python™

Colecciones en Python

Formadora: Calletana López Baleta

Colecciones

Una colección permite agrupar varios objetos bajo un mismo nombre. Por ejemplo, si necesitamos almacenar en nuestro programa los nombres de los alumnos de un curso de programación, será más conveniente ubicarlos a todos dentro de una misma colección de nombre alumnos, en lugar de crear los objetos alumno1, alumno2, etc.

En Python existen tres tipos de colecciones básicas:

Fuente: https://tutorial.recursospython.com/

Listas

Una lista es un conjunto ordenado de objetos. Por objetos entendemos cualquiera de los tipos de dato ya mencionados, incluso otras listas.

Para crear una lista, especificamos sus elementos entre corchetes y separados por comas.

Sintaxis

```
>>> lenguajes = ["Python", "Java", "C", "C++"]
```

Dado que se trata de una colección ordenada, accedemos a cada uno de los elementos a partir de su posición, comenzando desde el 0.

```
>>> lenguajes[0]
'Python'
>>> lenguajes[1]
'Java'
```

O también de derecha a izquierda, Indicando un número negativo

```
>>> lenguajes[-1]
'C++'
>>> lenguajes[-2]
'C'
```

Fuente: https://tutorial.recursospython.com/

Listas

Para cambiar un elemento, combinamos la nomenclatura descrita con la asignación.

```
>>> lenguajes[1] = "Rust"
>>> lenguajes
['Python', 'Rust', 'C', 'C++']
```

Para remover un elemento, utilizamos la palabra reservada **del.**

```
>>> del lenguajes[1]
>>> lenguajes
['Python', 'C', 'C++']
```

Nótese que cuando un elemento es eliminado, a excepción del último, todos los que lo suceden se corren una posición hacia la izquierda.

La operación inversa es la de insertar un elemento en una posición determinada, desplazando el resto que le sucede una posición hacia la derecha, con **insert().**

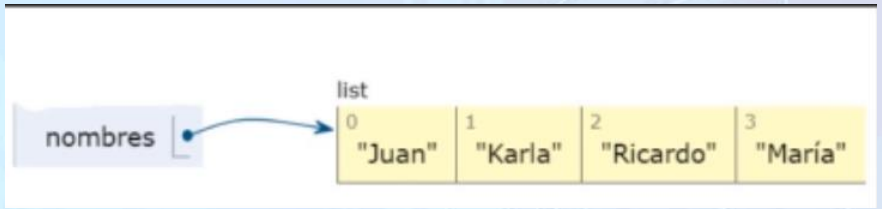
```
# Insertar la cadena "Elixir" en la posición 1.
>>> lenguajes.insert(1, "Elixir")
>>> lenguajes
['Python', 'Elixir', 'C', 'C++']
```

Por último, la operación más común es la de agregar un elemento al final de la lista. Para ello empleamos append().

```
>>> lenguajes.append("Haskell")
>>> lenguajes
['Python', 'Elixir', 'C', 'C++', 'Haskell']
```

Fuente: https://tutorial.recursospython.com/

Listas



Podemos ingresar cualquier tipo de datos en mi lista en Python, para los números simplemente se ubican sin comillas

Para imprimirlas simplemente con la función print(lista)

```
nombres = ['Juan', 'Karla', 'Ricardo', 'Maria' ]
#imprimir lista
print(nombres)
#Acceder a los elementos de una lista
print(nombres[0])
print(nombres[1])
#También lo puedo hacer de derecha a izquierda o de manera inversa
print(nombres[-1])
```

```
['Juan', 'Karla', 'Ricardo', 'Maria']
Juan
Karla
Maria
```

Fuente: Global Mentoring (Acosta, Ubaldo)

Acceder a un rango

```
#Tomar un rango de la lista
print(nombres[0:2])#nota: no incluye el último (2)
```

```
['Juan', 'Karla']
```

Si queremos ir al desde el primer hasta un índice, simplemente no ubicamos el valor inicial; al igual de un índice hasta el último, no escribirnos el último en el rango

```
#Desde el inicio hasta un índice indicado
print(nombres[:3])
#Desde un índice indicado hasta el final
print(nombres[1:])
```

```
['Juan', 'Karla', 'Ricardo']
['Karla', 'Ricardo', 'Maria']
```

Modificar un índice

```
#Cambiar el valor en el índice 2
nombres[2]='Calletana'
print(nombres)
```

```
['Juan', 'Karla', 'Calletana', 'Maria']
```

Fuente: Global Mentoring (Acosta, Ubaldo)

Iterar una lista

Lo podemos hacer con un for

```
#iteramos nuestra lista
for nombre in nombres:
    print(nombre)
```

```
Juan
Karla
Calletana
Maria
```

Obtenemos el largo de una lista con **len**

```
print(len(nombres))
```

Nótese que trae el tamaño real de la lista, no el conteo del índice…

```
4
```

Funciones para Listas

append

Agregar un elemento al final de la lista

```
#agregar elementos a la lista (al final)
nombres.append('Dulce Maria')
print(nombres)
```

```
['Juan', 'Karla', 'Calletana', 'Maria', 'Dulce Maria']
```

insert

Insertar un elemento en un índice específico

```
#insertar un elemento y desplazar a los demás a la derecha
nombres.insert(1, 'Beiker')
print(nombres)
```

```
['Juan', 'Beiker', 'Karla', 'Calletana', 'Maria', 'Dulce Maria']
```

remove

Remueve un elemento por su valor, no Por su índice

```
#remover un elemento por su valor
nombres.remove('Beiker')
print(nombres)
```

```
['Juan', 'Karla', 'Calletana', 'Maria', 'Dulce Maria']
```

pop

Remueve el último elemento de la lista

```
#remover el último elemento de la lista
nombres.pop()
print(nombres)
```

```
['Juan', 'Karla', 'Calletana', 'Maria']
```

Fuente: Global Mentoring (Acosta, Ubaldo)

Funciones para Listas

del

Elimina un elemento por su índice

```
#remover un elemento por su índice
del nombres[0]
print(nombres)
```

```
['Karla', 'Calletana', 'Maria']
```

clear

Eliminar todos los elementos de nuestra lista

```
#Eliminar todos los elementos de nuestra lista
nombres.clear()
print(nombres)
```

```
[]
```

del lista

Remueve la lista por completo de la Memoria, si la mando a imprimir vota un error

```
#Eliminar la lista de memoria
del nombres
print(nombres)
```

```
. Did you mean: 'nombre'?
```

Fuente: Global Mentoring (Acosta, Ubaldo)

Tuplas

En Python, una **tupla** es un conjunto ordenado e inmutable de elementos del mismo o diferente tipo.

Al ser inmutable no se puede modificar ningún elemento de una tupla, esa es su diferencia con las listas

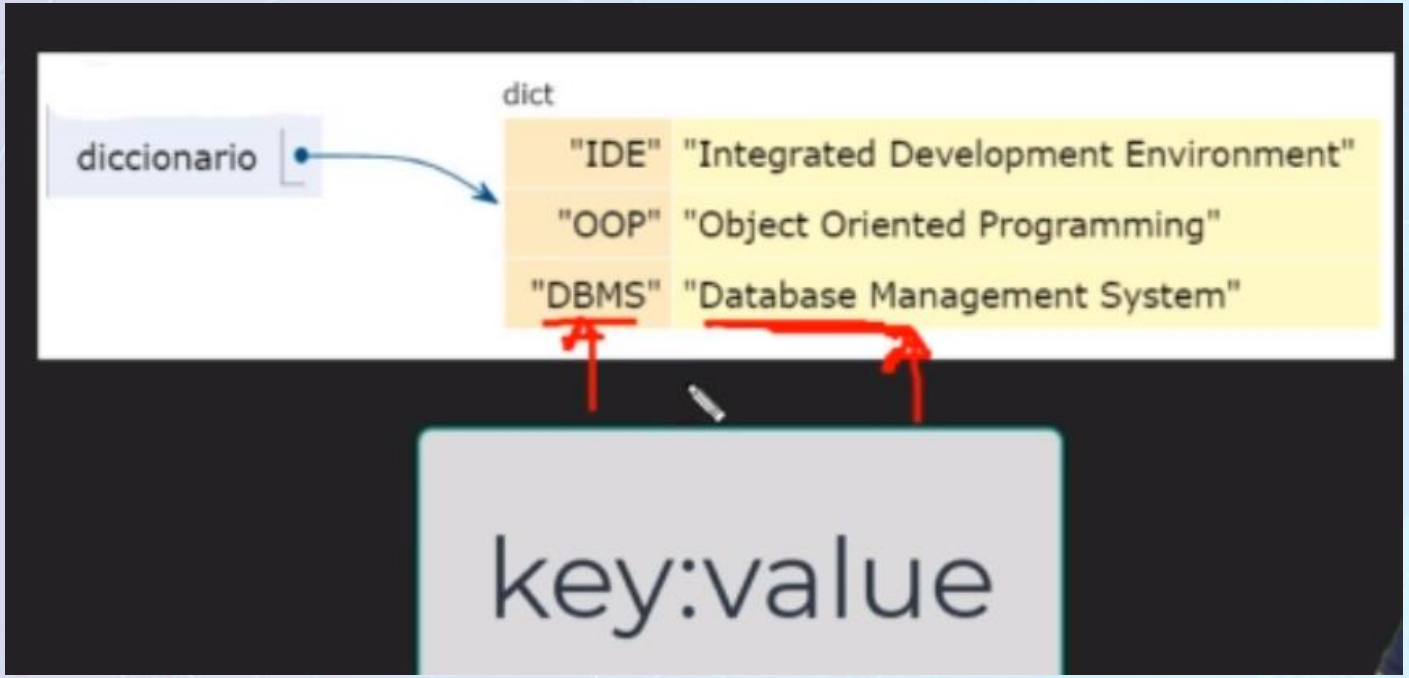
Las tuplas se representan escribiendo los elementos entre paréntesis y separados por comas.

```
>>> (1, "a", 3.14)
(1, 'a', 3.14)
```

```
#Tuplas en Python
frutas = ('Naranja', 'Platano', 'Guayaba')
print(frutas)
```

Fuente: https://www.mclibre.org/

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. Estos diccionarios nos permiten además identificar cada elemento por una clave (Key).

Fuente: https://www.mclibre.org/

Diccionarios

Sintaxis

```
#Diccionarios
#Están computestos por una llave un un valor (key, valor)
# #Para poder acceder a los elementos proporcionamos el key

diccionario={
    101: 'Calletana López',
    102: 'Maria Baleta',
    103: 'Camilo Torres',
    104: 'Rafael José'

}
```

```
print(diccionario)

#Hallar el largo
print(len(diccionario))

#Accedemos a un elemento por su key
print(diccionario[101])

#Otra forma sería con Get, como en java
print(diccionario.get(102))

#Modificar
diccionario[101]= 'Cayetana López'

print(diccionario)
```

Fuente: https://www.mclibre.org/

Diccionarios

Recorrer un diccionario

```
#Recorrer un diccionario con un for

#Podemos hacerlo, por la lalve y el valor
for key, valor in diccionario.items():
    print(key,valor)

#Podemos hacerlo sólo por el key o por el valor

for key in diccionario.keys():
    print(key)

for valor in diccionario.values():
    print(valor)
```

Fuente: https://www.mclibre.org/

Diccionarios

```
#Algunas funciones
#Comprobar la existencia de algún elemento por su Key
print(101 in diccionario)#Devuelve true si está

#Agregar un elemento
diccionario[105]='Jorge Toro'
print(diccionario)

#Remover o eliminar un elemento
diccionario.pop(101)
print(diccionario)

#Limpiar mi diccionario (eliminar elementos más no el diccionario)
diccionario.clear()
print(diccionario)

#Eliminar el diccionario definitivamente
del diccionario
print(diccionario)#Nos va a mandar un error debido a que ya no existe
```

Algunas funciones

También podemos imprimir los diccionarios con la función pprint

Fuente: https://www.mclibre.org/

Diccionarios pprint

Algunas funciones

```
from pprint import pprint

print('Impresión de diccionario con pprint')
pprint(diccionario)
```

```
Impresión de diccionario con pprint
{102: 'Maria Baleta',
 103: 'Camilo Torres',
 104: 'Rafael José',
 105: 'Jorge Toro'}
```

Observemos que agrega un orden

También podemos imprimir los diccionarios con la función pprint

```
pprint(diccionario, sort_dicts=False)
```

Con sort_dicts en false, nos deja el orden de creación del dict

Fuente: https://www.mclibre.org/

Gracias….