

Visualización y Análisis de Datos con Python

Juan Pablo Zaldumbide Proaño



Contenido

- Introducción.
- Instalación de herramientas.
- Lenguaje de programación Python.
- Uso y sintaxis, ejercicios varios.
- Escritura y lectura de archivos planos.
- Instalación de librerías externas.
- Carga de datos de diferentes fuentes y tipos.
- Uso de numpy, pandas y matplotlib.
- Visualización de datos
- Ejercicios

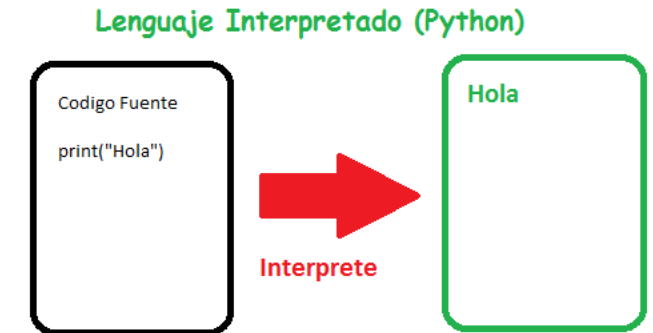


¿Qué lenguajes de programación conozco?



Python es un lenguaje de programación interpretado, multiparadigma, multiplataforma, con una curva de aprendizaje muy corta.

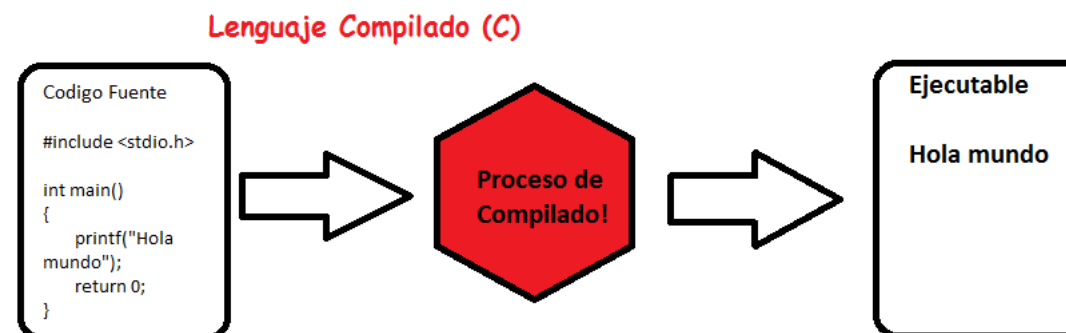
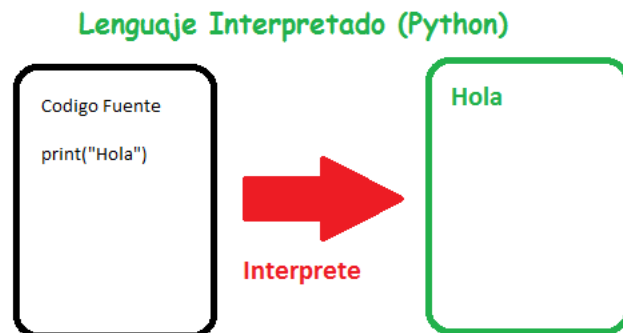
Diseñado por Guido van Rossum cuya primera versión salió en el año 1991.



Características:

- 1.- Lenguaje de programación de alto nivel.
- 2.- Fácil de leer y entender.
- 3.- Lenguaje interpretado.
- 4.- Multiplataforma.
- 5.- Gratuito y de código abierto

Lenguaje de Alto Nivel C	Lenguaje Ensamblador
<pre>SWITCH TYPE) { case 'a': type=type+10; break; case 'b': type= type+20; break; default: break;}</pre>	<pre>MOV1 R1 = Type LD4 R2 = [R1] ;; cmp.eq P1, P2 = 'a', R2 cmp.eq P3, P4 = 'b', R2 ;; (P1) Add R2 = 10, R2 (P3) Add R2 = 20, R2 ;; st4 (R1) = R2 default::</pre>



Características:

6.- Multiparadigma.

Imperativo:

Los lenguajes de programación también se pueden agrupar en imperativos y declarativos, los del primer grupo son aquellos que describen el estado del programa y permiten su modificación mediante condiciones o instrucciones de código que le indican al computador cómo realizar una tarea.

Funcional:

La programación funcional es un paradigma de la programación declarativa basada en el uso de funciones matemáticas que permite la variación del programa mediante la mutación de variables. Esto nos va a permitir operar con datos de entrada y salida. Brindándole así la posibilidad al usuario de ingresar datos que serán procesados para darnos otros datos de salida.

Orientado a Objetos (POO)

```
1  function fibonacci(n) {
2      var actual, ant1, ant2;
3      if (n === 0) {
4          actual = 0;
5      } else if (n === 1) {
6          actual = 1;
7      } else {
8          ant1 = ant2 = 1;
9          for (i = 2; i < n; i++) {
10             actual = ant1 + ant2;
11             ant2 = ant1;
12             ant1 = actual;
13         }
14     }
```

Función de javascript — Solución imperativa para calcular el N-ésimo término de la sucesión de Fibonacci.

```
1  -module(fibonacci).
2  -export([fibonacci/1]).
3
4  -spec fibonacci(non_neg_integer()) -> non_neg_integer().
5  fibonacci(0) -> 0;
6  fibonacci(1) -> 1;
7  fibonacci(N) -> fibonacci(N - 1) + fibonacci(N - 2).
```

Función de Erlang — Solución declarativa y funcional para calcular el N-ésimo término de la sucesión de Fibonacci.

7.- Es extensible, es decir puede interactuar con otros lenguajes de programación tales como C, java, etc.

8.- Gran cantidad de Librerías y Paquetes adicionales.

9.- Varios IDE's de desarrollo

10.- Bastante documentación

- .
- .
- .
- .
- .



¿En qué podemos utilizar Python?



Use Python for...

>>> More

Web Development: Django , Pyramid , Bottle , Tornado , Flask , web2py

GUI Development: tkinter , PyGObject , PyQt , PySide , Kivy , wxPython

Scientific and Numeric: SciPy , Pandas , IPython

Software Development: Buildbot , Trac , Roundup

System Administration: Ansible , Salt , OpenStack

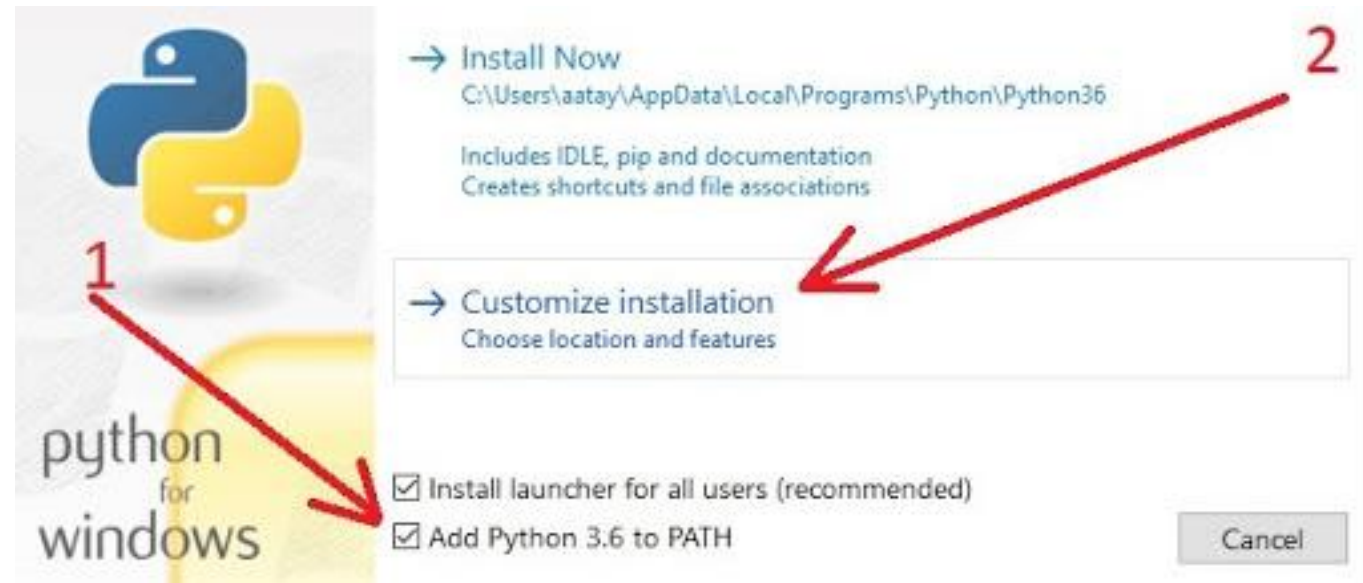
Instalación de Herramientas



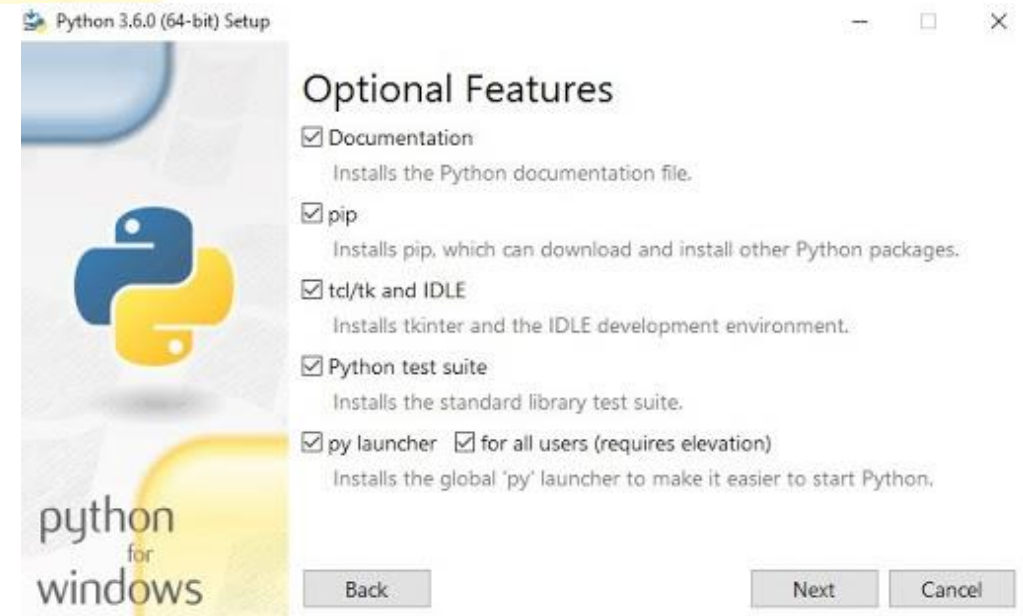
Instalación Python:



<https://www.python.org/downloads/>



<https://www.python.org/downloads/release/python-376/>



Instalación de IDE's

<https://www.jetbrains.com/pycharm/>

<https://www.anaconda.com/distribution/>

The image shows the PyCharm logo, which consists of a black square with 'PC' and a horizontal line, followed by the word 'PyCharm' in a bold, black, sans-serif font. Below the logo, the text 'The Python IDE for Professional Developers' is written in a smaller, black, sans-serif font. At the bottom, there is a black button with the word 'DOWNLOAD' in white, and below that, the text 'Full-fledged Professional or Free Community' in a small, black, sans-serif font. The background features a large, stylized 'X' shape made of green and yellow geometric shapes.

PC PyCharm

The Python IDE
for Professional Developers

DOWNLOAD

Full-fledged Professional or Free Community





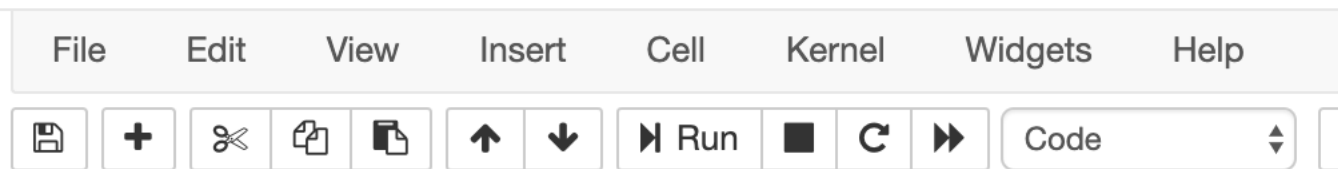
```
pip3 install --upgrade pip
```

```
pip3 install jupyter
```

```
jupyter notebook
```

<http://localhost:8888>

 jupyter Untitled1 (unsaved changes)



In []: |



replit

Uso y sintaxis

```
jp@Juans-MacBook-Pro ~ % python3
Python 3.7.6 (v3.7.6:43364a7ae0, Dec 18 2019, 14:18:50)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Modo interactivo

```
>>> a=1
>>> b=1.0
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
```


Operadores aritméticos

Descripción	Operador	Ejemplo
Suma	+	c=2+3
Resta	-	c=2-3
Multiplicación	*	c=2*3
División	/	c=2/3
Potencia	**	c=2**3
División entera	//	c=2//3
Módulo	%	c=2%3

En cada caso, probar los comandos:

```
print(c)  
type(c)
```

Operadores relacionales

Descripción	Operador	Ejemplo
Igual	==	2==3
Diferente	!=	2!=3
Mayor	>	2>3
Menor	<	2<3
Mayor igual	>=	2>=3
Menor igual	<=	2<=3

En cada caso, asignar el valor a una variable y probar los comandos:

```
print(c)  
type(c)
```

Operadores lógicos

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

```
>>> not True
False
>>> not False
True
```

Verificar los resultados:

```
>>> 3 > 2
>>> 3 < 2
>>> 2 >= 1 + 1
>>> 4 - 2 <= 1
>>> 2 == 1 + 1
>>> 6 / 2 != 3
```

Python permite encadenar varias comparaciones y el resultado será verdadero si y sólo si todas las comparaciones lo son.

```
>>> 4 == 3 + 1 > 2
True
>>> 2 != 1 + 1 > 0
False
```

integer, float, boolean, string, bytes

Base Types

int 783 0 -192 0b010 0o642 0xF3
zero binary octal hexa


float 9.23 0.0 -1.7e-6
 $\times 10^{-6}$

bool True False

str "One\nTwo"
escaped new line

Multiline string:
"""X\tY\tZ
1\t2\t3"""
escaped tab

bytes b"toto\xfe\775"
hexadecimal octal

 *immutables*

Ingreso de datos:

Para introducir una cadena:

```
cadena = input("Introduce una cadena de texto: ")  
print ("La cadena que ingreso es:\n",cadena)
```

Para introducir un int:

```
numero = int(input("Introduce un numero: "))  
print ("El valor que ingreso es:\n",numero)
```

Para introducir un float:

```
numero = float(input("Introduce un numero: "))  
print ("El valor que ingreso es:\n",numero)
```

Escribir un programa que pida al usuario su peso (en kg) y estatura (en metros), calcule el índice de masa corporal y lo almacene en una variable, muestre por pantalla la frase:

"Tu índice de masa corporal es <imc>"

Donde <imc> es el índice de masa corporal calculado redondeado con dos decimales.

Para redondear utilice la función round:

```
>>>x = round(5.76543, 2)  
>>>print(x)  
>>>5.77
```

Strings

El método `count()` retorna el número de veces que se repite un conjunto de caracteres especificado.

```
>>> s = "Hola mundo"
>>> s.count("Hola")
1
```

El método `find()` retorna la ubicación (comenzando desde el cero) en la que se encuentra el argumento indicado.

```
>>> s.find("mundo")
```

```
>>> s = "Hola mundo"
>>> s.startswith("Hola")
True
>>> s.endswith("mundo")
True
>>> s.endswith("world")
```

`rfind()`

Container Types

- **ordered sequences**, fast index access, repeatable values

list [1, 5, 9]

["x", 11, 8.9]

["mot"]

[]

tuple (1, 5, 9)

11, "y", 7.4

("mot",)

()

Non modifiable values (immutables)

☞ expression with only comas → **tuple**

str bytes (ordered sequences of chars / bytes)

" "
b" "

- **key containers**, no *a priori* order, fast key access, each key is unique

dictionary **dict** {"key": "value"}

dict (a=3, b=4, k="v")

{}

(key/value associations) {1: "one", 3: "three", 2: "two", 3.14: "π"}

collection **set** {"key1", "key2"}

{1, 9, 3, 0}

set ()

☞ keys=hashable values (base types, immutables...)

frozenset immutable set

empty


```
numero = 2017  
decimal = 3.14  
booleano = True  
cadena = "Hola Mundo"  
lista = ["a", "b", "c"]  
tupla = (3, "b", True)  
diccionario = {'a':2, 'b':14}
```

```
>>> a=1  
>>> type(a)  
<class 'int'>
```

Cadenas

Subíndices y subcadenas

Dada la cadena:

```
>>> palabra='Python'
```

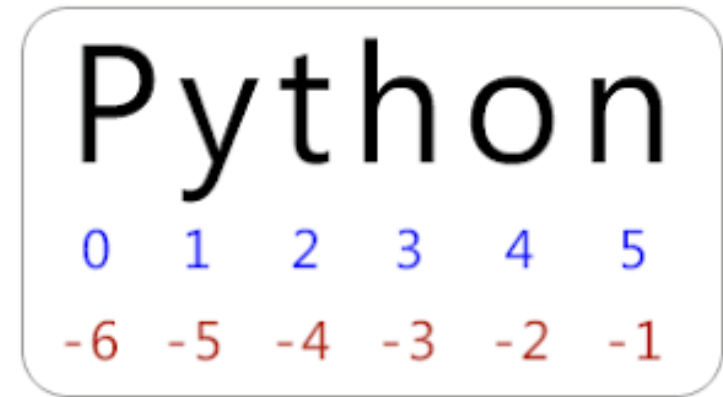
```
>>> palabra[:2] # caracteres desde el principio hasta la posición 2 (excluída)
'Py'
```

```
>>> palabra[4:] # caracteres desde la posición 4 (incluída) hasta el final
'on'
```

```
>>> palabra[-2:] # caracteres desde la ante-última (incluída) hasta el final
'on'
```

Las cadenas de Python no pueden ser modificadas – son inmutables

```
>>> palabra[0] = 'J'
```



Probar los siguientes ejercicios y verificar su salida:

```
>>> cadena.replace("Hola", "Adiós")
>>> cadena = " esta cadena tiene espacios a los lados "
>>> cadena.strip()
>>> cadena.lstrip()
>>> cadena.rstrip()

>>> print(cadena.upper())
>>> print(cadena.lower())
>>> cadena.capitalize()

>>> nombres = "Carlos|Cristina|Rodrigo|Hugo"
>>> nombres.split("|")

>>> caracter = "|"
>>> nombres2 = ["Carlos", "Cristina", "Rodrigo", "Hugo"]
>>> print (caracter.join(nombres2) )
```

Listas

```
>>> cuadrados = [1, 4, 9, 16, 25]
```

```
>>> cuadrados
```

```
>>> cuadrados[-1]
```

```
>>> cuadrados[-3:]
```

A diferencia de las cadenas de texto, que son immutable, las listas son un tipo mutable, es posible cambiar un su contenido:

```
>>> cubos = [1, 8, 27, 65, 125] # algo anda mal
```

```
>>> 4 ** 3 # el cubo de 4 es 64, no 65!
```

```
64
```

```
>>> cubos[3] = 64 # reemplazar el valor incorrecto
```

```
>>> cubos
```

```
[1, 8, 27, 64, 125]
```

También se puede agregar nuevos ítems al final de la lista, usando el método `append()` :

```
>>> cubos.append(216) # agregar el cubo de 6
```

```
>>> cubos.append(7 ** 3) # y el cubo de 7
```

```
>>> cubos
```

```
[1, 8, 27, 64, 125, 216, 343]
```

Es posible anidar listas:

```
>>> a = ['a', 'b', 'c']
```

```
>>> n = [1, 2, 3]
```

```
>>> x = [a, n]
```

```
>>> x [['a', 'b', 'c'], [1, 2, 3]]
```

```
>>> x[0]
```

```
['a', 'b', 'c']
```

```
>>> x[0][1]
```

```
'b'
```

```
>>> a, b = 0, 1
>>> while b < 10:
...     print(b)
...     a, b = b, a+b
...
1
1
2
3
5
8
```

```
>>> if x<0:
...     print("negativo")
... elif x>0:
...     print("positivo")
... else:
...     print("cero")
...
positivo
```

```
>>> palabras = ['gato', 'ventana', 'perro']
>>> for p in palabras:
...     print(p, len(p))
...
gato 4
ventana 7
perro 5
```


Conjuntos

```
>>> canasta = {'manzana', 'naranja', 'manzana', 'pera', 'naranja', 'banana'}  
>>> print (canasta) # muestra que se removieron los duplicados  
{'pera', 'manzana', 'banana', 'naranja'}  
>>> 'naranja' in canasta # verificación de pertenencia rápida  
True  
>>> 'yerba' in canasta  
False
```

Diccionarios

Un diccionario es un conjunto no ordenado de pares clave: valor, con el requerimiento de que las claves sean únicas.

```
>>>tel={'Pedro':4098,'Luis':4139}
>>>tel['Juan']=4127
>>>tel
{'Luis':4139,'Pedro':4098,'Juan':4127}
>>>tel['Pedro']
4098
>>>del tel['Luis']
>>>tel['Lili']=4127
>>>tel
{'Pedro':4098,'Lili':4127,'Juan':4127}
>>>list(tel.keys())
['Lili','Juan','Pedro']
>>>sorted(tel.keys())
['Juan','Lili','Pedro']
>>>'Juan' in tel
True
>>>'Pedro' not in tel
False
```

Funciones

```
>>> def funcion():  
...     return ("hola")  
...  
>>> funcion()  
'hola'
```

```
>>> def mi_funcion(nombre, apellido):  
...     nombre_completo = nombre + apellido  
...     print(nombre_completo)  
...  
>>> mi_funcion("Juan Pablo", " Zaldumbide")  
Juan Pablo Zaldumbide
```

Manejo de Archivos

```
def creatxt():  
    archi=open('datos.txt','w')  
    archi.close()  
def grabartxt():  
    archi=open('datos.txt','a')  
    archi.write('Linea 1\n')  
    archi.write('Linea 2\n')  
    archi.write('Linea 3\n')  
    archi.close()
```

```
creatxt()  
grabartxt()
```

```
def leertxt():  
    archi=open('datos.txt','r')  
    linea=archi.readline()  
    while linea!="":  
        print (linea)  
        linea=archi.readline()  
    archi.close()
```

```
def leertxtenlista():  
    archi=open('datos.txt','r')  
    lineas=archi.readlines()  
    print (lineas)  
    archi.close()
```

Ejercicio guiado

Ejercicio
Frecuencia de una cadena



Ejercicio

El objetivo es contar el número de ocurrencias de las palabras en un archivo de texto.
El resultado final deberá ser similar a:

(harry,250)

(potter,153)

(niño,48)

Realizarlo en las parejas asignadas por el instructor y compartir su solución a
juan.zaldumbide@epn.edu.ec

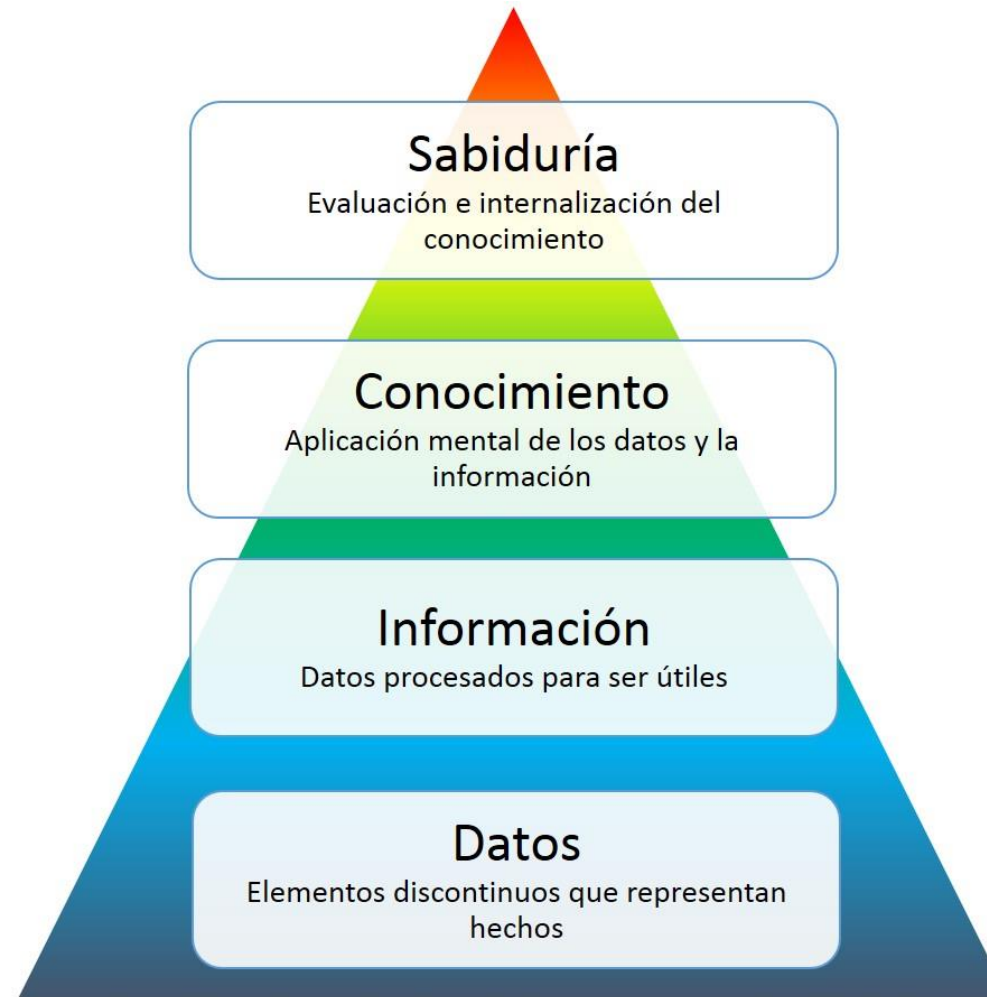
Realizar el mismo ejercicio pero obviar las palabras cortas.



Introducción al análisis de datos

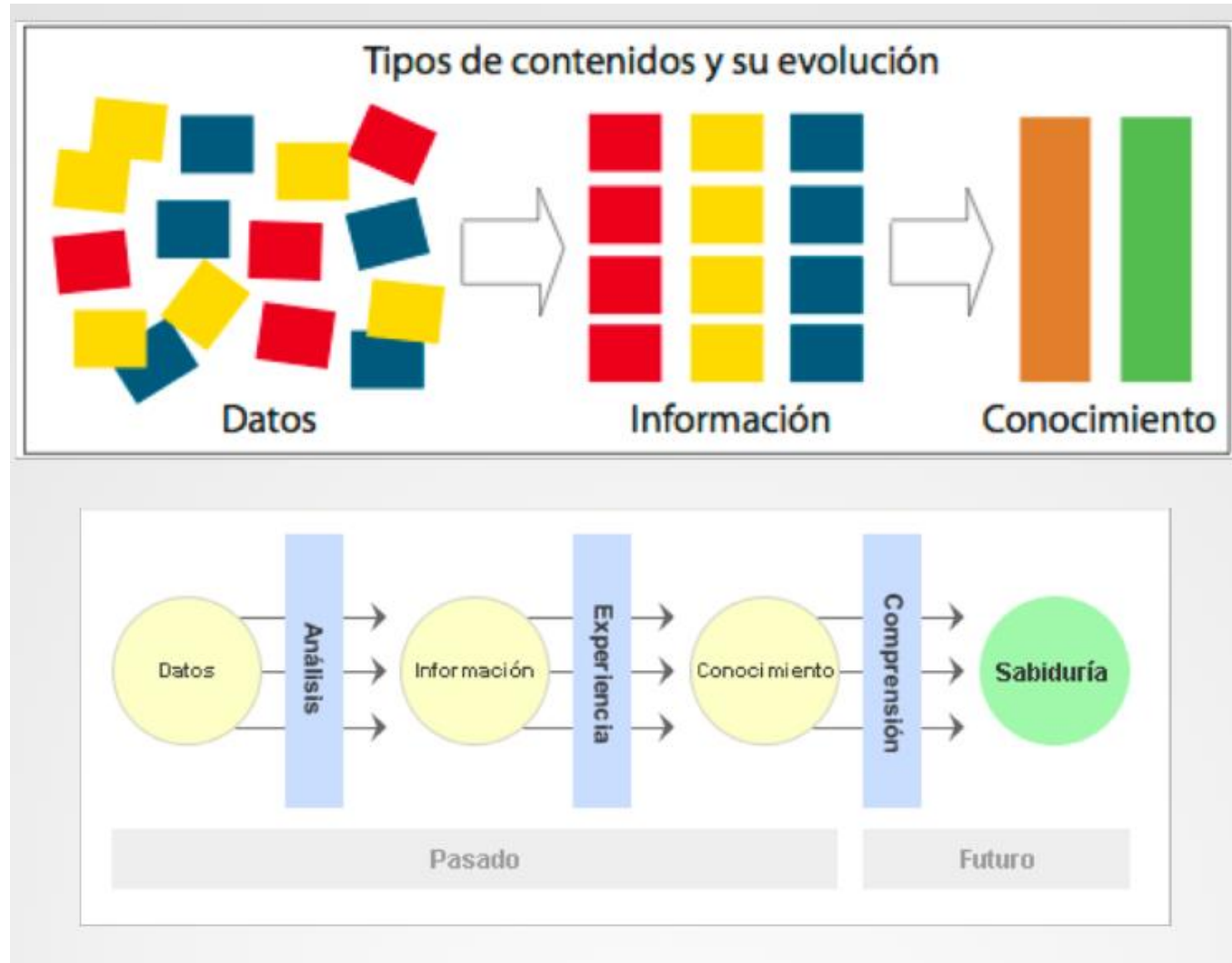


Datos, información, conocimiento y sabiduría



Fuente: Hey, J.: The Data, Information, Knowledge, Wisdom Chaim: The Metaphorical Link

Tipos de contenidos y su evolución



Datos



Información



Conocimiento



Sabiduría



- Operacionales
- Histórico clientes
- Demográficos
- Geográficos
- Corporativos
- Compras
- Fuentes externas
- etc.

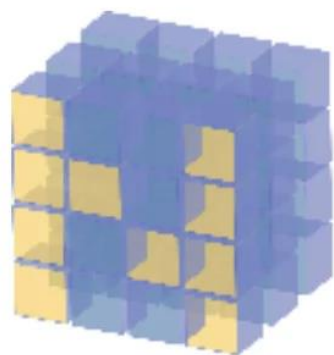
- Patrones
- Tendencias
- Relaciones
- Asociaciones
- Predicciones
- etc.

- Tipología de clientes
- Frecuencia de contactos
- Potencial del cliente.
- Grado de lealtad
- Participación
- Satisfacción
- etc.

- Captación
- Fidelización
- Recuperación
- Segmentación de clientes
- Campañas
- Promociones
- Precios
- etc.



pythonTM



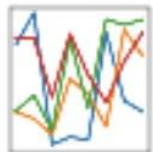
NumPy



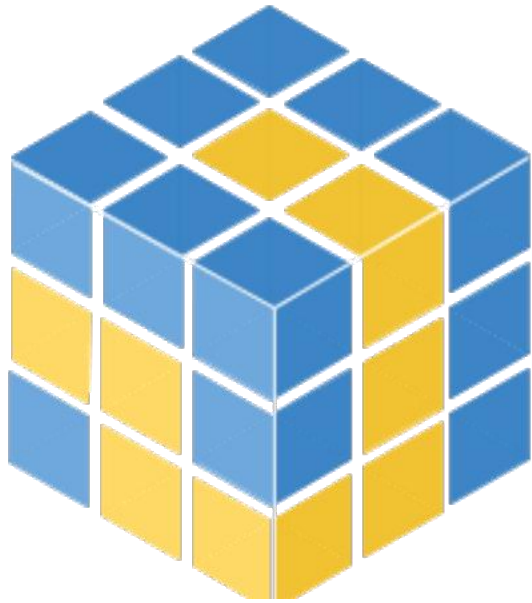
SciPy

pandas

$y_t = \beta'x_t + \mu_t + \epsilon_t$



matplotlib



NumPy

