Taller de programación en Python para estadística descriptiva

Elaborado por Joshua Martínez Domínguez

22/03/2025

Este documento contiene los temas desarrollados del *Taller de programación en Python para estadística descriptiva* con ejemplos en celdas de código. Este notebook corresponde a la sesion 3

Tipos estructurados: Tuplas, Listas y Matrices

Similar a las cadenas o *strings*, las tuplas son secuencias de elementos ordenados. La diferencia es que los elementos de una *tupla* no necesitan ser carácteres, pueden ser de cualquier tipo y no necesitan ser del mismo tipo cada elemento.

Las tuplas se escriben definiento elementos separados por comas dentro de un paréntesis.

In [1]:
```
Tupla = ("conejo", 3, 2.566, True)
```

In [2]:
```
Tupla[1]
```

Out[2]:
```
3
```

In [3]:
```
Tupla[0]
```

Out[3]:
```
'conejo'
```

In [5]:
```
Tupla[1:3]
```

Out[5]:
```
(3, 2.566)
```

Las tuplas pueden contener tuplas

In [7]:
```
Tupla1 = ("hoy", 2, 4)
Tupla2 = (0, 1, 2)

Tupla3 = (Tupla1, Tupla2, 7, 8)
print(Tupla3)
```

```
(('hoy', 2, 4), (0, 1, 2), 7, 8)
```

In [8]:
```
Tupla3[0][0]
```

Out[8]:
```
'hoy'
```

Podemos observar que cada elemento esta indexado y podemos ingresar a el de la misma forma que las posiciones de las letras en una cadena.

Las *listas* son como las *tuplas*, una lista es una secuencia ordenada de valores donde cada valor es identificado con un índice. La sintaxis para expresar listas es usando elementos separados por comas entre corchetes.

In [9]:
```
Lista = ["celular", 2, "amor"]
```

Al ser una secuencia, es usual encontrar la combinación con la iteración `for`

In [10]:
```
for i in Lista:
    print(i)
```

```
celular
2
amor
```

In [13]:
```
for i in range(len(Lista)):
    print(i)
```

```
0
1
2
```

In [14]:
```
len(Lista)
```

Out[14]: 3

In [16]:
```
range(3)
```

Out[16]: range(0, 3)

In [17]:
```
for i in range(len(Lista)):
    print(Lista[i])
```

```
celular
2
amor
```

Ocasionalmente, el hecho de que los corchetes son usados para listas, indexacion dentro de listas y separar listas puede conducir a confusiones visuales.

In [18]:
```
[1, 2, 3, 4, 5][1:3][1]
```

Out[18]: 3

En el ejemplo interior podemos observar los 3 usos: el primer corchete es una lista, el segundo es una partición de la primer lista y el tercero es la posición del elemento de la partición realizada.

Las listas difieren de las tuplas en un aspecto importante: las listas son modificables.

A continuación se muestran algunos métodos asociados con las modificaciones en las listas.

Figure 5.4 contains short descriptions of some of the methods associated with lists.

`L.append(e)` adds the object `e` to the end of `L`.

`L.count(e)` returns the number of times that `e` occurs in `L`.

`L.insert(i, e)` inserts the object `e` into `L` at index `i`.

`L.extend(L1)` append the items in list `L1` to the end of `L`.

`L.remove(e)` deletes the first occurrence of `e` from `L`.

`L.index(e)` returns the index of the first occurrence of `e` in `L`.

`L.pop(i)` remove and return the item at index `i`. If `i` is omitted, it defaults to -1.

`L.sort()` has the side effect of sorting the elements of `L`.

`L.reverse()` has the side effect of reversing the order of the elements in `L`.

Figure 5.4 Methods associated with lists

In [19]:
```
#Veamos ejemplos
Lista.append("nuevo")

print(Lista)
```

```
['celular', 2, 'amor', 'nuevo']
```

In [20]:
```
Lista.index("amor")
```

Out[20]: 2

Hasta este momento se ha descrito tres diferentes tipos de secuencias: *str, tuplas y listas*. Ellas son similares en que estos objetos pueden ser operados de la siguiente forma.

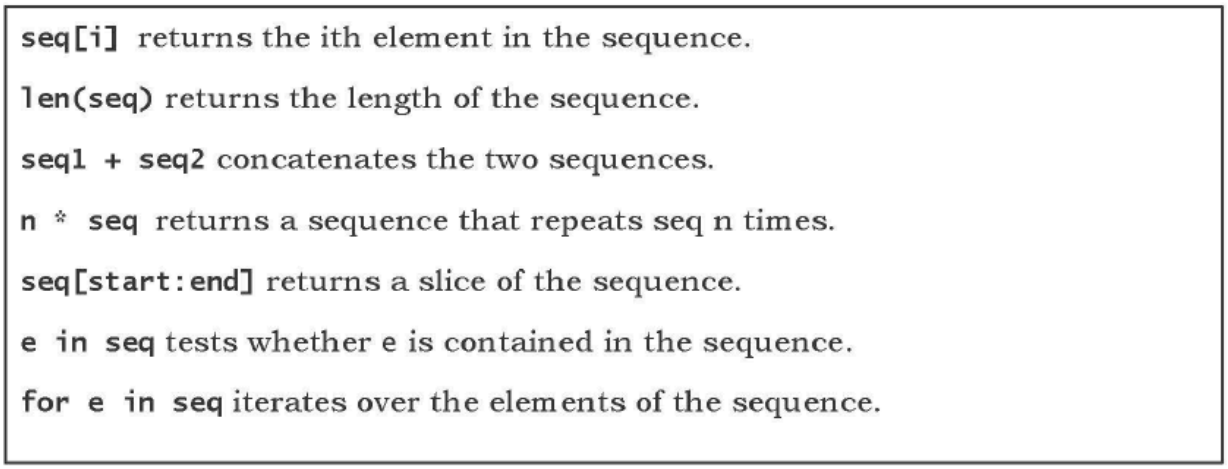

Figure 5.6 Common operations on sequence types

Y a continuación se presenta una tabla de comparación entre ellas

Type	Type of elements	Examples of literals	Mutable
str	characters	'', 'a', 'abc'	No
tuple	any type	(), (3,), ('abc', 4)	No
list	any type	[], [3], ['abc', 4]	Yes

Figure 5.7 Comparison of sequence types

Las matrices son disposiciones bidimensionales de valores. En notación matemática, una matriz se denota encerrando entre paréntesis valores que se disponen en filas y columnas:

$$\begin{pmatrix} 2 & 5 & 0 \\ 7 & 3 & 8 \\ 3 & 0 & 1 \end{pmatrix} \tag{1}$$

Las listas permiten representar series de datos en una sola dimensión. Con una lista de numeros no se puede representar directamente una matriz, pero si con una lista de listas.

In [22]:
```
#Veamos ejemplos
M = [[2, 5, 0], [7, 3, 8], [3, 0, 1]]
print(M)
```

```
[[2, 5, 0], [7, 3, 8], [3, 0, 1]]
```

En la notación matemática el elemento que ocupa la fila i-ésima y la columna j-ésima de una matriz M se representa con $M_{i,j}$. Por ejemplo, el elemento de una matriz que ocupa la celda de la fila 1 y la columna 2 se denota con $M_{1,2}$. Pero si deseamos acceder a ese elemento en la matriz Python M, hemos de tener en cuenta que Python siempre cuenta desde cero, así que la fila tendrá índice 0 y la columna tendrá índice 1

In [23]:
```
M[0][1]
```

Out[23]:
```
5
```

In [27]:
```
M = [ [1, 0, 0], [0, 1, 0], [0, 0, 1] ]

print ("--")
for i in range(0, 3):
    print (M[i])
print ("--")
```

```
--
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
--
```

Diccionarios

Son objetos del tipo *dict* y son como las listas, excepto porque los *indices* no tienen que ser enteros, pueden ser valores de cualquier tipo *inmutable*. No necesariamente estan ordenados y los llamaremos *llaves* o *keys* en lugar de índices.

Es decir, un diccionario es un conjunto de pares de llaves y valores.

La sintaxis de un diccionario cubre sus elementos con *llaves o corchetes curvos* y cada elemento es escrito como *llave* seguido de dos puntos y seguido de un valor.

In [32]:
```
#Veamos ejemplos
Numero_de_mes = {"Enero":1, "Febrero": 2, "Marzo":3, "Abril":4, "Mayo": 5, "Junio":6,
                 1: "Enero_", 2:"Febrero_", 3:"Marzo_", 4:"Abril_", 5:"Mayo_", 6:"Junio
```

In [30]:
```
print(Numero_de_mes["Marzo"])
```

```
3
```

In [33]:
```
print(Numero_de_mes[3])
```

```
Marzo_
```

In [34]:
```
#conozcamos el método keys
Numero_de_mes.keys()
```

Out[34]:
```
dict_keys(['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 1, 2, 3, 4, 5, 6])
```

In [35]:
```
#conozcamos el método values
Numero_de_mes.values()
```

Out[35]:
```
dict_values([1, 2, 3, 4, 5, 6, 'Enero_', 'Febrero_', 'Marzo_', 'Abril_', 'Mayo_', 'Junio
_'])
```

Existen una serie de métodos que permiten usar los diccionarios:

`len(d)` returns the number of items in d.

`d.keys()` returns a list containing the keys in d.

`d.values()` returns a list containing the values in d.

`k in d` returns True if key k is in d.

`d[k]` returns the item in d with key k.

`d.get(k, v)` returns d[k] if k in d, and v otherwise.

`d[k] = v` associates the value v with the key k. If there is already a value associated with k, that value is replaced.

`del d[k]` remove the key k from d.

`for k in d` iterates over the keys in d.

Figure 5.10 Some common operations on dicts

In [36]:
```
Pacientes = {"Jesus":[27, 1], "Armando":[56, 2], "Carmen":[18, 0]}
Pacientes["Jesus"]
```

Out[36]:
```
[27, 1]
```

In [37]:
```
for k in Pacientes:
    print(k)
```

```
Jesus
Armando
Carmen
```

In [39]:
```
for k in Pacientes:
    print(Pacientes[k])
```

```
[27, 1]
[56, 2]
[18, 0]
```

In [41]:
```
for k in Pacientes:
    print("El paciente se llama: " + str(k) + " , tiene " + str(Pacientes[k][0]) + " añ
```

```
El paciente se llama: Jesus , tiene 27 años y 1 enfermedades.
El paciente se llama: Armando , tiene 56 años y 2 enfermedades.
El paciente se llama: Carmen , tiene 18 años y 0 enfermedades.
```

Ejercicio 3

Construye un diccionario con los nombres de 5 personas asociados a una lista con sus valores de peso y altura y obten un diccionario con sus nombres asociados a su IMC usando operadores de listas y diccionarios.

Referencias

Guttag J. (2013). Introduction to Computation and Programming Using Python (Spring 2013 Edition). MIT Press.

Marzal A. (2003). Introducción a la programación con Python. Universitat Jaume I

In [ ]: