

Python para Ciencia de los Datos

Introducción:

OBJETIVOS DE APRENDIZAJE

Los objetivos de este curso son comenzar con Python como lenguaje de programación y darle una idea de cómo comenzar a trabajar con datos en Python.

En este curso aprenderás sobre:

- Que es Python y por qué es útil
- La aplicación de Python
- Cómo definir variables
- Conjuntos y declaraciones condicionales en Python
- El propósito de tener funciones en Python
- Cómo operar en archivos para leer y escribir datos en Python
- Cómo usar pandas, un paquete imprescindible para cualquiera que intente realizar análisis de datos en Python.

Programa de estudios

Módulo 1 - Conceptos básicos de Python

- Tu primer programa
- Tipos
- Expresiones y variables
- Operaciones de cadena

Módulo 2 - Estructuras de datos de Python

- Listas y tuplas
- Conjuntos
- Diccionarios

Módulo 3 - Conceptos básicos de programación en Python

- Condiciones y ramificación
- Bucles
- Funciones
- Objetos y clases

Módulo 4 - Trabajar con datos en Python

- Leer archivos con abierto
- Escribir archivos con open
- Cargando datos con Pandas
- Trabajar y guardar datos con Pandas

OBJETIVOS DE APRENDIZAJE

En esta lección aprenderás los conceptos básicos de Python y escribirás tu primer programa en Python:

- Tipos
- Expresiones y variables
- Operaciones de cadena

1) Probemos su primer programa en Python.

Una **declaración** o **expresión** es una **instrucción**. La computadora se ejecutará o ejecutará. Quizás el programa más simple que puedas escribir es una declaración impresa. Cuando ejecuta la declaración de impresión, Python simplemente muestre el valor entre paréntesis.

El **valor entre paréntesis** se llama **argumento**. Si está utilizando un cuaderno de Jupyter en este Por supuesto, verá un pequeño rectángulo con la declaración. A esto se le llama celda. Si selecciona esta celda con su mouse, entonces haga clic en el botón Ejecutar celda. La declaración se ejecutará. El resultado se mostrará debajo la celda. Seguiremos este paradigma en los videos.

Es costumbre comentar su código. Esto les dice a otras personas lo que hace su código. Tú simplemente coloque un símbolo de almohadilla antes de su comentario. Cuando ejecute el código, Python ignorará el comentario. Un error sintáctico es cuando Python no entienda su código.

Por ejemplo, si escribe `print 'frint'`, obtendrá un error mensaje. Un **error semántico** es cuando **su lógica es incorrecta**. Por ejemplo, si ingresa Python 102 en lugar de Python 101, no obtendrá un mensaje de error, pero tu código es incorrecto.

2) Python tipos

Un **tipo** es cómo Python representa diferentes **tipos de datos**. En este video, discutiremos algunos tipos ampliamente utilizados en Python. Puede tener diferentes tipos en Python.

Pueden ser números enteros como 11, números reales como 21.213. Incluso pueden ser palabras. Los enteros, los números reales y las palabras se pueden expresar como diferentes tipos de datos.

El siguiente cuadro resume tres tipos de datos para los últimos ejemplos:

La primera columna indica la expresión. La segunda columna indica el tipo de datos. Podemos **ver el tipo de datos** real en Python usando **el comando type**.

<code>type(11)</code>	<code>int</code>
<code>type(21.213)</code>	<code>float</code>
<code>type("Hello Python 101")</code>	<code>str</code>

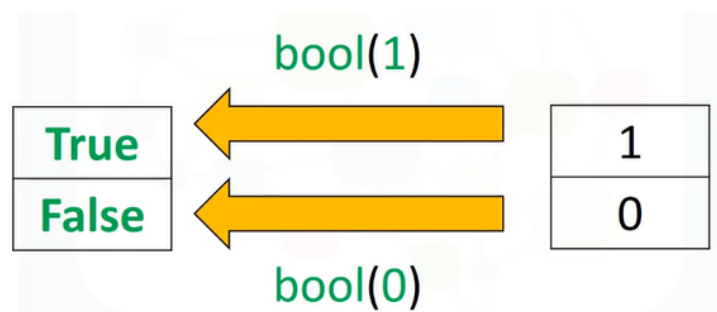
Puedes cambiar el tipo de expresión en Python; esto se llama **fundición tipográfica**. Puede **convertir un int en un flotante**.

Por ejemplo, puede convertir o convertir el número entero 2 en un flotante 2.

Nada cambia realmente. Si lanza un flotador a un entero, debe tener cuidado. Por ejemplo, si lanzas el float 1.1 a 1, perderá información. Si una cadena contiene un valor entero, puede convertirlo a int. Si convertimos una cadena que contiene un número no entero valor, obtenemos un error. Consulte más ejemplos en el laboratorio. Puede convertir un int en una cadena o un flotante en una cadena. Boolean es otro tipo importante en Python. Un booleano puede tomar dos valores.

- El primer valor es verdadero, solo recuerde que usamos una T mayúscula.
- Los valores booleanos también pueden ser falsos, con una F mayúscula.

Usando el comando type en un valor booleano, obtenemos el término bool, esto es la abreviatura de Booleano. Si convertimos un booleano verdadero a un entero o float, obtendremos un 1. Si lanzamos un booleano falso a un entero o float, obtenemos un cero. Si lanza un 1 a un booleano, obtiene un verdadero. Del mismo modo, si lanza un 0 a un booleano, obtiene un falso.



3) expresiones y variables

Las **expresiones** describen un **tipo de operación** que realizan las computadoras.

Las expresiones son **operaciones que realiza Python**. Por ejemplo, operaciones aritméticas básicas como sumando varios números. Llamamos a los números operandos y los símbolos matemáticos operadores.

También podemos usar variables para almacenar valores usando el **operador de asignación**. es decir, el signo igual, usaremos dos puntos para denotar el valor de la variable. También podemos usar el **comando type en variables**.

Es una buena práctica utilizar nombres de variables significativos, para no tener que realizar un seguimiento de qué está haciendo la variable. Supongamos que nos gustaría convertir el número.

Es común para usar el guión bajo para representar el comienzo de una nueva palabra, también puede usar una mayúscula. Llamamos a la variable que contiene el total.

Los valores del resultado final cambian en consecuencia, pero no tenemos que modificar el resto del código.

4) String

Una cadena está contenida entre dos comillas, también puede utilizar comillas simples. Una cadena puede ser espacios o dígitos, puede contener caracteres especiales. Podemos vincular o asignar una cadena a otra. Es útil pensar en una cadena como una secuencia ordenada. Ya que se puede acceder a cada elemento de la secuencia utilizando un índice representado por la matriz de números.

Name= "Michael Jackson"

M	i	c	h	a	e	l		J	a	c	k	s	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Name[0]:M

Name[6] :l

Name[13]:o

Se puede acceder al primer índice de la siguiente manera. Podemos acceder al índice 6. Además, podemos acceder al índice 13. También podemos usar la indexación negativa con cadenas.

Name= "Michael Jackson"

M	i	c	h	a	e	l		J	a	c	k	s	o	n
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Name[-15] :M

Name[-7] :J

Name[-1]: n

El último elemento viene dado por el índice -1. El primer elemento se puede obtener por índice -15 y así sucesivamente, podemos vincular una cadena a otra variable.

Es útil pensar en la cadena como una lista o tupla. Podemos tratar la cadena como una secuencia y realizar operaciones de secuencia. El 2 indica que seleccionamos cada segunda variable. Podemos usar el comando "len" para obtener la longitud de la cuerda. Como hay 15 elementos, el resultado es 15.

Las barras invertidas representan el comienzo de las secuencias de escape. Las secuencias de escape representan cadenas que pueden ser difíciles de ingresar.

Por ejemplo, las barras invertidas '**n**' **representan una nueva línea** de manera similar, la **barra invertida 't' representa una tabulación**. La salida viene dada por una pestaña donde la barra invertida 't' es. Si desea colocar una **barra invertida en su cadena, use una barra invertida doble**.

Utilice el corte(slicing) para encontrar los primeros cuatro elementos de la siguiente cadena:

```
Letters="ABCDEFGHIIJK"  
Letters[0:4]
```

'ABCD'

Utilice un valor de zancada(stride) de dos en la siguiente cadena:

```
Good="GsoAo+d"  
Good[:2]
```

'Good'

Convierta la siguiente cadena a mayúsculas usando el método upper()

```
: "uppercase".upper()  
Number="123456"
```

: 'UPPERCASE'

Método replace()

A='Michael Jackson is the best'

B=A.replace('Michael', 'Janet')

B: 'Janet Jackson is the best'

Método find()

Name= "Michael Jackson"

M	i	c	h	a	e	l		J	a	c	k	s	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Name.find('el'):5

Name.find('Jack'):8

"0123456".find('1')

☐ 0

☒ 1

Listas y Tuplas

En este video, cubriremos listas y tuplas, estos se denominan tipos de datos compuestos y son uno de los tipos clave de estructuras de datos en Python.

Tuplas

Las tuplas son una secuencia ordenada. Aquí hay una tupla "Calificaciones".

Las tuplas se expresan como elementos separados por comas entre paréntesis.

Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)

En Python, hay diferentes tipos: strings, integer, float. Todos pueden estar contenidos en una tupla, pero el tipo de variable es tupla. Se puede acceder a cada elemento de una tupla a través de un índice. La siguiente tabla representa la relación entre el índice y los elementos del tupla.

Tuple1 = ("disco", 10, 1.2)

-3	0	"disco"	Tuple1[-3]: "disco"
-2	1	10	Tuple1[-2]: 10
-1	2	1.2	Tuple1[-1]: 1.2

Se puede acceder al primer elemento por el nombre de la tupla seguido de un corchete con el número de índice, en este caso, cero. Podemos acceder al segundo elemento de la siguiente manera. También podemos acceder al último elemento. En Python, podemos usar un índice negativo.

Podemos concatenar o combinar tuplas agregándolas.

("disco", 10, 1.2)



tuple2 = tuple1 + ("hard rock", 10)

("disco", 10, 1.2, "hard rock", 10)

0	1	2	3	4
---	---	---	---	---

Si quisiéramos varios elementos de una tupla, también podríamos dividir las tuplas. Por ejemplo, si queremos los primeros tres elementos, usamos el siguiente comando.

El último índice es 1 más grande que el índice que desea. Del mismo modo, si queremos los dos últimos elementos, usamos el siguiente comando. Observe cómo el último índice es 1 más grande que la longitud de la tupla.

`("disco", 10, 1.2, "hard rock", 10)`

0	1	2	3	4
---	---	---	---	---

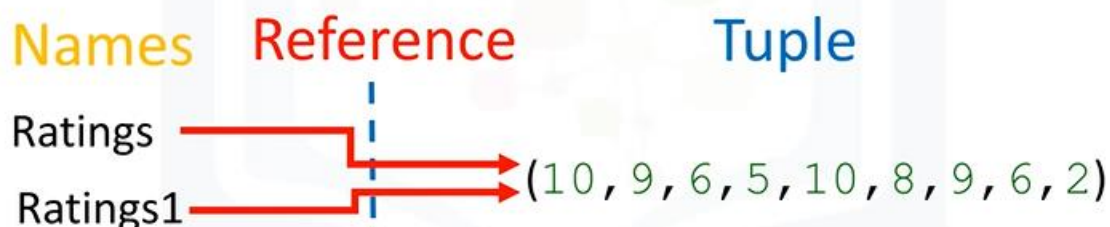
`tuple2[3:5]=>("hard rock", 10)`

Las tuplas son inmutables, lo que significa que no podemos cambiarlas.

Para ver por qué esto es importante, veamos qué sucede cuando establecemos la variable Calificaciones 1 a las calificaciones.

`Ratings =(10, 9, 6, 5, 10, 8, 9, 6, 2)`

`Ratings1=Ratings`



Usamos la imagen para proporcionar una explicación simplificada de lo que está sucediendo. Cada variable no contiene una tupla, pero hace referencia al mismo objeto de tupla inmutable. Consulte el módulo "objetos y clases" para obtener más información sobre los objetos.

Digamos que queremos cambiar el elemento en el índice 2. Debido a que las tuplas son inmutables, no podemos. Por lo tanto, las calificaciones 1 no se verán afectadas por un cambio en la calificación porque la tupla es inmutable, es decir, no podemos cambiarlo.

Podemos asignar una tupla diferente a la variable Calificaciones. La variable Calificaciones ahora hace referencia a otra tupla. Como consecuencia de la inmutabilidad, si queremos manipular una tupla, debemos crear en su lugar, una nueva tupla. Por ejemplo, si queremos ordenar una tupla, usamos la función **sorted**.

`Ratings =(10, 9, 6, 5, 10, 8, 9, 6, 2)`

`RatingsSorted=sorted(Ratings)`

`(10, 9, 6, 5, 10, 8, 9, 6, 2)`

La entrada es la tupla original. La salida es una nueva tupla ordenada.

Una tupla puede contener otras tuplas, así como otros tipos de datos complejos; esto se llama anidamiento.

$NT = (1, 2, ("pop", "rock"), (3,4), ("disco", (1,2)))$

0	1	2	3	4
---	---	---	---	---

$NT[2]: ("pop", "rock") [1] = "rock"$ \rightarrow $NT[2] [1] = "rock"$

0	1
---	---

Podemos acceder a estos elementos utilizando los métodos de indexación estándar. Si seleccionamos un índice con una tupla, se aplica la misma convención de índice. Como tal, podemos acceder a los valores de la tupla. Por ejemplo, podríamos acceder al segundo elemento.

Podemos visualizar este anidamiento como un árbol. La tupla tiene los siguientes índices. Si consideramos índices con otras tuplas, vemos que la tupla en el índice 2 contiene una tupla con dos elementos. Podemos acceder a esos dos índices.

$NT = (1, 2, ("pop", "rock"), (3,4), ("disco", (1,2)))$

0	1	2	3	4
---	---	---	---	---

$NT[2]$

$NT[3]$

$NT[4]$

$("pop", "rock")$

$(3,4)$

$("disco", (1,2))$

"pop"	"rock"
-------	--------

3	4
---	---

"disco"	(1,2)
---------	-------

$NT[2][0]$

$NT[2][1]$

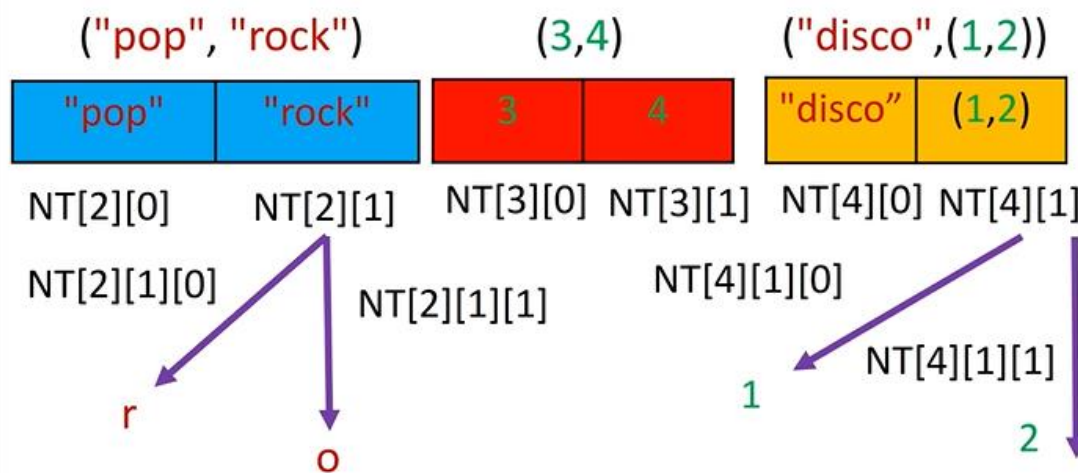
$NT[3][0]$

$NT[3][1]$

$NT[4][0]$

$NT[4][1]$

La misma convención se aplica al índice 3. También podemos acceder a los elementos de esas tuplas. Podemos continuar el proceso. Incluso podemos acceder a niveles más profundos del árbol agregando otro corchete. Podemos acceder a diferentes caracteres en la cadena o varios elementos en la segunda tupla contenido en el primero.



Listas

Las listas también son una estructura de datos popular en Python. Las listas también son una secuencia ordenada. A continuación, se muestra una lista L. Una lista se representa con corchetes. En muchos aspectos, las listas son como tuplas, una diferencia clave es que son mutables. Las listas pueden contener cadenas, flotantes, enteros.

`L = ["Michael Jackson", 10.1, 1982]`

Podemos anidar otras listas. También anidamos tuplas y otras estructuras de datos; se aplican las mismas convenciones de indexación para el anidamiento. Al igual que las tuplas, se puede acceder a cada elemento de una lista a través de un índice. La siguiente tabla representa la relación entre el índice y los elementos de la lista.

`L = ["Michael Jackson", 10.1, 1982]`

-3	0	"Michael Jackson"	L[-3]: "Michael Jackson"
-2	1	10.1	L[-2]: 10
-1	2	1982	L[-1]: 1982

Se puede acceder al primer elemento por el nombre de la lista seguido de un corchete con el número de índice, en este caso cero. Podemos acceder al segundo elemento de la siguiente manera. También podemos acceder al último elemento. En Python, podemos usar un índice negativo.

```
L = ["Michael Jackson", 10.1, 1982, "MJ", 1]
```

0	1	2	3	4
---	---	---	---	---

```
L[3:5]: ["MJ", 1]
```

También podemos realizar cortes en listas. Observe cómo el último índice es 1 más grande que la longitud de la lista. Las convenciones de índice para listas y tuplas son idénticas.

```
L = ["Michael Jackson", 10.1, 1982]
```

```
L1 = L + ["pop", 10]
```

```
L1 = ["Michael Jackson", 10.1, 1982, "pop", 10]
```

0	1	2	3	4
---	---	---	---	---

Podemos concatenar o combinar listas agregándolas. La nueva lista tiene los siguientes índices. Las listas son mutables, por lo tanto, podemos cambiarlas. Por ejemplo, aplicamos el método ampliar(extend) agregando un "punto" seguido del nombre del método, luego paréntesis. El argumento dentro del paréntesis es una nueva lista que vamos a concatenar a la lista original.

```
L = ["Michael Jackson", 10.1, 1982]
```

```
L.extend(["pop", 10])
```

```
L = ["Michael Jackson", 10.1, 1982, "pop", 10]
```

0	1	2	3	4
---	---	---	---	---

En este caso, en lugar de crear una nueva lista, L1, la lista original L se modifica agregando dos nuevos elementos.

```
L = ["Michael Jackson", 10.1, 1982]
```

```
L.extend(["pop", 10])
```

```
["Michael Jackson", 10.1, 1982, "pop", 10]
```

```
L.append("A")
```

```
["Michael Jackson", 10.1, 1982, "pop", 10, "A"]
```

Otro método similar es adjuntar(append). Si aplicamos agregar en lugar de extendido, agregamos un elemento a la lista. Si miramos el índice, solo hay un elemento más. El índice 3 contiene la lista que adjuntamos. Cada vez que aplicamos un método, las listas cambian. Si aplicamos extender, agregamos dos nuevos elementos a la lista. La lista L se modifica agregando dos nuevos elementos. Si agregamos la cadena "A", cambiamos aún más la lista agregando la cadena "A". Como las listas son mutables, podemos cambiarlas.

```
A=["disco", 10, 1.2]
```

```
A[0]="hard rock"
```

```
A=["hard rock", 10, 1.2]
```

Por ejemplo, podemos cambiar el primer elemento de la siguiente manera. La lista ahora se convierte en: "hard rock", 10, 1.2. Podemos borrar un elemento de una lista usando el comando "del"; simplemente indicamos la lista elemento que nos gustaría eliminar como argumento.

```
A=["hard rock", 10, 1.2]
```


`del(A[1])`


A: ["hard rock", 1.2]

Por ejemplo, si quisiéramos eliminar el primer elemento, el resultado será 10,1.2. Podemos eliminar el segundo elemento. Esta operación elimina el segundo elemento de la lista. Podemos convertir una cadena en una lista usando split.

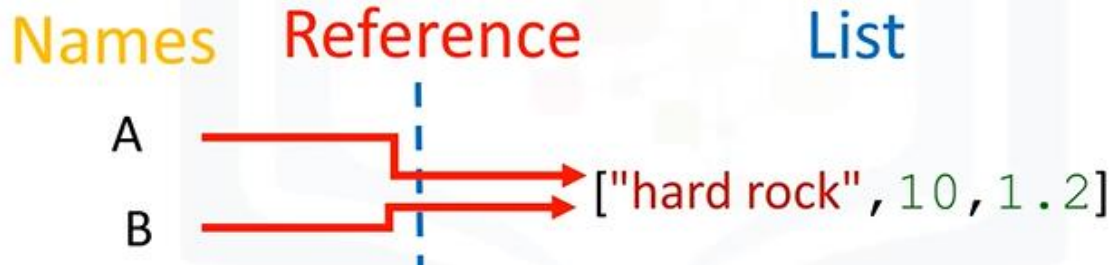
```
"A,B,C,D".split(",")
```

```
["A", "B", "C", "D"]
```

Por ejemplo, el método split convierte cada grupo de caracteres separados por un espacio en un elemento de una lista. Podemos usar la función de división para separar cadenas en un carácter específico, conocido como un delimitador. Simplemente pasamos el delimitador en el que nos gustaría dividir como argumento, en este caso un coma.

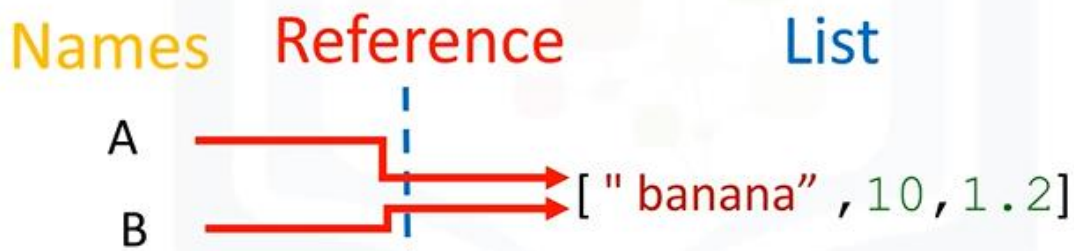
```
A=["hard rock", 10, 1.2]
```

```
B=A
```



El resultado es una lista, cada elemento corresponde a un conjunto de caracteres que se han separado por una coma. Cuando establecemos una variable, B igual a A, tanto A como B hacen referencia a la misma lista.

```
B[0]="hard rock"
A[0]="banana" ➡ B[0]: "banana"
```



Los nombres múltiples que hacen referencia al mismo objeto se conocen como aliasing. Sabemos por la última diapositiva que el primer elemento en B está configurado como roca dura. Si cambiamos el primer elemento en "A" a "banana" obtenemos un efecto secundario; el valor de B cambiará como consecuencia. "A" y "B" hacen referencia a la misma lista, por lo tanto, si cambiamos "A", la lista "B" también cambia. Si marcamos el primer elemento de B después de cambiar la lista "A" obtenemos banana en lugar de hard Roca.

```
A=["hard rock", 10, 1.2]
```

```
B=A[:]
```



Puede clonar la lista "A" utilizando la siguiente sintaxis. La variable "A" hace referencia a una lista. La variable "B" hace referencia a una nueva copia o clon de la lista original.

```
A=["hard rock", 10, 1.2]
A[0]= " banana "
```



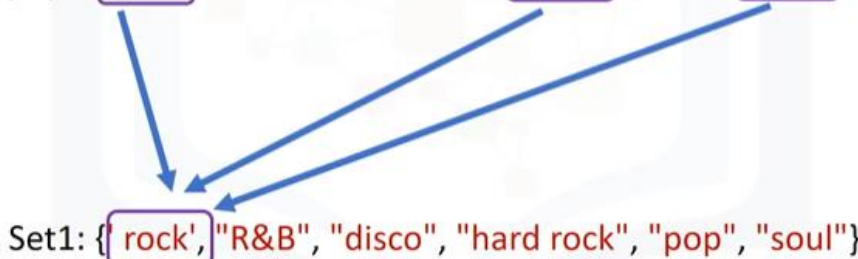
```
B[0]: " hard rock "
```

Ahora, si cambia "A", "B" no cambiará. Podemos obtener más información sobre listas, tuplas y muchos otros objetos en Python usando la ayuda mando. Simplemente pase la lista, la tupla o cualquier otro objeto de Python.

Conjuntos:

Los conjuntos son un tipo de colección. Esto significa que, al igual que las listas y tuplas, puede ingresar diferentes tipos de Python. A diferencia de las listas y tuplas, no están ordenadas. Esto significa que los conjuntos no registran la posición del elemento.

```
Set1={"pop", "rock", "soul", "hard rock", "rock", "R&B", "rock", "disco" }
```



```
Set1: {'rock', 'R&B', 'disco', 'hard rock', 'pop', 'soul'}
```

Los conjuntos solo tienen elementos únicos. Esto significa que solo hay uno de un elemento en particular en un conjunto. Para definir un conjunto, utilice llaves. Cuando se crea el conjunto real, los elementos duplicados no estarán presentes.

```
album_list = ["Michael Jackson", "Thriller", "Thriller", 1982]
```

```
album_set = set(album_list)
```

```
album_set : {'Michael Jackson', 'Thriller', 1982}
```

set()

album_set

Puede convertir una lista en un conjunto utilizando el conjunto de funciones; esto se llama **fundición tipográfica**. Simplemente use la lista como entrada para el conjunto de funciones. El resultado será una lista convertida en un conjunto.


```
A = {"Thriller", "Back in Black", "AC/DC"}  
A.add("NSYNC")  
A:{"AC/DC", "Back in Black", "NSYNC", "Thriller"}
```



"NSYNC"
"Thriller", "Back in Black", "AC/DC"

Considere el conjunto "A". Representemos este conjunto con un círculo. Si está familiarizado con los conjuntos, esto puede ser parte de un diagrama de Venn. Un diagrama de Venn es una herramienta que generalmente usa formas para representar conjuntos.

Podemos agregar un elemento a un conjunto usando el método agregar(add). Simplemente ponemos el nombre del conjunto seguido de un punto, luego el método add.

El argumento es el nuevo elemento del conjunto que nos gustaría agregar, en este caso, "NSYNC". El conjunto "A" ahora tiene "NSYNC" como elemento. Si agregamos el mismo elemento dos veces, nada sucede ya que no puede haber duplicados en un conjunto. Supongamos que nos gustaría eliminar NSYNC del conjunto A.

```
A:{"AC/DC", "Back in Black", "NSYNC", "Thriller"}  
A.remove("NSYNC")  
A:{"AC/DC", "Back in Black", "Thriller"}
```



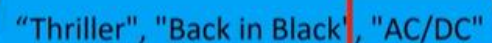
"Thriller", "Back in Black", "AC/DC"

También podemos eliminar un elemento de un conjunto mediante el método de eliminación(remove). Simplemente colocamos el nombre del conjunto seguido de un punto y luego el método de eliminación. El argumento es el elemento del conjunto que nos gustaría eliminar, en este caso, "NSYNC".

```
A:{"AC/DC", "Back in Black", "Thriller"}
```

```
"AC/DC" in A
```

```
True
```



"Thriller", "Back in Black", "AC/DC"

Puede utilizar este método para cualquier elemento del conjunto. Podemos verificar si un elemento está en el conjunto usando el comando "in" de la siguiente manera. El comando

comprueba si el elemento, en este caso, "AC / DC" está en el set. Si el elemento está en el conjunto, devuelve verdadero. Si buscamos un elemento que no está en el conjunto, en este caso el elemento "Quién", como el artículo no está en el conjunto, obtendremos un falso.

```
album_set_1 = {"AC/DC", "Back in Black", "Thriller"}  
album_set_2 = {"AC/DC", "Back in Black", "The Dark Side of the Moon"}  
album_set_3 = album_set_1 & album_set_2  
album_set_3: {"AC/DC", "Back in Black"}
```

La intersección de dos conjuntos es un nuevo conjunto que contiene elementos que están en ambos conjuntos. Es útil usar diagramas de Venn. Los dos círculos que representan los conjuntos se combinan; la superposición representa el nuevo conjunto. Como la superposición está compuesta por el círculo rojo y el círculo azul, definimos la intersección en términos de "y". En Python, usamos el **ampersand(&)** para encontrar el unión de dos conjuntos.

El resultado es un nuevo álbum de conjunto, el conjunto 3, que contiene todos los elementos tanto del conjunto de álbumes 1 como del álbum. conjunto 2.

```
album_set_1.union(album_set_2)  
{'AC/DC', 'Back in Black', 'The Dark Side of the Moon', 'Thriller'}
```

La unión de dos conjuntos es el nuevo conjunto de elementos que contienen todos los elementos de ambos conjuntos. Podemos encontrar la unión de los conjuntos de álbumes. 1 y conjunto de álbumes 2 de la siguiente manera. El resultado es un nuevo conjunto que tiene todos los elementos. del conjunto de álbumes 1 y conjunto de álbumes 2.



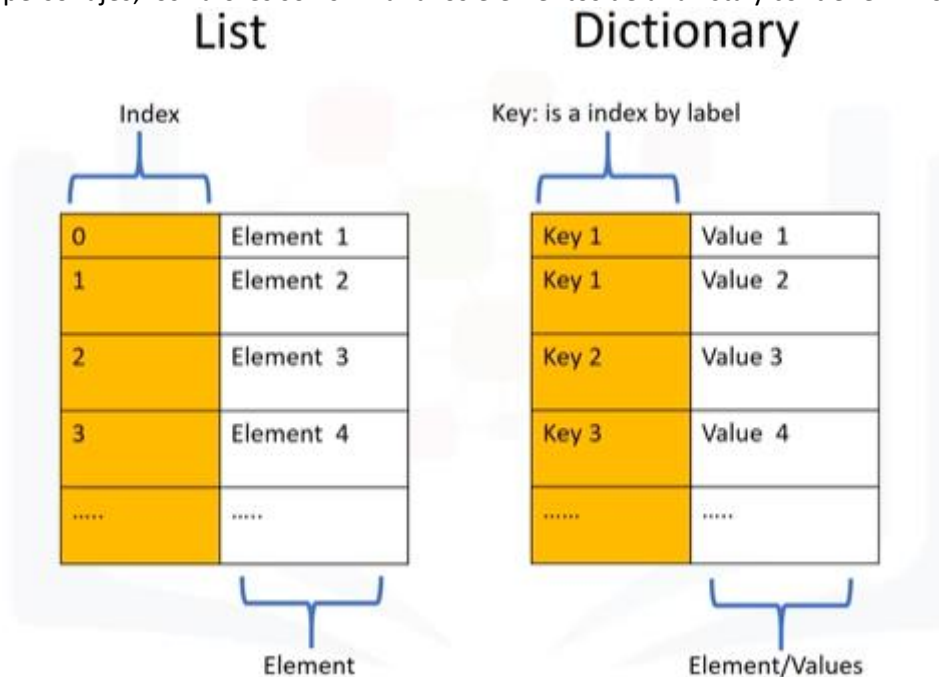
Considere el nuevo conjunto de álbumes conjunto de álbumes 3; el conjunto contiene los elementos "AC / DC" y "Back in Black". Podemos representar esto con un diagrama de Venn, ya que todos los elementos del conjunto de álbumes 3 están en el conjunto de álbumes 1. El

círculo que representa el conjunto de álbumes 1 encapsula el círculo que representa el conjunto de álbumes 3.

Podemos comprobar si un conjunto es un subconjunto usando el método `is subset`. Como el conjunto de álbumes 3 es un subconjunto del conjunto de álbumes 1, el resultado es verdadero. Hay mucho más que puedes hacer con los sets.

Diccionarios

Los diccionarios son un tipo de colección en Python. Un diccionario tiene claves y valores. La clave es análoga al índice, son como direcciones, pero no tienen que ser enteros. Suelen ser personajes; los valores son similar a los elementos de una lista y contienen información.



Para crear un diccionario, usamos llaves. Las claves son los primeros elementos; ellos deben sea inmutable y único. Cada tecla va seguida de un valor separado por dos puntos. Los valores pueden ser inmutables, mutables y duplicados. Cada par de clave y valor está separado por una coma.

```
{ "key1":1, "key2 ":"2","key3" :[3,3,3], "key4":(4,4,4), ('key5'):5 }  
[ "Thriller": 1982, "Back in Black": 1980, "The Dark Side of the Moon": 1973, "The Bodyguard": 1992 ]
```

Key		
"Thriller"	"1982"	
"Back in Black"	"1980"	
"The Dark Side of the Moon"	"1973"	
"The Bodyguard"	"1992"	
"Bat Out of Hell"	"1977"	DICT["Bat Out of Hell"]:"1977"
"Their Greatest..."	"1976"	
Saturday Night Fever	"1977"	
"Rumors"	"1977"	
		Value

La clave se usa para buscar el valor. Usamos corchetes; el argumento es la clave. Esto genera el valor. Usando la tecla de "Back in Black", esto devuelve el valor de 1980. La clave "El lado oscuro de la luna" nos da el valor de 1973. El uso de la clave "El guardaespaldas" nos da el valor 1992, y así sucesivamente.

"Thriller"	"1982"	
"Back in Black"	"1980"	
"The Dark Side of the Moon"	"1973"	
"The Bodyguard"	"1992"	
"Bat Out of Hell"	"1977"	
"Their Greatest..."	"1976"	
Saturday Night Fever	"1977"	
"Rumors"	"1977"	
'Graduation'	"2007"	DICT['Graduation']='2007'

Podemos agregar una nueva entrada al diccionario de la siguiente manera. Esto agregará el valor 2007 con la nueva clave llamada "Graduación".

"Back in Black"	"1980"	
"The Dark Side of the Moon"	"1973"	
"The Bodyguard"	"1992"	
"Bat Out of Hell"	"1977"	
"Their Greatest..."	"1976"	
Saturday Night Fever	"1977"	
"Rumors"	"1977"	

del(DICT['Thriller'])

Podemos eliminar una entrada de la siguiente manera. Esto elimina la clave "Thriller" y su valor. Podemos verificar si un elemento está en el diccionario.

"Thriller"	"1982"
"Back in Black"	"1980"
"The Dark Side of the Moon"	"1973"
"The Bodyguard"	"1992"
"Bat Out of Hell"	"1977"
"Their Greatest..."	"1976"
Saturday Night Fever	"1977"
"Rumors"	"1977"

'The Bodyguard' in DICT

True

usando el comando **in** de la siguiente manera. El comando comprueba las claves. Si estan en el diccionario, devuelven un verdadero. Si intentamos con otra clave que no está en el diccionario, obtenemos un falso.

"Thriller"	"1982"
"Back in Black"	"1980"
"The Dark Side of the Moon"	"1973"
"The Bodyguard"	"1992"
"Bat Out of Hell"	"1977"
"Their Greatest..."	"1976"
"Saturday Night Fever"	"1977"
"Rumors"	"1977"

```
DICT.keys()=[ "Thriller", "Back in Black", "The Dark Side of the Moon", "The Bodyguard",
              "Bat Out of Hell", "Their Greatest...","Saturday Night Fever", "Rumors" ]
```

Para ver todas las claves en un diccionario, podemos usar las claves de método para obtener las claves. La salida es una lista como un objeto con todas las llaves.

"Thriller"	"1982"
"Back in Black"	"1980"
"The Dark Side of the Moon"	"1973"
"The Bodyguard"	"1992"
"Bat Out of Hell"	"1977"
"Their Greatest..."	"1976"
Saturday Night Fever	"1977"
"Rumors"	"1977"

```
DICT.values()=[ "1982","1980","1973","1992", "1977","1976" "1977", "1977" ]
```

De la misma forma, podemos obtener los valores utilizando los valores del método. Consulte los laboratorios para obtener más ejemplos e información en diccionarios.

Condiciones y Ramificación:

Condiciones

Las operaciones de comparación comparan algún valor u operando, luego, en función de alguna condición, producir un booleano. Digamos que asignamos "a" valor de "a" a 6. Podemos usar el operador de igualdad denotado con dos signos iguales para determinar si dos valores son iguales, en este caso, si siete es igual a 6. En este caso, como seis no es igual a 7, el resultado es falso.

a=6

a==6

True

a=6

a==7

False

Si realizáramos una prueba de igualdad para el valor 6, los dos valores serían iguales. Como resultado, obtendríamos un verdadero. Considere el siguiente operador de comparación de igualdad.

i=2

i>=5

False



False

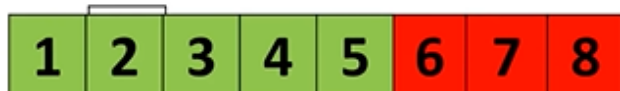
Si establecemos el valor de "i" igual a 5, el operando producirá un verdadero. Si establecemos el valor de "i" en 2, obtendríamos un falso porque 2 es menor que 5. Podemos cambiar la desigualdad, si el valor del operando izquierdo, en este caso, "i" es menor que el valor del operando derecho, en este caso, 6, la condición se convierte en verdadera.

Si el valor de "i" se establece en 2, el resultado es verdadero, ya que 2 es menor que 6. La prueba de desigualdad usa una marca de explicación que precede al signo igual, si dos operandos

i=2

i<6

True



Si establecemos "i" igual a 2, el operador es verdadero ya que 2 no es igual a 6.

"AC/DC"=="Michael Jackson"

False

"AC/DC"!="Michael Jackson"

True

Al comparar "AC / DC" y "Michael Jackson" mediante la prueba de igualdad, obtenemos un falso, Usando la prueba de desigualdad, obtenemos un verdadero, ya que las cadenas son diferentes.

Ramificación

La ramificación nos permite ejecutar diferentes declaraciones para una entrada diferente. Es útil pensar en una "declaración si" como una habitación cerrada:

- Si la afirmación es verdadera, puede ingresar a la sala y su programa puede ejecutar alguna tarea.
- Si la declaración es falsa, su programa omitirá la tarea.

Por ejemplo, considere el rectángulo azul que representa un concierto de ACDC.

- Si la persona tiene 18 años o más, puede ingresar al concierto de ACDC.
- Si son menores de 18 años, no pueden ingresar al concierto.

El individuo procede al concierto, su edad es de 17 años, por lo que no se les concede acceso al concierto, y deben seguir adelante.

Si el individuo tiene 19 años, la condición es verdadera, puede ingresar al concierto; entonces ellos pueden seguir adelante.

Tenemos el enunciado if Tenemos la expresión que puede ser verdadera o falso, los corchetes no son necesarios sino los dos puntos. Dentro de una sangría, tenemos la expresión que se ejecuta si la condición es verdadera.

Para el caso en el que la edad sea 17, establecemos el valor de la variable edad en 17. Comprobamos la declaración if; la declaración es falsa, por lo tanto, el programa no se ejecutará

la declaración para imprimir "entrará".

En este caso, simplemente imprimirá "seguir adelante".

Para el caso en el que la edad sea 19, establecemos el valor de la variable edad en 19.

Comprobamos la declaración if.

La declaración es verdadera, por lo tanto, el programa ejecutará la declaración para imprimir "usted entrara."

Entonces simplemente imprimirá "seguir adelante".

La "instrucción else" ejecutará un bloque de código diferente, si la misma condición es falsa. Usemos de nuevo la analogía del concierto de ACDC; si el usuario tiene 17 años, no puede ir al ACDC concierto, pero pueden ir al concierto de Meat Loaf representado por el cuadrado morado. Si el individuo tiene 19 años, la condición es verdadera. Pueden ingresar al concierto de ACDC, luego pueden seguir adelante como antes. La sintaxis de la instrucción else es similar. Simplemente agregamos la declaración else. Luego agregamos la expresión que nos gustaría ejecutar con una sangría. Para el caso en el que la edad sea 17, establecemos el valor de la variable edad en 17. Comprobamos la declaración if. La declaración es falsa, por lo tanto, avanzamos a la declaración else. Ejecutamos la declaración en la sangría. Corresponde a la persona que asiste al concierto de Meat Loaf. El programa continuará ejecutándose. Para el caso en el que la edad sea 19, establecemos el valor de la variable edad en 19. Comprobamos la declaración if. La declaración es verdadera, por lo tanto, el programa ejecutará la declaración para imprimir "usted entrara."

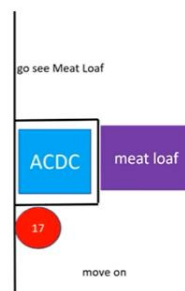
El programa omite las expresiones en la instrucción else y continúa ejecutando el resto de las expresiones. La declaración elif, abreviatura de "else if", nos permite verificar condiciones adicionales, si la condición de procedimiento es falsa. Si la condición es verdadera, se ejecutarán las expresiones alternativas. Considere el ejemplo del concierto, si la persona tiene 18 años, irá al concierto de Pink Floyd, en lugar de asistir al concierto de ACDC o Meat Loaf. La persona mayor de 18 años ingresa al área por no ser mayor de 19 años. No pueden ver ACDC, pero como tienen 18 años, asisten a Pink Floyd. Después de ver a Pink Floyd, siguen adelante. La sintaxis de la "instrucción elseif" es similar. Simplemente agregamos la declaración elseif con la condición. Luego agregamos la expresión que nos gustaría ejecutar si la declaración es verdadera, con un sangrar. Ilustremos el código de la izquierda. Entra un joven de 18 años. No tienen más de 18 años, por lo tanto, la condición es falsa, por lo que la condición de la instrucción else if está marcada. La condición es verdadera, entonces imprimimos "ve a ver a Pink Floyd". Luego seguiríamos adelante, como antes. Si la edad variable fuera 17, se imprimirá la declaración "ir a ver Meat Loaf". De manera similar, si la edad es mayor de 18 años, se imprimirá la declaración "puede ingresar". Consulte los laboratorios para ver más ejemplos. Ahora echemos un vistazo a los operadores lógicos. Las operaciones lógicas toman valores booleanos y producen diferentes valores booleanos. La primera operación es el operador no. Si la entrada es verdadera, el resultado es falso. De manera similar, si la entrada es falsa, el resultado es verdadero. Deje que A y B representen variables booleanas. El operador "o" toma los dos valores y produce un nuevo valor booleano. Podemos usar esta tabla para representar los diferentes valores.

La primera columna representa los posibles valores de A.
 La segunda columna representa los posibles valores de B.
 La última columna representa el resultado de aplicar la operación "o".
 Vemos que el "operador o" solo produce un falso si todos los valores booleanos son falsos.
 Se imprimirán las siguientes líneas de código: "Este álbum se hizo en los años 70 o 90"
 si el año variable del álbum no cae en los 80.
 Veamos qué sucede cuando establecemos el año del álbum en 1990.
 La recta numérica de color es verde cuando la condición es verdadera y roja cuando la condición es verdadera.
 Es falso.
 En este caso, la condición es verdadera.
 Examinando la segunda condición, vemos que 1990 es mayor que 1989, por lo que la condición también es cierto.
 Podemos verificar examinando la segunda recta numérica correspondiente.
 En la recta numérica final, la región verde indica dónde el área es verdadera.
 Esta región corresponde a donde al menos una afirmación es verdadera.
 Vemos que 1990 cae en la zona.
 Por lo tanto, ejecutamos la declaración.
 Dejemos que A y B representen variables booleanas que el "operador y" toma en los dos valores,
 y produce un nuevo valor booleano.
 Podemos usar esta tabla para representar los diferentes valores.
 La primera columna representa los posibles valores de A.
 La segunda columna representa los posibles valores de B.
 La columna final representa el resultado de aplicar la operación "y".
 Vemos que el "operador o" solo produce un verdadero si todos los valores booleanos son verdaderos.
 Se imprimirán las siguientes líneas de código: "Este álbum se hizo en los años 80" si el año del álbum variable es entre 1980 y 1989.
 Veamos qué sucede cuando establecemos el año del álbum en 1983.
 Como antes, podemos usar la recta numérica coloreada para examinar dónde se cumple la condición.
 En este caso, 1983 es mayor que 1980, por lo que la condición es verdadera.
 Examinando la segunda condición, vemos que 1990 es mayor que 1983 por lo que esta condición también es cierto.
 Podemos verificar examinando la segunda recta numérica correspondiente.
 En la recta numérica final, la región verde indica dónde el área es verdadera.
 De manera similar, esta región corresponde a donde ambas declaraciones son verdaderas.
 Vemos que 1983 cae en la zona.
 Por lo tanto, ejecutamos la declaración.
 La ramificación nos permite ejecutar diferentes declaraciones para diferentes entradas.

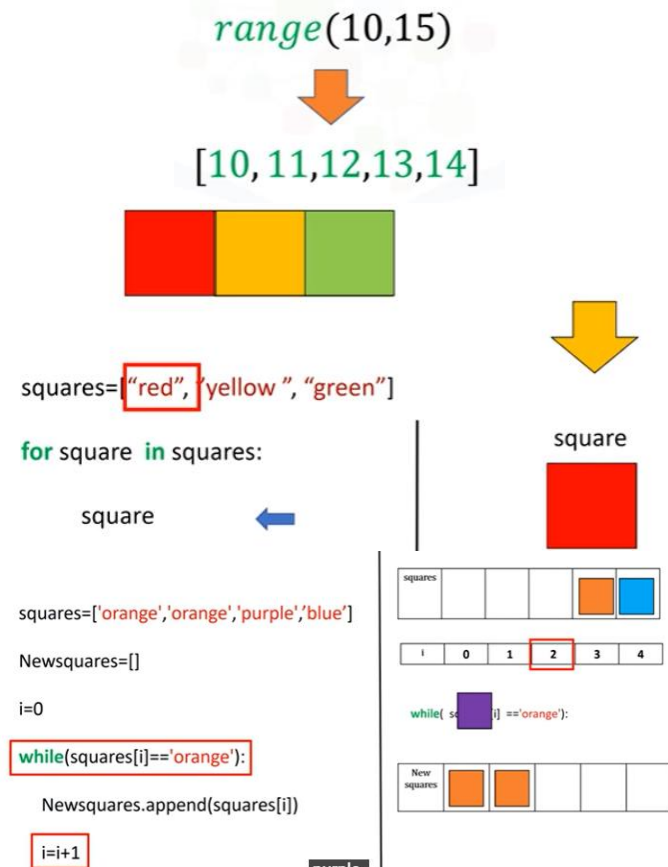
```

age=17
if (age>18):           False
    print("you can enter")
else:
    print("go see Meat Loaf")
print("move on")

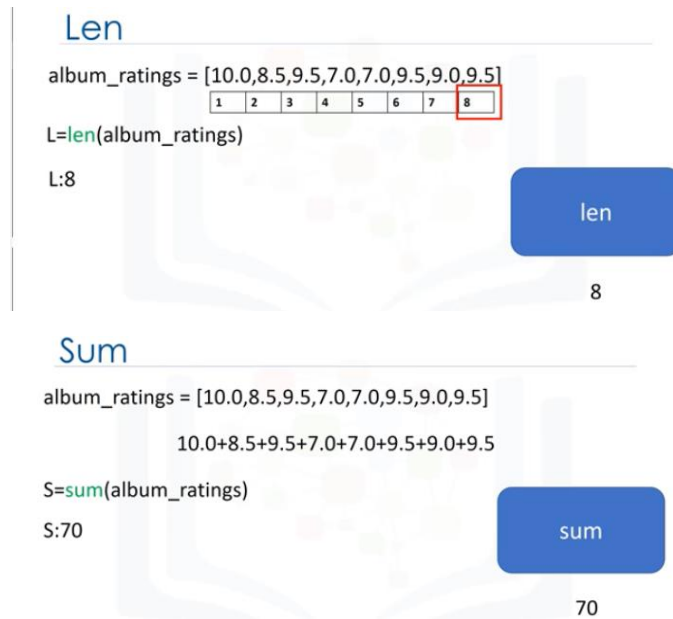
```



Bucles:



Funciones



Sorted vs Sort

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
sorted_album_rating = sorted(album_ratings)
```

```
sorted_album_rating:
```

```
[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]
```

```
album_ratings :
```

```
[10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

sorted

[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]

Sorted vs Sort

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
album_ratings.sort()
```

```
album_rating:
```

```
[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]
```

album_ratings

```
def Mult(a,b):  
    c=a*b  
    return c
```



2*"Michael Jackson "

"Michael Jackson Michael Jackson "

```
Mult(2, "Michael Jackson ")
```

"Michael Jackson Michael Jackson "

```
def printStuff(Stuff):
```

```
    for i,s in enumerate(Stuff):
```

```
        print("Album", i, "Rating is ", s)
```

Stuff: [10.0, 8.5, 9.5]

Index: [0] [1] [2]

```
album_ratings = [10.0,8.5,9.5]
```

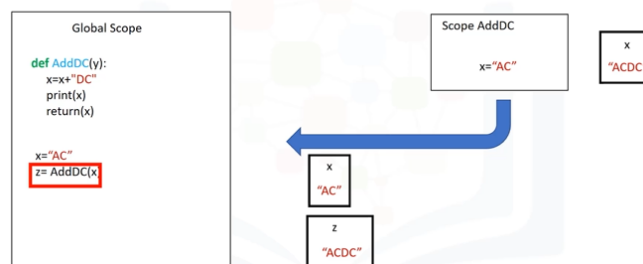
```
printstuff(album_ratings)
```

Album 0 Rating is 10

Album 1 Rating is 8.5

Album 2 Rating is 9.5

Scope



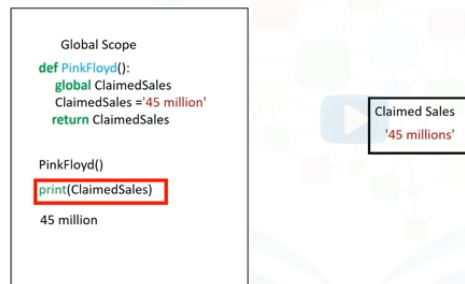
Scope: Local Variables



Scope: Variables



Scope: Local Variables



Objetos y Clases

Built-in Types in Python

- Python has lots of data types

- Types:

- int: 1, 2, 567...
- float: 1.2, 0.62...
- String: 'a', 'abc', 'The cat is yellow'
- List: [1, 2, 'abc']
- Dictionary: {"dog": 1, "Cat": 2}
- Bool: False, True

- Each is an **Object**

Objects: Type

- You can find the type of an object by using the command `type()`

```
>>>type([1, 34, 3])  
<class 'list'>
```

Instance of type `list`

List

```
>>>type("The cat is yellow")  
<class 'str'>
```

Instance of type `str`

str

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

Instance of type `int`

int

```
>>>type({"dog": 1, "Cat": 2})  
<class 'dict'>
```

Instance of type `dict`

dict

Ratings= [2, 5, 6, 6, 8, 9, 9, 10, 10]

[2, 5, 6, 6, 8, 9, 9, 10, 10]

.reverse()

Method

Ratings= [10, 10, 9, 9, 8, 6, 6, 5, 2]

Name of Class

class Circle (object):

Class Definition

Class parent

Create a class: Circle

```
class Circle(object):  
    def __init__(self, radius, color):  
        self.radius = radius;  
        self.color = color;
```

Define your class

Data attributes used to Initialize each instance of the class

special method or constructor used to initialize data attributes parameters

```
def __init__(self, radius, color):  
    self.radius = radius;  
    self.color = color;
```

The self parameter

```
class Rectangle(object):  
    def __init__(self, color, height, width):  
        self.height = height;  
        self.width = width;  
        self.color = color;
```

Define your class

Initialize the object's Data attributes

How to create an object of class circle:

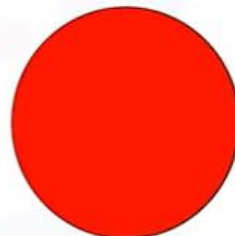
Name of Class

```
RedCircle = Circle(10, "red")
```

Attributes

```
C1=Circle(10,'red')
```

```
class Circle(object):  
    def __init__(self, 10, 'red'):  
        self.radius = 10;  
        self.color = 'red';
```



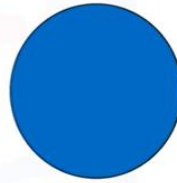
```
self.radius = 10;  
self.color = 'red';
```

Create an Instance of a Class: Circle

```
C1=Circle(10, "red")
```

```
C1.color="blue"
```

```
C1.color  
"blue"
```



```
self.radius = 10;  
self.color = 'red';
```

```
C1=Circle(2, 'red' )
```

```
C1.add_radius(8)
```

```
self .radius = 2  
self. color = 'red'
```

```
def add_radius(self,8):  
    self.radius= self.radius +8  
    return (self.radius)
```

```
def add_radius(self,8):  
    self.radius= 2 +8  
    return (10)
```

Name of object

```
dir (Nameofobject ):
```



```
['_class_',  
 '_delattr_',  
 '_dict_',  
 '_doc_',  
 '_format_',  
 '_getattr_',  
 '_hash_',  
 '_init_',  
 '_module_',  
 '_new_',  
 '_reduce_',  
 '_reduce_ex_',  
 '_repr_',  
 '_setattr_',  
 '_sizeof_',  
 '_str_',  
 '_subclasshook_',  
 '_weakref_']
```

-La función DIR permite obtener la lista de atributos de datos y métodos asociados

Read Open File

```
File1 = open("/resources/data/Example2.txt", "w")
```

Mode

r: para lectura; w: Para escritura; a: Para agregar

```
with open("Example1.txt","r") as File1:
```

```
    file_stuff=File1.read()
```

```
    print(file_stuff)
```

```
print(File1.closed)
```

```
print(file_stuff)
```

Usando with para abrir(Buena practica)

```
with open("Example1.txt","r") as File1:
```

```
    file_stuff[0]
```

```
    file_stuff=File1.readlines()
```

```
    print(file_stuff)
```

```
    file_stuff[1]
```

```
    file_stuff[2]
```

```
file_stuff: ['This is line 1 \n', 'This is line 2 \n', 'This is line 3']
```

```
with open("Example1.txt","r") as File1:
```

```
    file_stuff=File1.readline ()
```

```
    print(file_stuff)
```

```
    file_stuff=File1.readline ()
```

```
    print(file_stuff)
```

This is line 1

This is line 2

```
with open("Example1.txt","r") as File1:
```

```
    for line in File1:
```

```
        print(line)
```

This is line 1

This is line 2

This is line 3

`with open("Example1.txt", "r") as File1:`

```
file_stuff=File1.readlines(16)
print(file_stuff)
file_stuff=File1.readlines(5)
print(file_stuff)
file_stuff=File1.readlines(9)
print(file_stuff)
```

This is line 1

This

is line 2

This is line 1

This is line 2

This is line 3

`Lines=["This is line A\n", "This is line B\n", "This is line C\n"]`

`with open("/resources/data/Example2.txt", "w") as File1:`

`for line in Lines:`



`File1.write(line)`

This is line A
This is line B
This is line C

Example2.txt

`with open("/resources/data/Example2.txt", "w") as File1:`

`File1.write("This is line A\n")`

`File1.write("This is line B\n")`

This is line A
This is line B

Example2.txt

```
with open("Example1.txt", "r") as readfile :
```

```
    with open("Example3.txt", "w") as writefile:
```

```
        for line in readfiles:
```

```
            writefile.write(line)
```



This is line A
This is line B
This is line C

Example1.txt

This is line A
This is line B
This is line C

Example3.txt



```
import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-o
filename = 'Example1.txt'
urllib.request.urlretrieve(url, filename)
```

```
# Print the path of file
```

```
file1.name
```

```
'Example1.txt'
```

```
file1.mode
```

```
'r'
```

```
FileContent = file1.read()
FileContent
```

```
'This is line 1 \nThis is line 2\nThis is line 3'
```

```
# Read one Line
```

```
with open(example1, "r") as file1:
    print("first line: " + file1.readline())
```

```
first line: This is line 1
```

```
with open(example1,"r") as file1:
    i = 0;
    for line in file1:
        print("Iteration", str(i), ": ", line)
        i = i + 1
```

Iteration 0 : This is line 1

Iteration 1 : This is line 2

Iteration 2 : This is line 3

```
xlsx_path='file1.xlsx'
df= pd.read_excel (xlsx_path)
df.head()
```

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0

Dataframes

```
songs = {'Album': ['Thriller', 'Back in Black', 'The Dark Side of the Moon', 'The Bodyguard', 'Bat Out of Hell'],
'Released': [1982, 1980, 1973, 1992, 1977],
'Length': ['00:42:19', '00:42:11', '00:42:49', '00:57:44', '00:46:33']}
```

	Album	Length	Released
0	Thriller	00:42:19	1982
1	Back in Black	00:42:11	1980
2	The Dark Side of the Moon	00:42:49	1973
3	The Bodyguard	00:57:44	1992
4	Bat Out of Hell	00:46:33	1977

x=df[['Length']]



	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5

X

	Length
0	0:42:19
1	0:42:11
2	0:42:49
3	0:57:44
4	0:46:33
5	0:43:08
6	1:15:54
7	0:40:01

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5

df.ix[0, 'Artist']:'Michael Jackson'

df.ix[1, 'Artist']:'AC/DC'

df.ix[0, 'Released']:'1982'

df.ix[1, 'Released']:'1980'

z=df.ix[0:2, 'Artist':'Released']

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5

Z

	Artist	Album	Released
0	Michael Jackson	Thriller	1982
1	AC/DC	Back in Black	1980
2	Pink Floyd	The Dark Side of the Moon	1973

df['Released'].unique()

	Released
0	1982
1	1980
2	1973
3	1992
4	1977
5	1976
6	1977

1982
1980
1973
1992
1977
1976

`df['Released']>=1980`

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5



0	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False

`df1.to_csv('new_songs.csv')`

API:

```
import pandas as pd
```

```
dict_ = {'a':[11, 21, 31], 'b':[12, 22, 32]}
```

```
df = pd.DataFrame(dict_)
```

```
df.head()
```

```
  a  b
0 11 12
1 21 22
2 31 32
```

```
df.mean()
```

```
 a    21.0
 b    22.0
dtype: float64
```

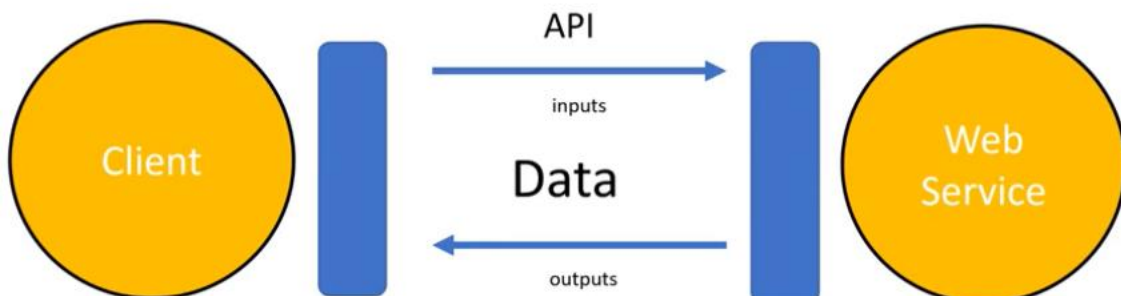
df

Your
Python
Program

Pandas

API-REST:

HTTP
Response
JSON: like Dictionary



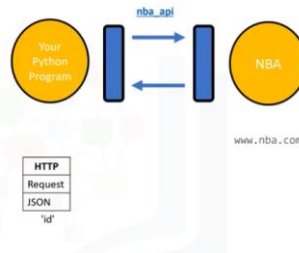
Nba_api:

```

from nba_api.stats.static import teams
nba_teams = teams.get_teams()
nba_teams[:5]

[{'id': 1610612737, 'full_name': 'Atlanta Hawks',
  'abbreviation': 'ATL', 'nickname': 'Hawks', 'city':
  'Atlanta', 'state': 'Atlanta', 'year_founded': 1949},
 {'id': 1610612738, 'full_name': 'Boston Celtics',
  'abbreviation': 'BOS', 'nickname': 'Celtics', 'city':
  'Boston', 'state': 'Massachusetts', 'year_founded':
  1946},
 {'id': 1610612739, 'full_name': 'Cleveland Cavaliers',
  'abbreviation': 'CLE', 'nickname': 'Cavaliers', 'city':
  'Cleveland', 'state': 'Ohio', 'year_founded': 1970},
 {'id': 1610612740, 'full_name': 'New Orleans Pelicans',
  'abbreviation': 'NOP', 'nickname': 'Pelicans', 'city': 'New
  Orleans', 'state': 'Louisiana', 'year_founded': 2002}, ...]

```



El ID de la clave del diccionario tiene un identificador único para cada equipo como valor.

```

def one_dict(list_dict):
    keys=list_dict[0].keys()
    out_dict={key:[] for key in keys}
    for dict_ in list_dict:
        for key, value in dict_.items():
            out_dict[key].append(value)
    return out_dict

dict_nba_team=one_dict(nba_teams)
df_teams=pd.DataFrame(dict_nba_team)
df_teams.head()

```

	id	full_name	abbreviation	nickname	city	state	year_founded
0	1610612737	Atlanta Hawks	ATL	Hawks	Atlanta	Atlanta	1949
1	1610612738	Boston Celtics	BOS	Celtics	Boston	Massachusetts	1946
2	1610612739	Cleveland Cavaliers	CLE	Cavaliers	Cleveland	Ohio	1970
3	1610612740	New Orleans Pelicans	NOP	Pelicans	New Orleans	Louisiana	2002
4	1610612741	Chicago Bulls	CHI	Bulls	Chicago	Illinois	1966

```

df_warriors=df_teams[df_teams['nickname']=='Warriors']
df_warriors

```

	id	full_name	abbreviation	nickname	city	state	year_founded
7	1610612744	Golden State Warriors	GSW	Warriors	Golden State	California	1946

```
id_warriors=df_warriors[['id']].values[0][0]
```

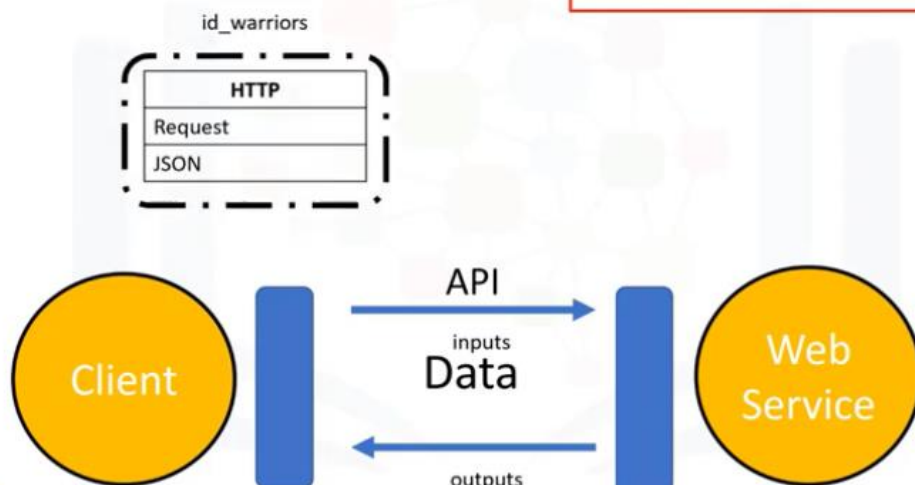
```
id_warriors:1610612744
```

-Ejemplo de adquirir data y crear un dataframe

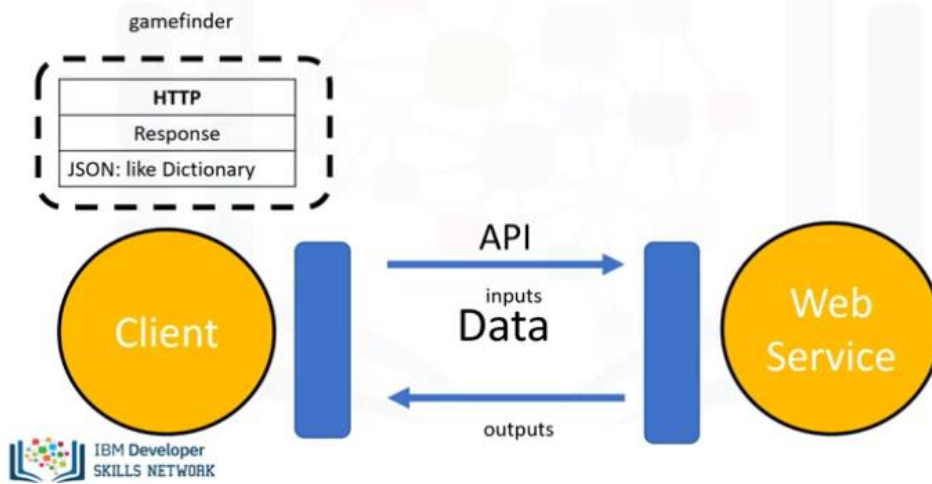
```

from nba_api.stats.endpoints import leaguegamefinder
gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=id_warriors)

```



```
from nba_api.stats.endpoints import leaguegamefinder
gamefinder = leaguegamefinder.LeagueGameFinder(team_id_nullable=id_warriors)
```




```
games = gamefinder.get_data_frames()[0]
games.head()
```

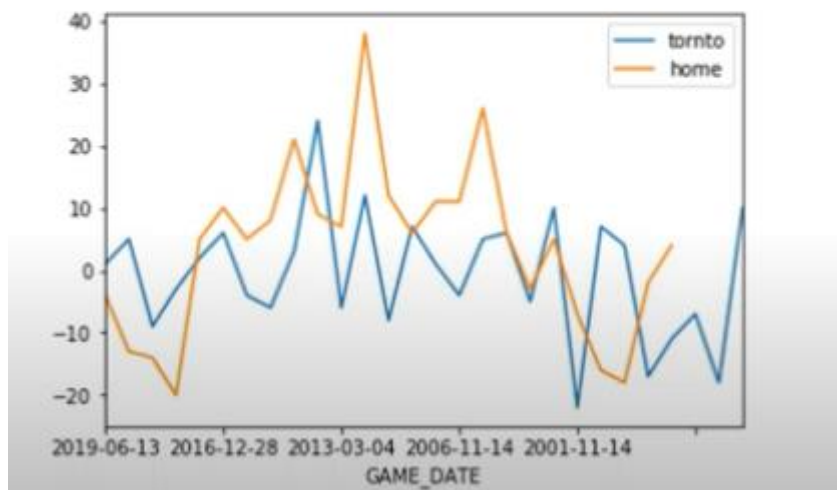
	SEASON_ID	TEAM_ID	TEAM_ABBREVIATION	TEAM_NAME	GAME_ID	GAME_DATE	MATCHUP	WL	MIN	PTS	...	FT_PCT	OREB	DREB	REB	AST	STL	BLK	TOV	PF	PLUS_MINUS
0	42018	1610612744	GSW	Golden State Warriors	0041800406	2019-06-13	GSW vs. TOR	L	240	110	...	0.700	11.0	31.0	42.0	28	9.0	6	16.0	23	-4.0
1	42018	1610612744	GSW	Golden State Warriors	0041800405	2019-06-10	GSW @ TOR	W	240	106	...	0.714	6.0	31.0	37.0	27	5.0	7	15.0	22	1.0
2	42018	1610612744	GSW	Golden State Warriors	0041800404	2019-06-07	GSW vs. TOR	L	241	92	...	0.667	8.0	34.0	42.0	26	6.0	6	17.0	23	-13.0
3	42018	1610612744	GSW	Golden State Warriors	0041800403	2019-06-05	GSW vs. TOR	L	241	109	...	0.833	13.0	28.0	41.0	25	8.0	3	14.0	21	-14.0
4	42018	1610612744	GSW	Golden State Warriors	0041800402	2019-06-02	GSW @ TOR	W	240	109	...	0.870	6.0	36.0	42.0	34	7.0	5	15.0	26	5.0

5 rows x 26 columns

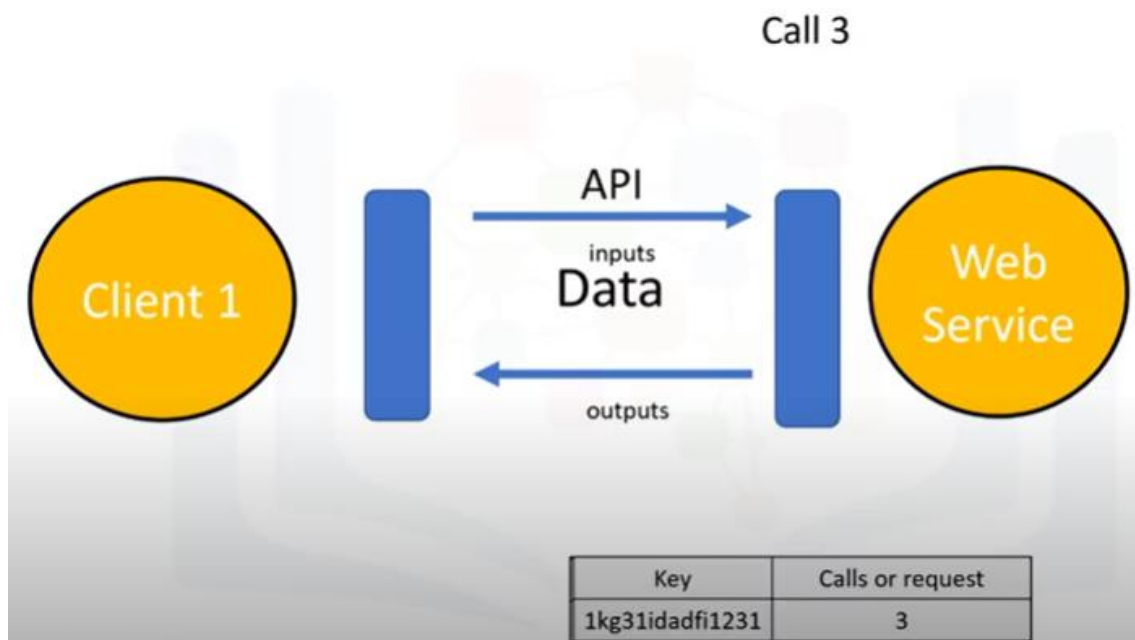
```
games_home=games[games['MATCHUP']=='GSW vs. TOR']
games_away=games[games['MATCHUP']=='GSW @ TOR']
```

```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots()
games_away.plot(x='GAME_DATE',y='PLUS_MINUS', ax=ax)
games_home.plot(x='GAME_DATE',y='PLUS_MINUS', ax=ax)
ax.legend(["away", "home"])
plt.show()
```



Api Key



Ejemplo de servicio Watson IBM

```
from ibm_watson import SpeechToTextV1
```

```
url_s2t="https://stream.watsonplatform.net/speech-to-text/api"
```

```
iam_apikey_s2t="EOeiZxxxDxV2xxxxxxxxxxxxxxxxxxjYen9SKARKW-"
```

```
s2t = SpeechToTextV1(iam_apikey=iam_apikey_s2t, url=url_s2t)
```



```
filename='hello_this_is_python.wav'
```

```
with open(filename, mode="rb") as wav:
```

```
response = s2t.recognize(audio=wav, content_type='audio/wav')
```

response
get_result
:
get_status_code



response.result

```
{'results': [{'alternatives': [{'confidence': 0.91, 'transcript': 'hello this is  
python'}], 'final': True}], 'result_index': 0}  
recognized_text=response.result['results'][0]['alternatives'][0]['transcript']
```

```
recognized_text:  
'hello this is python '
```



Ejemplo Language translator watsonm ibm

```
from ibm_watson import LanguageTranslatorV3  
url_it = 'https://gateway.watsonplatform.net/language-translator/api'  
apikey_it = 'dU2SaxxxxxxxxxxxxxxasdfCuasdf'  
version_it = '2018-05-01'  
language_translator = LanguageTranslatorV3(iam_apikey=apikey_it, url=url_it, version=version_it)  
language_translator.list_identifiable_languages().get_result()  
  
{'languages': [{'language': 'af', 'name': 'Afrikaans'}, {'language': 'ar', 'name': 'Arabic'}, {'language': 'az', 'name':  
'Azerbaijani'}, {'language': 'ba', 'name': 'Bashkir'}, {'language': 'be', 'name': 'Belarusian'}, {'language': 'bg', 'name':  
'Bulgarian'}, {'language': 'bn', 'name': 'Bengali'}, {'language': 'bs', 'name': 'Bosnian'}, {'language': 'ca', 'name':  
'Catalan'}, {'language': 'cs', 'name': 'Czech'}, {'language': 'cv', 'name': 'Chuvash'}, {'language': 'da', 'name': 'Danish'},  
{'language': 'de', 'name': 'German'}, {'language': 'el', 'name': 'Greek'}, {'language': 'en', 'name': 'English'}, {'language':  
'eo', 'name': 'Esperanto'}, {'language': 'es', 'name': 'Spanish'},
```

```
recognized_text = 'hello this is python '  
translation_response = language_translator.translate(text=recognized_text, model_id='en-es')  
translation = translation_response.get_result()  
  
translation  
{'translations': [{'translation': 'Hola, esta es la pitón. '}],  
'word_count': 4, 'character_count': 21}  
spanish_translation = translation['translations'][0]['translation']  
spanish_translation  
'Hola, esta es la pitón. '
```

```
translation_new = language_translator.translate(text=spanish_translation  
model_id='es-en').get_result()
```

```
translation_eng=translation_new['translations'][0]['translation']  
translation_eng  
'Hey, this is the python. '
```

```
French_translation=language_translator.translate(text= translation_eng,  
model_id='en-fr').get_result()  
French_translation['translations'][0]['translation']  
"Hé, c'est le python. "
```

Numpy

```
import numpy as np
a=np.array( [0, 1, 2, 3, 4] )
```

```
a=np.array( [0, 1, 2, 3, 4] )
```

1	2	3	4	5
---	---	---	---	---

```
a:array([0, 1, 2, 3, 4])
```

```
a.size :5
```

```
type(a): numpy.ndarray
```

```
a.ndim: 1
```

```
a.dtype: dtype('int64')
```

```
a.shape: (5,)
```

```
b=np.array([3.1, 11.02, 6.2, 213.2, 5.2])
```

```
type(b): numpy.ndarray
```

```
b.dtype: dtype('float64')
```

```
c=np.array([20, 1, 2, 3, 4])
```

```
c:array([20, 1, 2, 3, 4])
```

```
c:array([100, 1, 2, 3, 0])
```

0	1	2	3	4
---	---	---	---	---

```
c[0]=100
```

```
c:array([100, 1, 2, 3, 4])
```

```
c[3:5]=300,400
```

```
c[4]=0
```

```
c:array([100, 1, 2, 3, 0])
```

```
c:array([100, 1, 2, 300, 400])
```

```
u=np.array([1,0])
```

```
v=np.array([0,1])
```

```
z=u+v
```

```
z:array([1, 1])
```

```
u=[1, 0]
```

```
v=[0, 1]
```

```
z=[ ]
```

```
for n, m in zip(u,v):
```

```
z.append(n+m)
```

```
y=np.array([1,2])
```

```
z=2*y
```

```
z:array([2,4])
```

```
y=[1, 2]
```

```
z=[ ]
```

```
for n in y:
```

```
z.append(2*n)
```

```

np.pi
x=np.array([ 0 , np.pi/2, np.pi ] )
y=np.sin(x)
y:array([ 0,1, 1.2e-16])

```

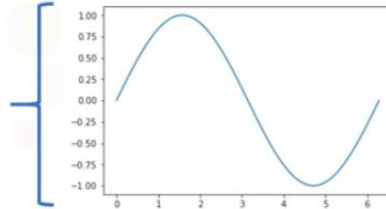
π
 $x = [0, \frac{\pi}{2}, \pi]$
 $y = [\sin(0), \sin(\frac{\pi}{2}), \sin(\pi)]$
 $y = [0,1,0]$

```

x=np.linspace( 0 , 2*np.pi,100)
y=np.sin(x)

import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(x,y)

```



$a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$

$A = \text{np.array}(a)$

A: $\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$

A.ndim:2

A.shape: (3,3)

A.size : 9

$[[11, 12, 13], [21, 22, 23], [31, 32, 33]]$

3x3=9

3 $\left\{ \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix} \right.$

A: $[A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]]$

$\left\{ \begin{bmatrix} A[0][0] & A[0][1] & A[0][2] \\ A[1][0] & A[1][1] & A[1][2] \\ A[2][0] & A[2][1] & A[2][2] \end{bmatrix} \right.$

$A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$

$A[0:2,2]:\text{array}([13, 23])$

	0	1	2
0	11	12	13
1	21	22	23
2	31	32	33

```
A=np.array([[0,1,1],[1,0,1]])  
B=np.array([[1,1],[1,1],[-1,1]])  
C=np.dot(A,B);  
C:array([[0,0],  
         [0,2]])
```