Métodos de minería de datos en Python

Programación básica en Python

Contenido

1. Listas
2. Diccionarios
3. Tuplas

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 reeemplazados **(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 **remover** elementos de una lista

- Remover 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 **remover** 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 **remover** elementos de una lista
  - Remover 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 caractéres y volver a unirlas:

- 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 caractéres y volver a unirlas:

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

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

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

Listas

Convertir palabras a listas de caractéres y volver a unirlas:

- 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]
```

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 diferencias 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-ésimo 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 especifico 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 especifico del *diccionario* mediante una clave nueva.

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

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

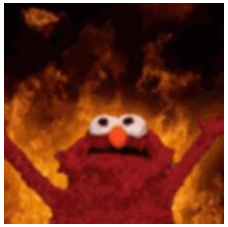
Remover valores del diccionario

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

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

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

Invocar ítems

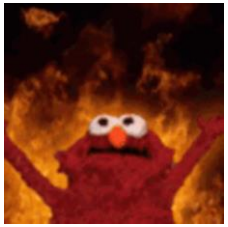


- Devuelve una lista de tuplas, cada tupla se compone de dos elementos: el primero será la clave y el segundo, su valor.

```
diccionario.items()
```

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

Invocar llaves

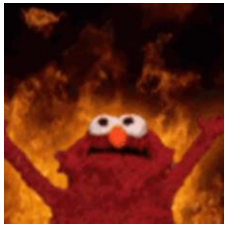


- Retorna una lista de elementos, los cuales serán las claves de nuestro diccionario.

```
diccionario.keys()
```

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

Invocar valores

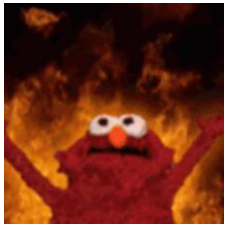


- Retorna una lista de elementos, que serán los valores de nuestro diccionario.

```
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'}
```

Tuplas

Tuplas

- Son secuencias ordenadas de elementos, las cuales pueden albergar **elementos de diferente tipo**.
- Se representa con parentesis ().
- Una vez creada la tupla, no se pueden cambiar los elementos de ella, **son inmutables**.

```
t = (2,"usta",3)
t
```

```
(2, 'usta', 3)
```

```
t[0]
```

```
2
```

```
t[1:2]
```

```
('usta',)
```

```
t[1:3]
```

```
('usta', 3)
```

Tuplas

- Son secuencias ordenadas de elementos, las cuales pueden albergar **elementos de diferente tipo**.
- Se representa con parentesis ().
- Una vez creada la tupla, no se pueden cambiar los elementos de ella, son inmutables.

```
t = (2,"usta",3)
t
```

```
(2, 'usta', 3)
```

```
t[0]
```

```
2
```

```
t[1:2]
```

```
('usta',)
```

```
t[1:3]
```

```
('usta', 3)
```

Tuplas

- Son secuencias ordenadas de elementos, las cuales pueden albergar elementos de diferente tipo.
- Se representa con parentesis ().
- Una vez creada la tupla, no se pueden cambiar los elementos de ella, son inmutables.

```
t + (5,6)
```

```
(2, 'usta', 3, 5, 6)
```

```
t[1:4]
```

```
('usta', 3)
```

```
t[1:5]
```

```
('usta', 3)
```

```
len(t)
```

```
3
```

Tuplas

- Son secuencias ordenadas de elementos, las cuales pueden albergar elementos de diferente tipo.
- Se representa con parentesis ().
- Una vez creada la tupla, no se pueden cambiar los elementos de ella, son inmutables.

```
t[1] = 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-87b0f225887f> in <module>()
----> 1 t[1] = 4

TypeError: 'tuple' object does not support item assignment
```

SEARCH STACK OVERFLOW

Tuplas

```
x = y
y = x
```

```
temp = x
x = y
y = temp
```

```
(x, y) = (y, x)
```

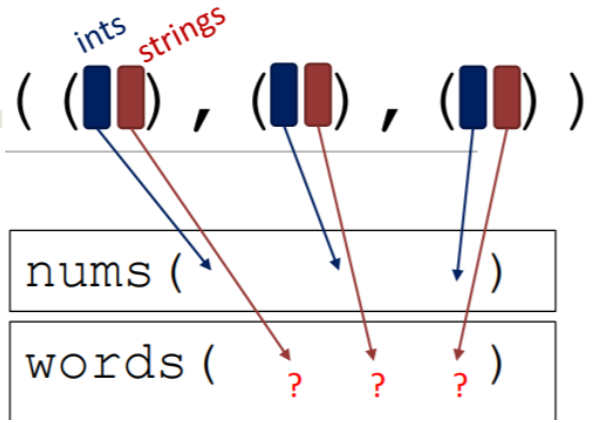
- O para retornar más de un valor en una función:

```
def divide(x, y):
    q = x // y
    r = x % y
    return (q, r)
```

```
print(divide(5,3))
(ent, res) = divide(5,3)
print(ent)
print(res)
```

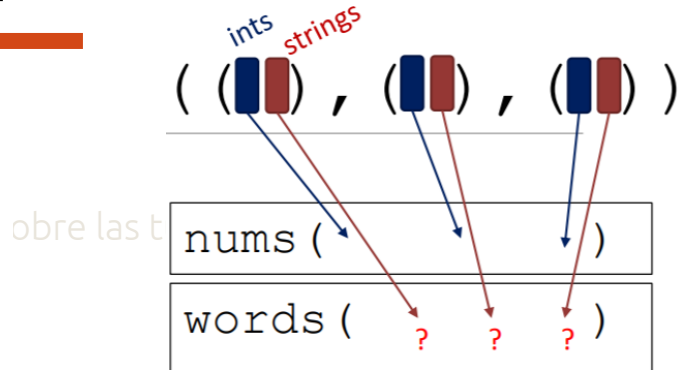
Tuplas

```
def extraer(multiTuple):
    numeros = ()    # tupla vacia
    palabras = ()
    for t in multiTuple:
        # concatenando
        numeros = numeros + (t[0],)
        # unicamente agregar palabras
        # que no han sido agregadas antes
        if t[1] not in palabras:
            palabras = palabras + (t[1],)
    min_n = min(numeros)
    max_n = max(numeros)
    unique_words = len(palabras)
    return (min_n, max_n, unique_words)
```



Tuplas

```
def extraer(multiTuple):
    numeros = ()    # tupla vacia
    palabras = ()
    for t in multiTuple:
        # concatenando
        numeros = numeros + (t[0],)
        # unicamente agregar palabras
        # que no han sido agregadas antes
        if t[1] not in palabras:
            palabras = palabras + (t[1],)
    min_n = min(numeros)
    max_n = max(numeros)
    unique_words = len(palabras)
    return (min_n, max_n, unique_words)
```




```
test = ((1,"a"),(2, "b"),
        (1,"a"),(7,"b"))
(a, b, c) = extraer(test)
print("a:",a,"b:",b,"c:",c)
```

```
a: 1 b: 7 c: 2
```

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)

¡Gracias!

¿Preguntas?