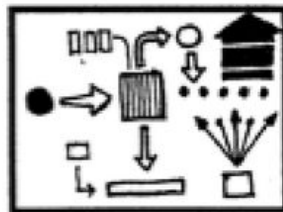
Magazine Style



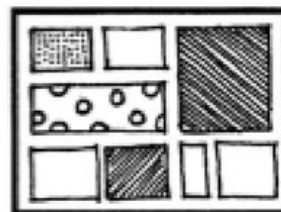
Annotated Chart



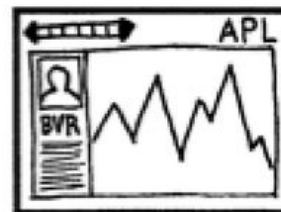
Partitioned Poster



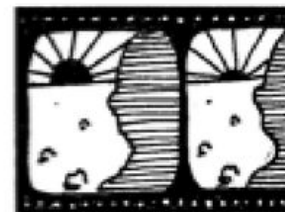
Flow Chart



Comic Strip



Slide Show



Film/Video/Animation

# 1

# PREPARAR

¿A quién?

¿Qué?

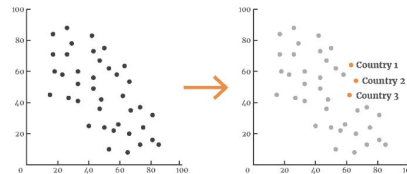
¿Cómo?

¿Tiempo?

## Core Principles of Data Visualization

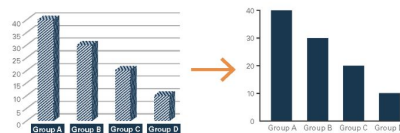
### Show the data

People read graphs in a research report, article, or blog to understand the story being told. The data is the most important part of the graph and should be presented in the clearest way possible. But that does not mean that all of the data must be shown—indeed, many graphs show too much.



### Reduce the clutter

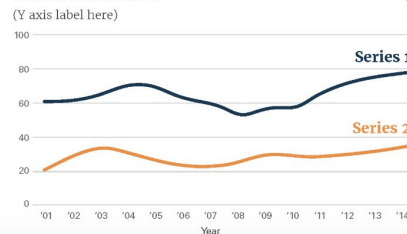
Chart clutter, those unnecessary or distracting visual elements, will tend to reduce effectiveness. Clutter comes in the form of dark or heavy gridlines; unnecessary tick marks, labels, or text; unnecessary icons or pictures; ornamental shading and gradients; and unnecessary dimensions. Too often graphs use textured or filled gradients.



### Integrate the text and the graph

Standard research reports often suffer from the **slideshow effect**, in which the writer narrates the text elements that appear in the graph. A better model is one in which visualizations are constructed to complement the text and at the same time to contain enough information to stand alone. As a simple example, legends that define or explain a line, bar, or point are often placed far from the content of the graph—off to the right or below the graph. Integrated legends—right below the title, directly on the chart, or at the end of a line—are more accessible.

### Chart Title Here



# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

### Ejemplos:

<http://drones.pitchinteractive.com/>

<https://www.nytimes.com/interactive/2017/06/14/world/europe/migrant-rescue-efforts-deadly.html>

<https://projects.propublica.org/houston-cypress/>

<https://projects.propublica.org/graphics/harvey-maps>

<https://podio.com/site/creative-routines>

<https://www.bloomberg.com/graphics/2015-whats-warming-the-world/>

<https://www.axios.com/wall-street-asset-management-climate-change-activism-19fe3b28-eab2-4b1c-951b-1c1c2ebe8f33.html>

# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

¿Presento

- **primero el resultado final** y luego el proceso o
- cuento el proceso **cronológicamente** hasta el resultado al final?

# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

### ESTRUCTURA VISUAL

*apoya la  
historia*

# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

### ESTRUCTURA VISUAL

*apoya la  
historia*

### MENSAJE

*cuenta la  
historia*



# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

**ESTRUCTURA  
VISUAL**

*apoya la  
historia*

**MENSAJE**

*cuenta la  
historia*

**INTERACTIVIDAD**

*hazla dinámica y  
atrapante*



# 1

## PREPARAR

¿A quién?

¿Qué?

¿Cómo?

¿Tiempo?

Cada uno/a tendrá  
**10 minutos como máximo para exponer**  
(luego podremos hacerle preguntas y/o  
darle feedback).

# La regla de las 3 Ps:

1

PREPARAR

2

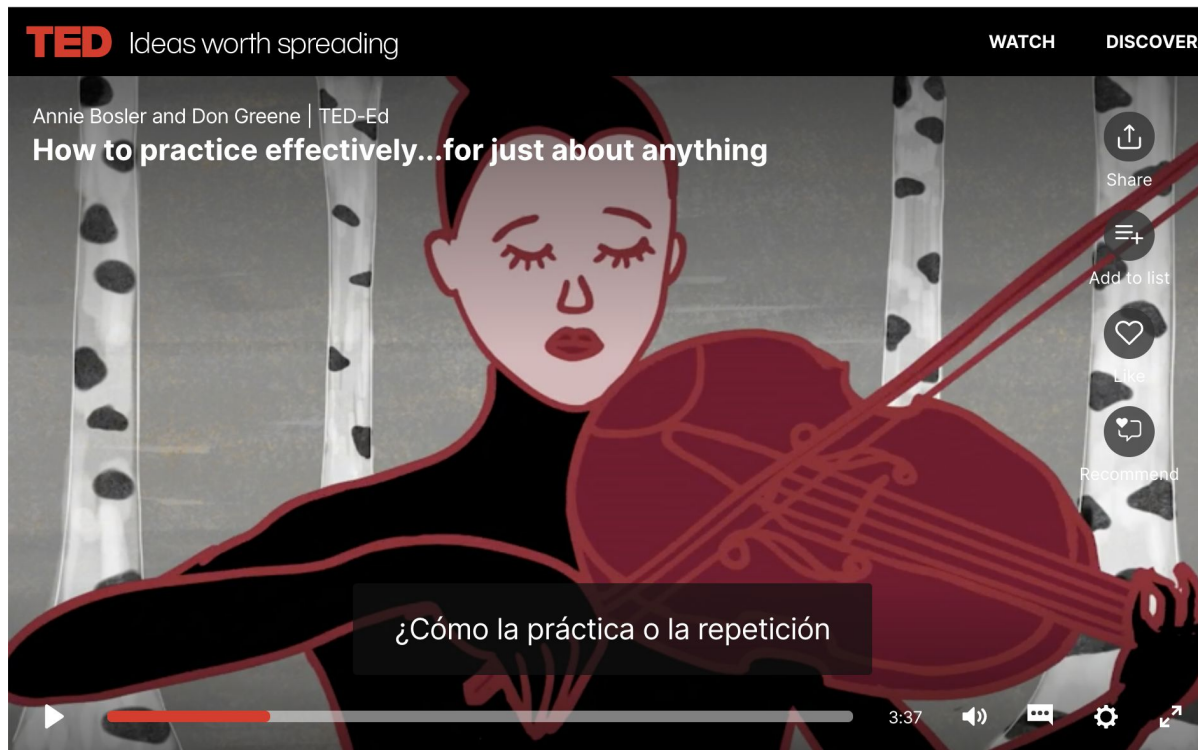
**PRACTICAR**

3

PRESENTAR

## 2

# PRACTICAR



# La regla de las 3 Ps:

1

PREPARAR

2

PRACTICAR

3

**PRESENTAR**

# 3

## PRESENTAR

### Emoción

### Actitud

### Intención

**Si demuestras entusiasmo por tu trabajo** (sin importar a lo que hayas llegado), vas a contagiar al público.

**Conecta con tu entorno.** Aunque te dé vértigo, miedo y timidez, ¡mira a los ojos!

**\*TIP: decir en voz alta “estoy muy nervioso/a” te ayudará a relajarte.** 😊

# 3

## PRESENTAR

Emoción

Actitud

Intención

Si pretendes diseñar tu presentación con una determinada emoción, ¡acompañala con tu **cuerpo, gestos y cara!**



# 3

## PRESENTAR

Emoción

Actitud

Intención

¿Buscas feedback del público?  
**¡Pídelo al terminar!**



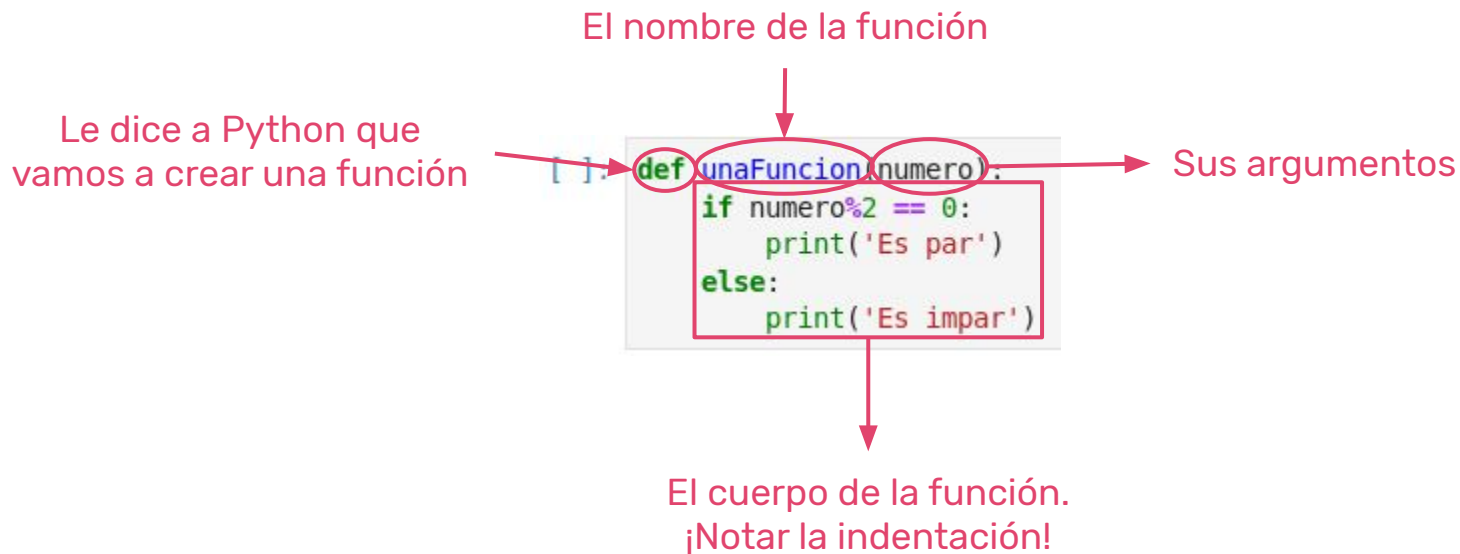
# Repaso: Funciones



Una función es un bloque de código que sólo *corre* cuando es *llamado*.

```
[ ]: def unaFuncion(numero):  
    if numero%2 == 0:  
        print('Es par')  
    else:  
        print('Es impar')
```

Una función es un bloque de código que sólo *corre* cuando es *llamado*.



## Vimos que las funciones...

- Podían tener argumentos por default

```
[7]: def division(dividendo, divisor = 2):  
      print(dividendo/divisor)
```

- Devolver resultados

```
[9]: def division(dividendo, divisor = 2):  
      variable_auxiliar = dividendo/divisor  
      return variable_auxiliar
```

- Las funciones pueden usar variables que *existen* por fuera de ellas (variables globales), pero que las variables definidas dentro de las funciones no *existen* fuera de ellas.  
¿Con qué concepto se relaciona?

- Existen un tipo de funciones especiales llamadas funciones anónimas (funciones lambda).  
¿Cuándo conviene usarlas?

```
[18]: lambda_division = lambda x,y: x/y  
      lambda_division(80,10)
```

```
[18]: 8.0
```



A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup is placed on a matching white saucer. In the background, a white napkin and a silver fork are visible, though they are out of focus. The overall lighting is soft and even, highlighting the textures of the coffee and the smooth surface of the cup.

**¡BREAK!**

---

Ph. Credit: Drew Coffmann



# Programación: Clases y Objetos



# Lenguajes de Programación

## Procedural Programming Languages

---

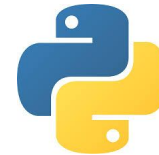


**BASIC**

Fortran

## Object Oriented Programming Languages (OOP)

---



# Lenguajes de Programación



Procedural Programming  
Languages

---



BASIC

Fortran

Object Oriented  
Programming Languages (OOP)

---





# ¿Qué es un objeto?

Un objeto es como un un paquete de variables y funciones que conviene tener agrupados por consistencia y comodidad.

Persona\_nombre = 'Juan'

Persona\_edad = 23

Persona\_profesion = 'vendedor'



## Objeto

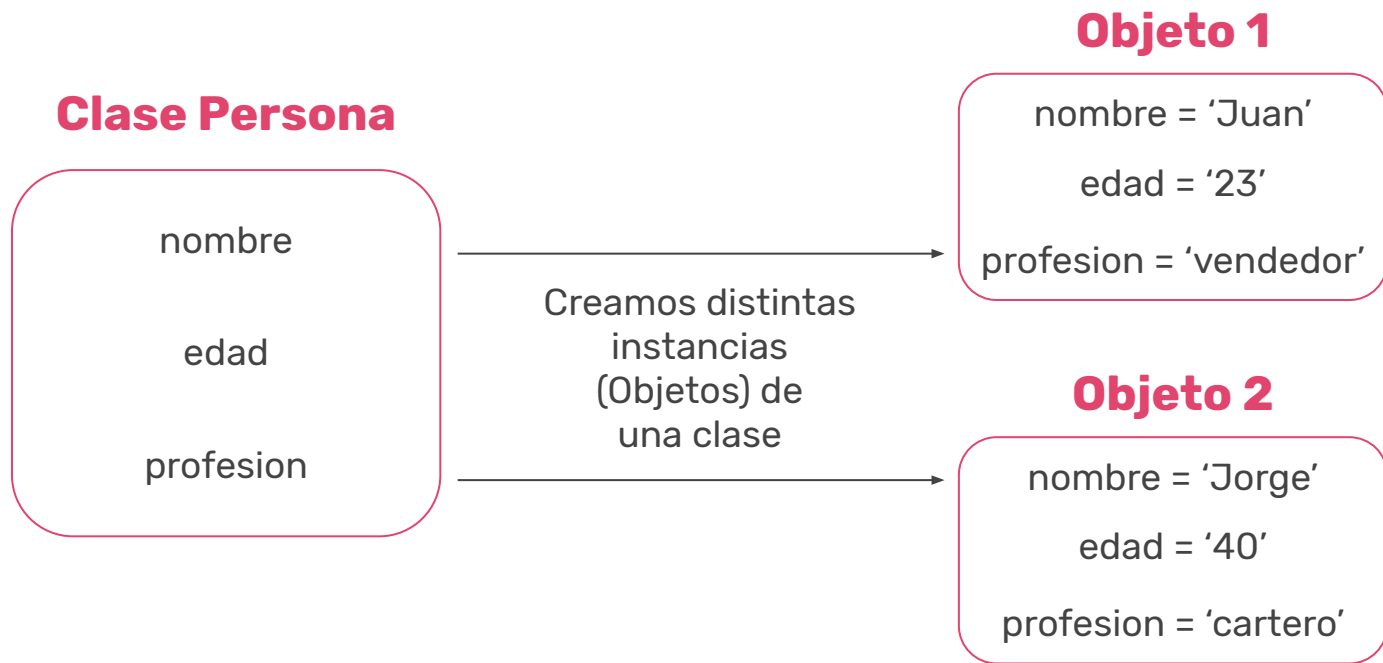
nombre = 'Juan'

edad = 23

profesión = 'vendedor'

# ¿Qué es una clase?

Los objetos suelen crearse a partir de unas *plantillas* a las que llamamos clases.



# Creando una **clase**

Nombre que le doy a la clase.

```
[ ]: class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad
```

# Creando una **clase**

```
[ ]: class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad
```

Nombre que le doy a la clase.

Atributos (variables) que van a tener los objetos de esta clase.

# Creando una **clase**

```
[ ]: class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad
```

Nombre que le doy a la clase.

Cuando creamos una instancia de esta clase, vamos a tener que pasarle un nombre y una edad de la persona

Atributos (variables) que van a tener los objetos de esta clase.

# Creando un objeto

```
[7]: p1 = Persona("Juan", 26)
```

```
print(p1.name)
```

```
print(p1.edad)
```

Juan

26

# Creando un **objeto**

```
[7]: p1 = Persona("Juan", 26)
```

```
print(p1.name)  
print(p1.edad)
```

```
Juan  
26
```

Recordar: lo que hicimos fue crear una instancia de la clase Persona

```
[ ]: class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad
```

# Creando un **objeto**

```
[7]: p1 = Persona("Juan", 26) →
```

```
print(p1.name)  
print(p1.edad)
```

Juan  
26

Recordar: lo que hicimos fue crear una instancia de la clase Persona

```
[ ]: class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad
```





# Métodos

Los métodos son funciones propias de una clase. Es decir son funciones que definimos para que actúan sobre un tipo de objeto determinado.

```
[15]: class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def mePresento(self):
        print("Hola, me llamo " + self.nombre)
```

# Métodos

```
[15]: class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def mePresento(self):
        print("Hola, me llamo " + self.nombre)
```

Definimos una función  
que imprime un texto  
en pantallas

# Métodos

```
[15]: class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def mePresento(self):
        print("Hola, me llamo " + self.nombre)
```

```
[16]: p1 = Persona("Juan", 26)
      p1.mePresento()
```

Hola, me llamo Juan

→ Usamos el método

# Métodos

Un método puede modificar el valor de algún atributo del objeto.

```
[34]: class Persona:
    """
    Esta es una clase donde se agregan todos los datos
    respecto a una persona
    """
    def __init__(self, nombre, edad):
        # Todo lo que definamos en __init__ se corre
        # al crear una instancia de la clase
        self.nombre = nombre
        self.edad = edad
    def cumplirAños(self):
        self.edad = self.edad + 1
```

**Cambia el valor del atributo edad, lo aumenta en 1.**

# Métodos

Un método puede modificar en valor de algún atributo del objeto.

```
[34]: class Persona:
      """
      Esta es una clase donde se agregan todos los datos
      respecto a una persona
      """
      def __init__(self, nombre, edad):
          # Todo lo que definamos en __init__ se corre
          # al crear una instancia de la clase
          self.nombre = nombre
          self.edad = edad
      def cumplirAños(self):
          self.edad = self.edad + 1
```

```
[35]: p1 = Persona("Juan", 26)
      p1.cumplirAños()
      p1.edad
```

```
[35]: 27
```



# Métodos: Beneficios

La programación orientada a objetos nos propone una **manera de trabajar** a la hora de escribir nuestro programa.

No sólo es práctico y ordenado, sino que también muchas veces nos ayuda a mantener la consistencia del código.

```
[1]: class Departamento:
      def __init__(self, calle, altura, sup_total, sup_cubierta):
          self.calle = calle
          self.altura = altura
          self.sup_total = sup_total
          if sup_cubierta < sup_total:
              self.sup_cubierta = self.sup_cubierta
          else:
              self.sup_cubierta = self.sup_total

[3]: depto_1 = Departamento('Humboldt', 1122, 50, 455)
      depto_1.sup_cubierta
```

```
[3]: 50
```

# Métodos: Beneficios

Más adelante veremos que el formato de clases y métodos propuestos por la librería scikit-learn es muy útil y es el más usado en la comunidad de data science.





# Hands-on training





# DS\_Clase\_11\_Classes.ipynb



# Entrega 1: Exploración



---



**Entrega 1: Exploración** 

Adquirir, explorar y visualizar tus datos

---

 Beginner by  Francisco Dorr

# Actividad:

## Dudas comunitarias

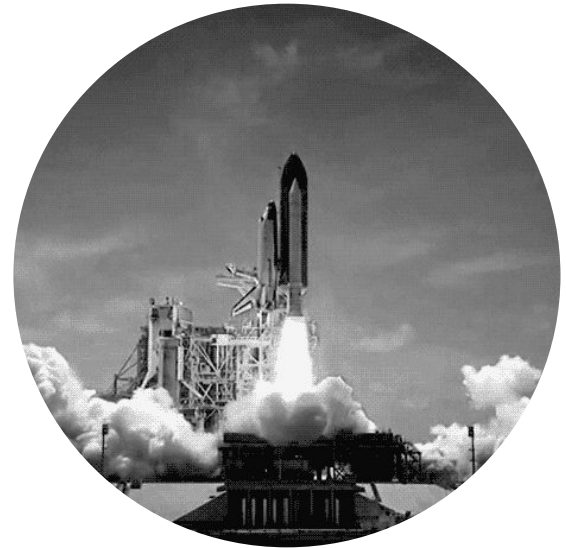


# Lanzamiento Entrega 02

---

¿Alguna duda con:

1. Notebook
2. Datos
3. Checklist?



# Para la próxima

---

1. Completar Notebooks de la clase pasada y de hoy.
2. Ver los videos de la plataforma “Transformación de Datos: Scikit-Learn
3. Terminar Entrega 01 y/o arrancar Entrega 02

ACÀMICA