

Técnicas para el manejo de datos en Python



Katherine Morales



- PhD candidate in TSP - Institut Polytechnique de París, Francia.
- Máster Matemática Aplicada - Ciencia de Datos.
- Ingeniera Matemática - Estadística e Investigación Operativa
- Analista de datos
- Consultora
- Capacitadora SEE



GitHub

LinkedIn





Python

Accesible, fácil y se puede usar en varios entornos.

Grandes empresas como Google, Instagram, Pinterest, Facebook, Netflix o Dropbox siguen utilizando Python en su desarrollo tecnológico.

Big Data, AI, Data Science, frameworks de pruebas y desarrollo web

Language Ranking: IEEE Spectrum

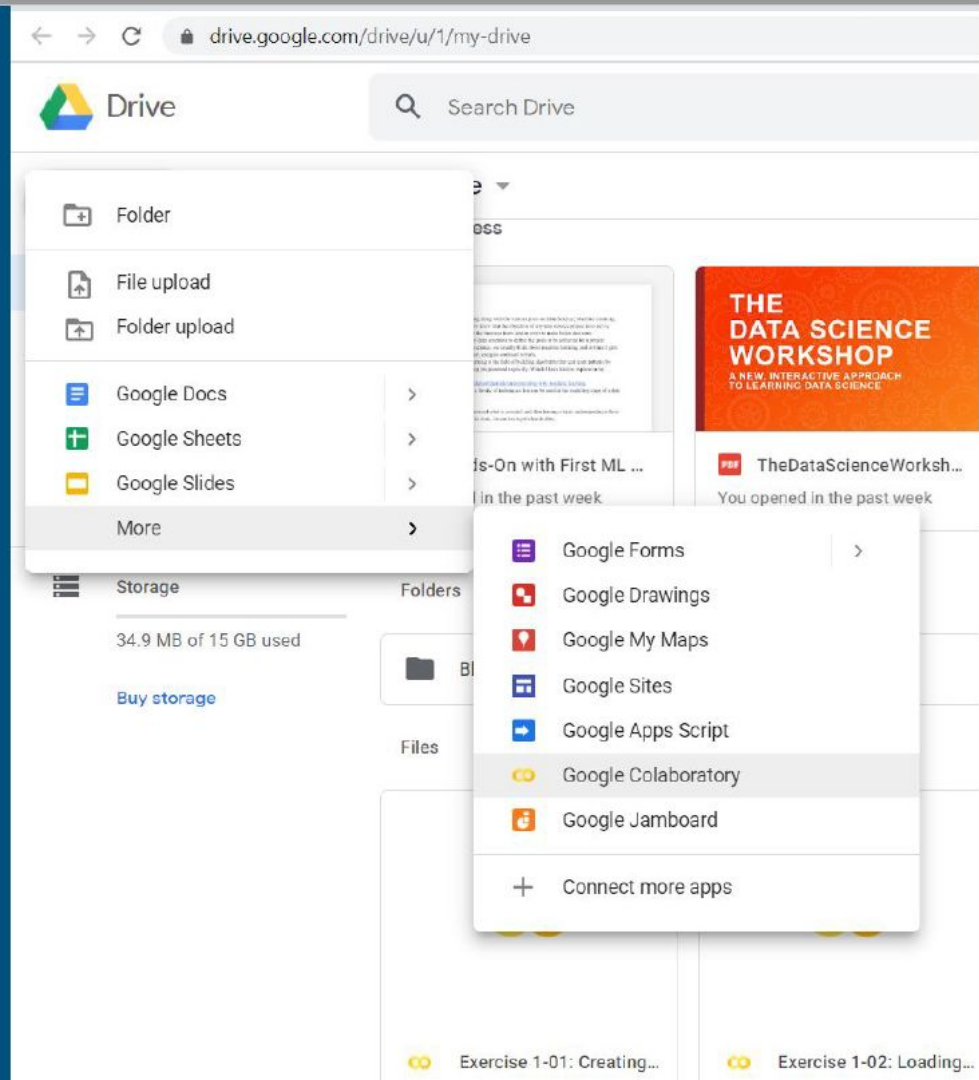
Rank	Language	Type	Score
1	Python▼	  	100.0
2	Java▼	  	95.3
3	C▼	  	94.6
4	C++▼	  	87.0
5	JavaScript▼		79.5

- **Interpretado:** significa que Python “interpreta” el código del programador, es decir, lo traduce y lo ejecuta a la vez.
- **Multiparadigma:** porque es un lenguaje de programación que admite el uso de varios paradigmas de programación (modelos de desarrollo), por lo que no exige a los programadores un estilo único para programar. ¿Cuáles son los paradigmas de programación que permite Python? Programación orientada a objetos, programación imperativa y programación funcional.
- **Multiplataforma:** el lenguaje Python puede ejecutarse en diferentes sistemas operativos como Unix, Linux, macOS y Windows.



GOOGLE COLAB

Colab es un servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de **Google**, con librerías como: Scikit-learn, PyTorch, TensorFlow, Keras y OpenCV.



Listas

- Acceder a elementos
- Desembalaje
- Añadir y eliminar elementos
- Encontrar elementos
- Ordenar elementos
- Bucles
- Comprensión de listas



¿Qué es una lista?

- Una lista es una estructura de datos y un tipo de dato en python con características especiales.
- Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras funciones

```
lista = [125, 45.5, 'Python', [0,1] ,4, 'SEE']
```

Creando una lista

[]

lista = [1 , 2 , 3]



Elementos separados por comas

```
# lista vacia
```

```
lista = []
```

```
# lista de enteros
```

```
lista = [1, 2, 3]
```

```
# lista de diferentes tipos de datos
```

```
lista = [1, "Curso Python", 3.4]
```


Accediendo a los elementos []

```
lista = [ 'hola' , 'amigos' , 'Python' ]
```



Índices:
(entero)

0

1

2

IndexError - TypeError

Ejemplo

```
# Indexacion de listas
```

```
lista = ['p', 'y', 't', 'h', 'o', 'n']
```

```
# Output: ?
```

```
print(lista[0])
```

```
# Output: ?
```

```
print(lista[2])
```

```
# Output: ?
```

```
print(lista[4])
```

```
print(lista[6])
```

```
-----  
IndexError
```

```
<ipython-input-4-b57f4761bb07> in <module>  
----> 1 print(lista[6])
```

```
IndexError: list index out of range
```

```
print(lista[1.0])
```

```
-----  
TypeError
```

```
Traceback (most recent call last):
```

```
<ipython-input-3-4062706af89c> in <module>()  
----> 1 print(lista[1.0])
```

```
TypeError: list indices must be integers or slices, not float
```

Índices negativos

[- i]

- -1 : el último elemento
- -2: el penúltimo elemento

```
lista = ['p', 'y', 't', 'h', 'o', 'n']
```

```
print(lista[-1])
```

```
print(lista[-2])
```

```
n
```

```
o
```

Slicing :

```
lista = ['p', 'y', 't', 'h', 'o', 'n']  
print(lista[0:4])  
  
['p', 'y', 't', 'h']
```

```
lista = ['p', 'y', 't', 'h', 'o', 'n']  
  
print(lista[:4])  
print(lista[4:])  
print(lista[:])  
print(lista[:-4])
```

$i:j$ ➡

$i, i+1, \dots, j-1$



$0:3$ ➡

0,1,2

```
['p', 'y', 't', 'h']  
['o', 'n']  
['p', 'y', 't', 'h', 'o', 'n']  
['p', 'y']
```

P	Y	T	H	O	N
---	---	---	---	---	---



0

1

2

3

4

5

-6

-5

-4

-3

-2


-1

Métodos

- `list.method()`
- `append()` - Añade un elemento al final de la lista
- `extend()` - Añade todos los elementos de una lista a otra lista
- `insert()` - Inserta un elemento en el índice definido
- `remove()` - Elimina un elemento de la lista
- `pop()` - Elimina y devuelve un elemento en el índice definido
- `clear()` - Elimina todos los elementos de la lista
- `index()` - Devuelve el índice del primer elemento coincidente
- `count()` - Devuelve la cuenta del número de elementos pasados como argumento
- `sort()` - Ordena los elementos de una lista en orden ascendente
- `reverse()` - Invierte el orden de los elementos de la lista
- `copy()` - Devuelve una copia superficial de la lista

Loops - listas

- Utilizando un bucle for podemos iterar a lo largo de una lista
- Podemos usar un bucle while recorrer todos los números de índice
- Podemos usar enumerate() si queremos convertir la lista en una lista iterable de tuplas (u obtener el índice basado en una comprobación de la condición)



```
for i, nombre in enumerate(nombres):  
    print (i, ", ", nombre)
```


```
0 , Tania  
1 , Roberto  
2 , Juan
```

```
for nombre in ['Tania', 'Roberto', 'Juan']:  
    print("Mi nombre es ", nombre)
```


```
Mi nombre es  Tania  
Mi nombre es  Roberto  
Mi nombre es  Juan
```

```
nombres = ['Tania', 'Roberto', 'Juan']  
i = 0  
while i < len(nombres):  
    print(nombres[i])  
    i = i + 1
```

```
Tania  
Roberto  
Juan
```



```
[nombre+' SEE' for nombre in nombres]
```



```
['Tania SEE', 'Roberto SEE', 'Juan SEE']
```

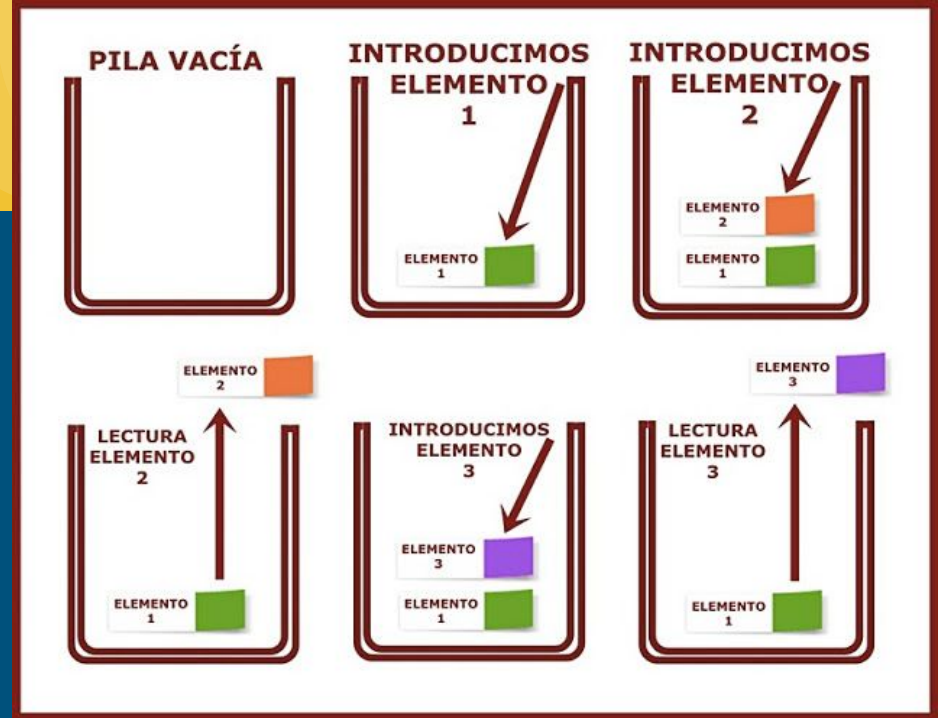


Pilas

pythonTM

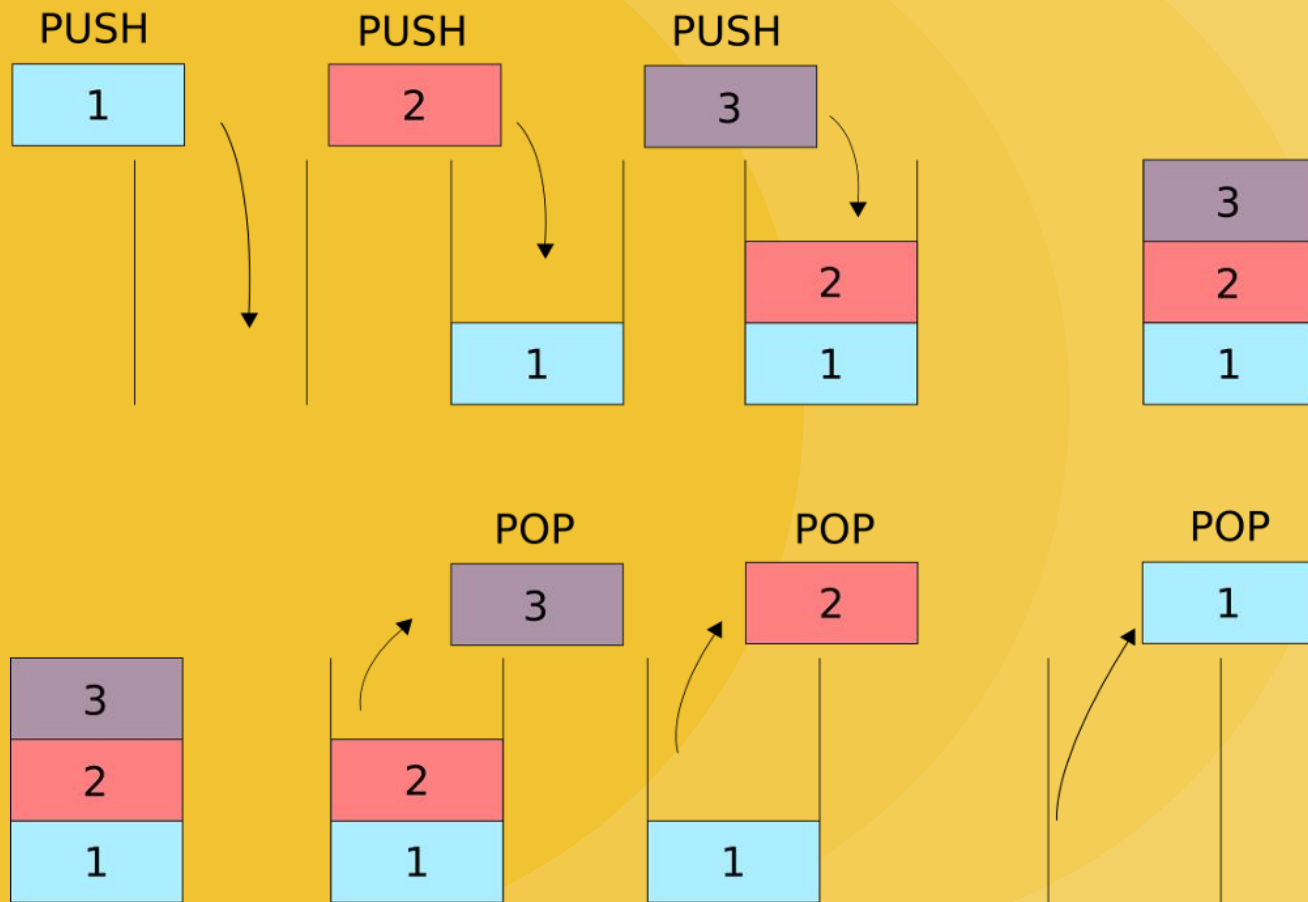
¿Qué es una pila?

- Una pila es una estructura de datos que almacena los elementos de la manera " último en entrar, primero en salir". A menudo se denomina LIFO.
- Tiene las operaciones inicializar, apilar (push) desapilar (pop) y es vacía que devuelve V o F si la pila está vacía o no.



"Lo último que se apiló es lo primero que se usa".

PILA



Implementando una cola en Python

Hay un par de opciones cuando estás implementando una pila de Python.

Veremos las siguientes implementaciones de pila de Python:

- ★ `list`
- ★ `collections.deque`
- ★ `queue.LifoQueue`

Usando listas

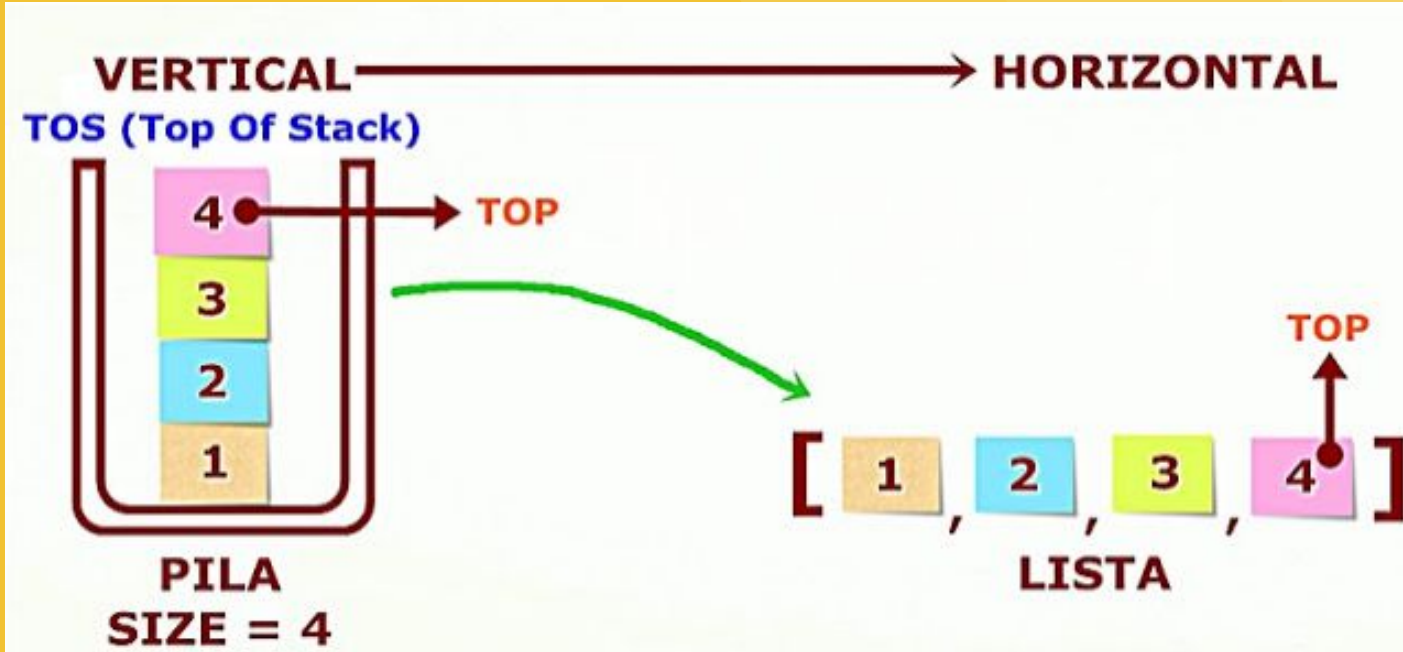


Image: conocephalon - pilas

- Tiene la ventaja de ser familiar. Sabemos cómo funciona.
- Desgraciadamente, las listas tienen algunas deficiencias en comparación con otras estructuras de datos que veremos más adelante.
- El mayor problema es que puede tener problemas de velocidad cuando crece. Los elementos de una lista se almacenan con el objetivo de proporcionar un acceso rápido a elementos aleatorios de la lista. En un nivel alto, esto significa que los elementos se almacenan uno al lado del otro en la memoria.
- Si la pila crece más que el bloque de memoria que la contiene actualmente, entonces Python necesita hacer algunas asignaciones de memoria. Esto puede llevar a que algunas llamadas a `.append()` tarden mucho más que otras.
- También hay un problema menos serio. Si usas `.insert()` para añadir un elemento a tu pila en una posición distinta a la del final, puede llevar mucho más tiempo. Sin embargo, esto no es algo que normalmente harías a una pila.

```
pila = []  
print(pila)  
print('Push')  
pila.append('1') # push  
print(pila)  
pila.append('2')  
print(pila)  
pila.append('3')  
print(pila)  
print('Pop')  
pila.pop()  
print(pila)  
pila.pop()  
print(pila)  
pila.pop()  
print(pila)
```

```
[]  
Push  
['1']  
['1', '2']  
['1', '2', '3']  
Pop  
['1', '2']  
['1']  
[]
```



Diccionarios

Diccionarios

- Un diccionario es una colección desordenada, modificable e indexada.
- Un diccionario consiste en una colección de pares clave-valor. (key -value)
- Cada par clave-valor asigna la clave a su valor asociado. Un diccionario se escribe entre llaves.
- Cada clave está separada de su valor por dos puntos (:), y los elementos están separados por comas.

```
mi_dict = {"nombre": "Katherine", "edad":27, "ciudad": "Paris"}
print(mi_dict)

# o utilizar el constructor dict, nota: no son necesarias las comillas para las claves
mi_dict_2 = dict(nombre="Juan", edad=27, ciudad="Quito")
print(mi_dict_2)

{'nombre': 'Katherine', 'edad': 27, 'ciudad': 'Paris'}
{'nombre': 'Juan', 'edad': 27, 'ciudad': 'Quito'}
```

Acceder a los elementos []

Podemos acceder al elemento de un Diccionario mediante la clave de este elemento

```
mi_dict = {"nombre": "Katherine", "edad":27, "ciudad": ["Paris", 'Quito']}
nombre_en_dicto = mi_dict["ciudad"]
print(nombre_en_dicto)
print(nombre_en_dicto[0])
# KeyError si no se encuentra ninguna clave
print(mi_dict["apellido"])
```

['Paris', 'Quito']
Paris

KeyError Traceback (most recent call last)
<ipython-input-29-0da7a7b7a21f> in <module>()
 4 print(nombre_en_dicto[0])
 5 # KeyError si no se encuentra ninguna clave
----> 6 print(mi_dict["apellido"])

KeyError: 'apellido'

Añadir, modificar y eliminar elementos

```
# Añadir y modificar elementos
# añadir una nueva clave
mi_dict["email"] = "katy@yahoo.com"
print(mi_dict)

# o sobrescribir la clave ya existente
mi_dict["email"] = "otrakaty@yahoo.com"
print(mi_dict)

# borrar un par clave-valor
del mi_dict["email"]

# esto devuelve el valor y elimina el par clave-valor
print("valor extraído:", mi_dict.pop("edad"))
print(mi_dict)

# devuelve y elimina el último par clave-valor insertado
print("elemento pop:", mi_dict.popitem())
print(mi_dict)

# clear() : elimina todos los pares
# mi_dict.clear()

{'nombre': 'Katherine', 'edad': 27, 'ciudad': ['Paris', 'Quito'], 'email': 'katy@yahoo.com'}
{'nombre': 'Katherine', 'edad': 27, 'ciudad': ['Paris', 'Quito'], 'email': 'otrakaty@yahoo.com'}
valor extraído: 27
{'nombre': 'Katherine', 'ciudad': ['Paris', 'Quito']}
elemento pop: ('ciudad', ['Paris', 'Quito'])
{'nombre': 'Katherine'}
```

*Muchas
gracias*