El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

Listas y tuplas

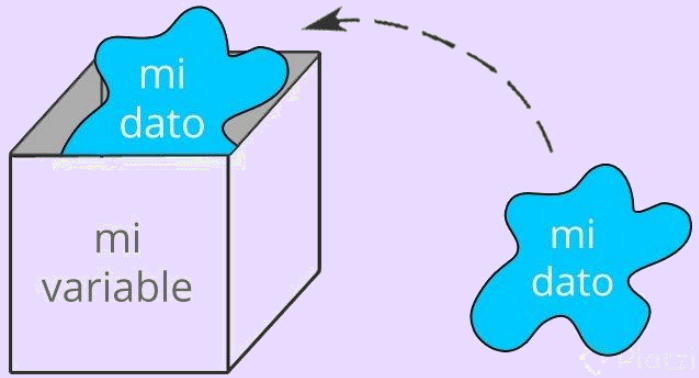
Tutores Misión TIC UIS

Universidad Industrial de Santander

MisionTIC2022

¿Qué es una variable?

Una variable se puede entender como una **especie de caja en la que se puede guardar un valor** (por ejemplo, un valor numérico). Esa caja suele corresponder a una posición de memoria en la memoria del ordenador.



¿Qué es una estructura de datos?

Las estructuras de datos en Python se pueden entender como **un tipo de dato compuesto**, debido a que **en una misma variable podemos almacenar una estructura completa con información**. Dichas estructuras, pueden tener diferentes características y funcionalidades.

Permiten agrupar fácilmente un conjunto de datos (normalmente relacionados) para operar fácilmente con ellos. Cosas como **ordenar, agregar, eliminar, mostrar, recorrer, entre otras operaciones**, son posibles y fáciles en Python.

Listas

Una lista es una **secuencias ordenadas de objetos** de distintos tipos.

Se caracterizan por:

- Tener un orden.
- Contener elementos de distintos tipos.
- Son mutables, es decir, pueden alterarse durante la ejecución de un programa.

Creación de listas

Se construyen poniendo los elementos entre corchetes [ ] separados por comas.

```
1    lista = [1,2,3,4,5,6,7]
2
3    lista = ["Manzana", "Pera", "Sandia"]
4
5    lista = [1, "Pera", "Sandia", 4]
```

Creación de listas con list

list(c) : Crea una lista con los elementos de la secuencia o colección c.

Se pueden indicar los elementos separados por comas, mediante una cadena, o mediante una colección de elementos iterable.

```
1   lista = list([1,2,3,4,5,6,7])
2
3   lista = list(["Manza", "Pera", "Sandia"])
4
5   lista = list([1, "Pera", "Sandia", 4]) # (1, 'Pera', 'Sandia', 4)
6
7   lista = list("Python") # ('P', 'y', 't', 'h', 'o', 'n')
```

Acceso a los elementos de una lista

nombres[i] : Devuelve el elemento de la lista nombres con el índice i.

```
1   lista = ['P', 'y', 't', 'h', 'o', 'n']
2
3   lista[0] # 'P'
4
5   lista[5] # 'n'
6
7   lista[6] # Error
8
9   lista[-1] #n
```

Sublistas

lista[i:j:k] : Devuelve la sublista desde el elemento de lista con el índice i hasta el elemento anterior al índice j, tomando elementos cada k.

```
1   lista = ['P', 'y', 't', 'h', 'o', 'n']
2
3   lista[1:4] # ['y', 't', 'h']
4
5   lista[1:1] # []
6
7   lista[:-3] # ['y', 't', 'h']
8
9   lista[:] #['P', 'y', 't', 'h', 'o', 'n']
10
11  lista[0:6:2] # ['P', 't', 'o']
```

Operaciones que no modifican una lista

- **len(a) :** Devuelve el número de elementos de la lista a.
- **min(a) :** Devuelve el mínimo elemento de la lista a siempre que los datos sean comparables.
- **max(a) :** Devuelve el máximo elemento de la lista a siempre que los datos sean comparables.

```
1   a = [1,2,3,4,5,6,7,8,9]
2
3   len(a) # 9
4
5   min(a) # 9
6
7   max(a)# 9
```

Operaciones que no modifican una lista

- **sum(a) :** Devuelve la suma de los elementos de la lista a, siempre que los datos se puedan sumar.
- **dato in a :** Devuelve True si el dato dato pertenece a la lista a y False en caso contrario.
- **a.index(dato) :** Devuelve la posición que ocupa en la lista a el primer elemento con valor dato.

```
1   a = [1,2,3,4,5,6,7,8,9]
2
3   sum(a) # 45
4
5   8 in a # True
6
7   a.index(7) # 6
```

Operaciones que no modifican una lista

- **a.count(dato) :** Devuelve el número de veces que el valor dato está contenido en la lista a.
- **all(a)** : Devuelve True si todos los elementos de la lista a son True y False en caso contrario.
- **any(a) :** Devuelve True si algún elemento de la lista a es True y False en caso contrario.

```
1   a = [1,2,3,4,5,2,3,4,4,4,5]
2   a.count(3) # 2
3
4   all(a) #True
5
6   any([0, False, 4<2]) #False
```

Operaciones que modifican una lista

- **a + b :** Crea una nueva lista concatenan los elementos de la listas a y b.
- **a.append(dato) :** Añade dato al final de la lista a.
- **a.extend(sequencia) :** Añade los datos de sequencia al final de la lista a.

```
1   a = [1,2,3]
2   b = [4,5,6]
3
4   c = a + b # c -> [1, 2, 3, 4, 5, 6]
5
6   a.append(4) # a -> [1, 2, 3, 4]
7
8   a.extend(b) # a -> [1, 2, 3, 4, 4, 5, 6]
```

Operaciones que modifican una lista

- **a.insert(índice, dato) :** Inserta dato en la posición índice de la lista a y desplaza los elementos una posición a partir de la posición índice.
- **a.remove(dato) :** Elimina el primer elemento con valor dato en la lista a y desplaza los que están por detrás de él una posición hacia delante.
- **a.pop(índice) :** Devuelve el dato en la posición índice y lo elimina de la lista a, desplazando los elementos por detrás de él una posición hacia delante. Si no se le indica índice lo hará con el último elemento.

Operaciones que modifican una lista

```
1   a = ["Pedro", "Maria", "Patricia", "Maria"]
2
3   a.insert(1, 45) # a -> ['Pedro', 45, 'Maria', 'Patricia', 'Maria']
4
5   a.remove("Maria") # a -> ['Pedro', 45, 'Patricia', 'Maria']
6
7   a.pop() # a -> ['Pedro', 45, 'Patricia'] y devuelve 'Maria'
8   a.pop(1) # a -> ['Pedro', 'Patricia'] y devuelve 45
```

Operaciones que modifican una lista

- **a.sort() :** Ordena los elementos de la lista a de acuerdo al orden predefinido, siempre que los elementos sean comparables.
- **a.reverse() :** invierte el orden de los elementos de la lista a.

```
1   a = [56,4,87,12]
2   b = ["maria","andrea","zara"]
3
4   a.sort() # a -> [4, 12, 56, 87]
5   b.sort() # b -> ['andrea', 'maria', 'zara']
6
7   a.reverse() # a -> [87, 56, 12, 4]
8   b.reverse() # a -> ['zara', 'maria', 'andrea']
```

Atención

Existen dos formas de copiar listas:

- **Copia por referencia l1 = l2:** Asocia la variable l1 la misma lista que tiene asociada la variable l2, es decir, ambas variables apuntan a la misma dirección de memoria. Cualquier cambio que hagamos a través de l1 o l2 afectará a la misma lista.
- **Copia por valor l1 = list(l2):** Crea una copia de la lista asociada a l2 en una dirección de memoria diferente y se la asocia a l1. Las variables apuntan a direcciones de memoria diferentes que contienen los mismos datos. Cualquier cambio que hagamos a través de l1 no afectará a la lista de l2 y viceversa.

Tuplas

Una tupla es **una secuencias ordenadas de objetos** de distintos tipos.

Se caracterizan por:

- Tener un orden.
- Pueden contener elementos de distintos tipos.
- Son inmutables, es decir, no pueden alterarse durante la ejecución de un programa.

Creación de tuplas

Se construyen poniendo los elementos entre paréntesis ( ) separados por comas.

```
1   tupla = (1,2,3,4,5,6,7)
2
3   tupla = ("Manza", "Pera", "Sandia")
4
5   tupla = (1, "Pera", "Sandia", 4)
```

Creación de tuplas con tuple

tuple(c) : Crea una tupla con los elementos de la secuencia o colección c.
Se pueden indicar los elementos separados por comas, mediante una cadena, o mediante una colección de elementos iterable.

```
1   tupla = tuple([1,2,3,4,5,6,7])
2
3   tupla = tuple(["Manza", "Pera", "Sandia"])
4
5   tupla = tuple([1, "Pera", "Sandia", 4]) # (1, 'Pera', 'Sandia', 4)
6
7   tupla = tuple("Python") # ('P', 'y', 't', 'h', 'o', 'n')
```

Operaciones con tuplas

El acceso a los elementos de una tupla se realiza del mismo modo que en las listas. También se pueden obtener subtuplas de la misma manera que las sublistas. **Las operaciones de listas que no modifican la lista también son aplicables a las tuplas.**

```
1   tupla = (1,2,3,4,5,6,7,4)
2
3   tupla[1] # 2
4   len(tupla) # 7
5   tupla.index(3) #2
6   0 in tupla # False
7   tupla[1:3] # (2,3)
8   min(tupla) # 1
9   max(tupla) # 7
10  sum(tupla) # 32
11  tupla.count(4) # 2
```

Ejercicios

1. Crea una estructura con los meses del año, pide números al usuario, si el numero esta entre 1 y la longitud máxima de la estructura, muestra el contenido de esa posición sino muestra un mensaje de error. El programa termina cuando el usuario introduce un cero.
2. Pide un numero por teclado y guarda en una estructura su tabla de multiplicar hasta el 10. Por ejemplo, si pide el 5 la lista tendrá: 5,10,15,20,25,30,35,40,45,50
3. Pide una cadena por teclado, mete los caracteres en una estructura sin repetir caracteres.