

Métodos de minería de datos en
Python

Programación básica en Python

Contenido

1

Listas

2

Diccionarios

3

Expresiones lambda



Listas

Listas

- Son **secuencias ordenadas de información**, a la cual se puede acceder con índices.
- Se representa con barras cuadradas `[]`.

Las listas contienen elementos:

- Usualmente homogéneos (ejemplo: Todos enteros)
- Puede tener elementos combinados (no es común)
- Los elementos de una lista pueden ser reemplazados (mutable).

Listas

1.

```
L = [2, 'a', 4, [1, 2]]  
print(L)
```



```
[2, 'a', 4, [1, 2]]
```


2.

```
len(L)
```



```
4
```


3.

```
L[0]
```



```
2
```


4.

```
L[0] = L[2]+1  
L[0]
```



```
5
```

5.

```
L[3]
```



```
[1, 2]
```


6.

```
L[3][1]
```



```
2
```


7.

```
L[4]
```

```
IndexError                                Traceback (most recent call last)  
<ipython-input-48-1eef6b78def1> in <module>()  
----> 1 L[4]
```

`IndexError: list index out of range`

SEARCH STACK OVERFLOW

Listas

Se pueden añadir elementos al final de la lista con **L.append(k)**

```
L = [2, 6, 7]  
L.append(10)  
L
```

```
[2, 6, 7, 10]
```

Qué es el punto?

- Las listas son objetos de Python
- Los objetos tienen datos
- Los objetos además tienen métodos y funciones
- Para acceder a los métodos o funciones de un objeto se usa: **object_name.do_something()**

Listas

Las siguientes funciones permiten **remove** elementos de una lista

➤ **Remove un elemento específico:** *L.remove(element)*

```
L = [2,1,3,6,3,7,0]
```

```
L.remove(2)
```

```
L
```

```
[1, 3, 6, 3, 7, 0]
```

Listas

- Las siguientes funciones permiten **remove** elementos de una lista
 - Indicando un **índice específico**: `del(L[index])`

```
L = [2,1,3,6,3,7,0]  
L.remove(2)  
L
```

```
[1, 3, 6, 3, 7, 0]
```

```
del(L[1])  
L
```

```
[1, 6, 3, 7, 0]
```


Listas

- Las siguientes funciones permiten **remove** elementos de una lista
 - **Remove el último elemento de la lista:** *L.pop()*

```
del(L[1])
```

```
L
```

```
[1, 6, 3, 7, 0]
```

```
L.pop()
```

```
L
```

```
[1, 6, 3, 7]
```

Listas

Convertir palabras a listas de caracteres y volver a unirlos:

➤ Convertir cadena de caracteres a lista: *list(s)*

```
s = "I<3 Python"  
print(list(s))
```

```
['I', '<', '3', ' ', 'P', 'y', 't', 'h', 'o', 'n']
```

Listas

Convertir palabras a listas de caracteres y volver a unirlas:

➤ Partir una lista de caracteres: *s.split()*

```
print(s.split('<'))
```

```
['I', '3 Python']
```

Listas

Convertir palabras a listas de caracteres y volver a unirlos:

➤ Convertir lista de caracteres en cadena de texto: `".join(L)`

```
L = ['a', 'b', 'c']  
print(''.join(L))  
print('_'.join(L))
```

abc

a_b_c

Listas

Realizar operaciones con listas que contengan números

➤ *Ordenar*

```
L = [9,6,0,3]  
print(sorted(L))
```

[0, 3, 6, 9]

```
L = [9,6,0,3]  
L.sort()  
L
```

[0, 3, 6, 9]

Listas

Realizar operaciones con listas que contengan números

➤ *Ordenar inversamente*

```
L = [9,6,0,3]  
L.reverse()  
L
```

```
[3, 0, 6, 9]
```

```
L = [9,6,0,3]  
L.sort()  
L.reverse()  
L
```

```
[9, 6, 3, 0]
```

Listas

Adicionalmente las listas permiten:

➤ *Ser clonadas*

```
cool = ['azul', 'verde', 'gris']
newList = cool[:]
newList.append('negro')
print(newList)
print(cool)
```

```
['azul', 'verde', 'gris', 'negro']
['azul', 'verde', 'gris']
```

➤ *Las listas pueden ser mutadas a través de procesos de iteraciones:*

```
L1 = [1,2,3,4,2,5,2,6]

def remove_dups(lista):
    for e in lista:
        lista.remove(e)
    return(lista)
```

```
remove_dups(L1)
```

```
[4, 5, 2, 6]
```

A solid orange horizontal bar, part of a larger decorative element on the left side of the slide.

Diccionarios

Diccionarios

- Un Diccionario es una estructura de datos y un tipo de dato en Python con características especiales que nos permite almacenar cualquier tipo de valor como enteros, cadenas, listas e incluso otras funciones.
- Los diccionario son mutables, es decir, es posible modificar su longitud, podemos agregar o quitar elementos de él; de igual forma todos los valores almacenados en el diccionario pueden ser modificados, *excepto las llaves*.
- A diferencia de las listas y de las tuplas los diccionarios no se rigen por la regla de los índices, en este caso todos los valores que se almacenen en el diccionario no corresponderá a un índice, si no a una llave.

Definir un diccionario

Para definir un diccionario, se encierra el listado de valores entre llaves.

Las parejas de clave y valor se separan con comas, y la clave y el valor se separan con dos puntos.

```
diccionario = {'nombre' : 'Carlos',  
               'edad' : 22,  
               'cursos': ['Python', 'Django', 'JavaScript'] }
```

diccionario

```
{'cursos': ['Python', 'Django', 'JavaScript'], 'edad': 22, 'nombre': 'Carlos'}
```

Otra forma de definir diccionarios

- Se llama el método *dict* y se asigna uno o más valores a una variable dentro del diccionario

```
dic = dict(nombre = 'Carlos',  
           edad = 22,  
           cursos= ['Python', 'Django', 'JavaScript'])
```

```
dic
```

```
{'cursos': ['Python', 'Django', 'JavaScript'], 'edad': 22, 'nombre': 'Carlos'}
```

Otra forma de definir diccionarios

- Recibe como parámetro dos elementos iterables, ya sea una cadena, una lista o una tupla. Ambos parámetros deben tener el mismo número de elementos. Se devolverá un diccionario relacionando el elemento i-esimo de cada uno de los iterables.

```
dic = dict(zip(['nombre',  
               'edad',  
               'Cursos'],  
            ['Carlos',  
             22,  
             ['Python', 'Django', 'JavaScript']]))
```

dic

```
{'Cursos': ['Python', 'Django', 'JavaScript'], 'edad': 22, 'nombre': 'Carlos'}
```

Acceder al diccionario

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

```
print(diccionario['nombre'])  
print(diccionario['edad'])  
print(diccionario['cursos'])
```

Carlos

22

['Python', 'Django', 'JavaScript']

Acceder a listas dentro del diccionario

- También es posible insertar una lista dentro de un diccionario. Para acceder a cada uno de los cursos usamos los índices:

```
print(diccionario['cursos'][0])  
print(diccionario['cursos'][1])  
print(diccionario['cursos'][2])
```

Python
Django
JavaScript

Reemplazar valores

- Esta operación le permite reemplazar el valor específico del *diccionario* mediante su clave.

```
print(diccionario['nombre'])  
diccionario['nombre'] = 'Tatiana'  
print(diccionario['nombre'])
```

Carlos
Tatiana

Asignar valores

- Esta operación le permite asignar el valor específico del *diccionario* mediante una clave nueva.

```
diccionario['Asignatura'] = 'Minería de Datos'  
diccionario
```

```
{'Asignatura': 'Minería de Datos',  
 'cursos': ['Python', 'Django', 'JavaScript'],  
 'edad': 22,  
 'nombre': 'Tatiana'}
```

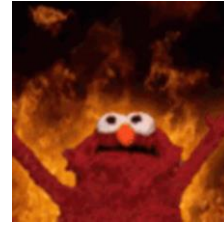

Remover valores del diccionario

- Esta operación le permite eliminar el valor específico del *diccionario* llamando la clave que quiere remover.

```
diccionario.pop('Asignatura', None)  
diccionario
```

```
{'cursos': ['Python', 'Django', 'JavaScript'], 'edad': 22, 'nombre': 'Tatiana'}
```

Invocar ítems

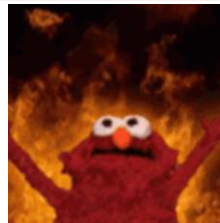


- Devuelve una lista de tuplas, cada tupla se compone de dos elementos: el primero será la clave y el segundo, su valor.

```
diccionario.items()
```

```
dict_items([('nombre', 'Tatiana'), ('edad', 22), ('cursos', ['Python', 'Django', 'JavaScript'])])
```

Invocar llaves

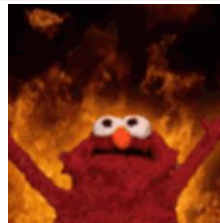


- Retorna una lista de elementos, los cuales serán las claves de nuestro diccionario.

```
diccionario.keys()
```

```
dict_keys(['nombre', 'edad', 'cursos'])
```

Invocar valores

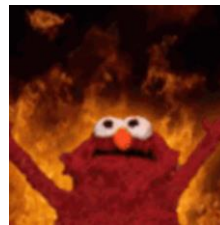


- Retorna una lista de elementos, que serán los valores de nuestro diccionario.

```
diccionario.values()
```

```
dict_values(['Tatiana', 22, ['Python', 'Django', 'JavaScript']])
```

Invocar un valor específico



- Recibe como parámetro una clave, devuelve el valor de la clave. Si no lo encuentra, devuelve un objeto none.

```
diccionario.get('cursos')
```

```
['Python', 'Django', 'JavaScript']
```

Remover todos los ítems del diccionario

- Elimina todos los ítems del diccionario dejándolo vacío.

```
dic.clear()  
dic
```

```
{}
```

Copiar un diccionario

- Retorna una copia del diccionario original.

```
diccionario2 = diccionario.copy()  
diccionario2
```

```
{'cursos': ['Python', 'Django', 'JavaScript'], 'edad': 22, 'nombre': 'Tatiana'}
```

Expresiones lambda

Expresiones lambda

- Las expresiones lambda se usan idealmente cuando necesitamos hacer algo simple y estamos más interesados en hacer el trabajo rápidamente en lugar de nombrar formalmente la función.
- Las expresiones lambda también se conocen como funciones anónimas.
- Como sabes, una función en Python se define con la palabra reservada *def*. Sin embargo, una función anónima se define con la palabra reservada *lambda*.

Definir una función anónima en Python

La sintaxis para definir una *función lambda* es la siguiente:

```
lambda parámetros: expresión
```

A continuación, te detallo las características principales de una *función anónima*:

- Son funciones que pueden definir cualquier número de parámetros pero una única expresión. Esta expresión es evaluada y devuelta.
- Se pueden usar en cualquier lugar en el que una función sea requerida.
- Estas funciones están restringidas al uso de una sola expresión.
- Se suelen usar en combinación con otras funciones, generalmente como argumentos de otra función.

Ejemplo

```
# Función Lambda para calcular el cuadrado de un número
square = lambda x: x ** 2
print(square(5))

# Funcion tradicional para calcular el cuadrado de un numero
def square1(num):
    return num ** 2
print(square(5))
```

25

25

Ejemplos

```
lambda_func = lambda x: True if x**2 >= 10 else False  
print(lambda_func(3))  
print(lambda_func(4))
```

False

True

Ejemplos

```
my_dict = {"A": 1, "B": 2, "C": 3}
sorted(mi_dict, key=lambda x: my_dict[x]%3)
```

```
['C', 'A', 'B']
```

```
print(my_dict["A"]%3)
print(my_dict["B"]%3)
print(my_dict["C"]%3)
```

```
1
2
0
```

Reto

Crear una función que al insertar una lista de *números* arroje la siguiente salida:

Los números ingresados fueron [1, 2, 3, 4]

La suma de los números es: 10

El valor de la multiplicación de los números es: 24

El valor de los números elevados al cuadrado y sumados es: 331776

Condiciones:

- Tiene que servir con cualquier lista de números de cualquier tamaño la lista
- No puede usar librerías, sólo código escrito a mano (Usar for o while Sí se puede, funciones o funciones lambda)



Intro a Github

¿Qué es Github?

GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código.



¿Qué es una versión de control?

Una Versión de Control ayuda a los desarrolladores llevar un registro y administrar cualquier cambio en el código del proyecto de software. A medida que crece este proyecto, la versión de control se vuelve esencial.

Con la *bifurcación*, un desarrollador duplica parte del código fuente (llamado *repositorio*). Este desarrollador, luego puede, de forma segura, hacer cambios a esa parte del código, sin afectar al resto del proyecto.

Luego, una vez que el desarrollador logre que su parte del código funcione de forma apropiada, esta persona podría *fusionar* este código al código fuente principal para hacerlo oficial.

¿Qué es Git?

Git es un sistema de control específico de versión de fuente abierta creada por Linus Torvalds en el 2005.

Específicamente, Git es un sistema de control de versión distribuida, lo que quiere decir que la base del código entero y su historial se encuentran disponibles en la computadora de todo desarrollador, lo cual permite un fácil acceso a las bifurcaciones y fusiones.



Fork and clone

La palabra fork se traduce al castellano, dentro del contexto que nos ocupa, como bifurcación. Cuando hacemos un fork de un repositorio, se hace una copia exacta en crudo del repositorio original que podemos utilizar como un repositorio git cualquiera.

Cuando creas un repositorio en GitHub, existe como un repositorio remoto. Puedes clonar tu repositorio para crear una copia local en tu ordenador y sincronizarlo entre las dos ubicaciones.

Commit, push and pull

Ahora que tienes una copia local y una copia en tu cuenta de GitHub, hay cuatro cosas que tendrás que saber hacer para colaborar.

- Commit (Confirmar) es el proceso que registra los cambios en el repositorio. Piensa en ello como una instantánea del estado actual del proyecto. Las confirmaciones se hacen localmente.
- Push (Empujar) envía el historial de confirmaciones recientes de tu repositorio local a GitHub. Si eres el único que trabaja en un repositorio, empujar es bastante simple. Si hay otras personas que acceden al repositorio, es posible que tengas que hacer un pull antes de poder hacer un push.
- Pull - un pull coge cualquier cambio del repositorio de GitHub y lo fusiona en tu repositorio local.
- Sincronización - la sincronización es como halar, pero en lugar de conectarse a su copia de GitHub del repositorio bifurcado, vuelve al repositorio original y trae cualquier cambio. Una vez que hayas sincronizado tu repositorio, tienes que empujar esos cambios de vuelta a tu cuenta de GitHub.

¡Gracias!

¿Preguntas?

Python is the
easier language
to learn.
No brackets,
no main.



You get errors
for writing an
extra space

