Tipos de datos compuestos en Python

.1 Lista, array o vector

Es una secuencia de datos ordenada, heterogénea y mutable, que se escriben entre corchetes y están separados por comas.

- heterogénea: puede contener diferentes tipos de datos
- mutable: sus elementos pueden modificarse
- ordenada: podemos acceder a cada uno de sus elementos por medio de un índice, siendo 0 el índice del primer elemento. También podemos utilizar índices negativos, siendo -1 el último elemento de la lista.

```
mi_lista = ['pan', 32, True, 20.5, [0, 'Juan']]
print(mi_lista) # ['pan', 32, True, 20.5, [0, 'Juan']]

print(mi_lista[0]) # pan
print(mi_lista[2]) # True

mi_lista[2] = False
print(mi_lista[2]) # False

print(mi_lista[-1]) # [0, 'Juan']
```

.1.a) Función len()

Devuelve la longitud (cantidad de elementos) de una lista

```
mi_lista = ['pan', 32, True, 20.5, [0, 'Juan']]
print(len(mi_lista)) # 5
```

.1.b) Métodos

- **append():** agrega un elemento al final de una lista
- **count():** recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista
- **extend():** extiende una lista agregando elementos
- **index():** recibe un elemento como argumento y devuelve el índice de su primera aparición en la lista. Opcionalmente se pueden agregar argumentos adicionales para indicar en qué indices iniciar y terminar la búsqueda. El método devuelve ValueError si no encuentra el elemento en la lista.
- **insert():** inserta un elemento en un índice determinado
- **pop():** muestra el último elemento y lo borra de la lista. Opcionalmente puede recibir un argumento como índice del elemento a eliminar (por defecto es -1).
- **remove():** recibe como argumento un elemento y borra su primera aparición de la lista. El método devuelve ValueError si no encuentra el elemento en la lista.
- **reverse():** invierte el orden de los elementos de una lista

⑩ **sort():** ordena los elementos de una lista

```
mi_lista = ['pan', 32, True, 20.5, [0, 'Juan']]
# append()
mi_lista.append('Toyota')
print(mi_lista) # ['pan', 32, True, 20.5, [0, 'Juan'], 'Toyota']
```

```
# count()
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
print(lista2.count(2)) # 3
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# extend
lista2.extend([4])
print(lista2) # [2, 3, 2.6, 2, 6, 2, 5, 6, 4]

lista2.extend(range(2,8,2))
print(lista2) # [2, 3, 2.6, 2, 6, 2, 5, 6, 4, 2, 4, 6]
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# index()
print(lista2.index(2)) # 0
print(lista2.index(2, 1)) # 3
print(lista2.index(22, 1)) # ValueError: 22 is not in list
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# insert()
lista2.insert(3, 1000)
print(lista2) # [2, 3, 2.6, 1000, 2, 6, 2, 5, 6]
```

```
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# pop()
print(lista2.pop()) # 6
print(lista2) # [2 , 3, 2.6, 2, 6, 2, 5]
print(lista2.pop(2)) # 2.6
print(lista2) # [2, 3, 2, 6, 2, 5]
```

```python
lista2 = [2 , 3, 2.6, 2, 6, 2, 5, 6]
# reverse()
lista2.reverse()
print(lista2) # [6, 5, 2, 6, 2, 2.6, 3, 2]
# sort()
lista2.sort()
print(lista2) # [2, 2, 2, 2.6, 3, 5, 6, 6]
```

.1.c) Conversión a tipo lista

Podemos convertir a tipo de dato lista, mediante la función list()

```python
lista3 = list(range(11))
print(lista3) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Tipos de datos compuestos en Python II

.2 Tuplas

Es una secuencia de datos ordenada, heterogénea e inmutable, que se escriben entre paréntesis y están separados por comas.

- heterogénea: puede contener diferentes tipos de datos
- inmutable: sus elementos no pueden modificarse después de su creación
- ordenada: podemos acceder a cada uno de sus elementos por medio de un índice, siendo 0 el índice del primer elemento. También podemos utilizar índices negativos, siendo -1 el último elemento de la lista.

.2.a) Función len()

Devuelve la longitud (cantidad de elementos) de una tupla

.2.b) Métodos

- **count():** recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la tupla
- **index():** recibe un elemento como argumento y devuelve el índice de su primera aparición en la tupla. Opcionalmente se pueden agregar argumentos adicionales para indicar en qué indices iniciar y terminar la búsqueda. El método devuelve ValueError si no encuentra el elemento en la lista.

```
mi_tupla = ('Python', False, 33, 'Santiago', False)
print(len(mi_tupla)) # 5
print(mi_tupla.count(False)) # 2
print(mi_tupla.index('Python')) # 0
```

.2.c) Conversión a tipo tupla

Podemos convertir a tipo de dato tupla, mediante la función tuple()

```
tupla2 = tuple(range(4,12))
print(tupla2) # (4, 5, 6, 7, 8, 9, 10, 11)
```

Tipos de datos compuestos en Python III

.3 Diccionarios

Son mapeos de datos desordenados y mutables, que se escribe entre llaves, están organizados por pares “key: value” y separados por comas.

Se puede acceder a los valores del diccionario a través de su clave (key).

```
diccionario = {
    'Nombre'   : 'Ana',
    'Apellido' : 'Alvarez',
    'Edad'     : 27,
    'Trabaja'  : True,
    'Notas'    : [8, 9, 7]
}
print(diccionario['Nombre']) # Ana
print(type(diccionario['Notas'])) # <class 'list'>
```

Existe una manera alternativa de crear diccionarios con el método dict():

```
dict2 = dict(lenguaje = 'Python', version = 3.10, framework = 'Flask')
print(dict2) # {'lenguaje': 'Python', 'version': 3.1, 'framework': 'Flask'}
```

.3.a) Operaciones

- Acceder al valor de una clave: mediante su clave.
- Asignar valor a una clave
- Iteración in: devuelve True si una clave está en el diccionario

```
dict2 = dict(lenguaje = 'Python', version = 3.10, framework = 'Flask')
# Acceder al valor de clave
print(dict2['lenguaje']) # Python

# Asignar valor a una clave
dict2['framework'] = 'Django'
print(dict2) # {'lenguaje': 'Python', 'version': 3.1, 'framework': 'Django'}

# Iteración in
print('version' in dict2) # True
print('apellido' in dict2) # False
```

.3.b) Métodos

- **clear():** remueve todos los elementos de un diccionario
- **copy():** devuelve una copia del diccionario

- **fromkeys():** crea un nuevo diccionario con claves a partir de un tipo de dato secuencia. El valor por defecto es de tipo None.
- **get():** devuelve el valor de una búsqueda mediante clave. None si no lo encuentra.
- **items():** Devuelve una lista de tuplas.
- **keys():** Devuelve una lista con las claves del diccionario.
- **pop():** Remueve una clave del diccionario. Lanza error KeyError si no la encuentra.
- **popitem():** Remueve un par clave:valor del diccionario como 2-tupla
- **setdefault():** asigna valores por defecto a las claves de un diccionario.
- **update():** actualiza un diccionario agregando los apres clave:valor en un segundo diccionario.
- **values():** Devuelve una lista de los valores del diccionario.

```python
dict1 = dict(lenguaje = 'Python', version = 3.10, framework = 'Flask')
# copy()
dict2 = dict1.copy()
print(dict2) # {'lenguaje': 'Python', 'version': 3.1, 'framework': 'Flask'}

# clear()
dict2.clear()
print(dict2) # {}
```

```python
# fromkeys()
tupla = ('nombre', 'apellido', 'edad')
dict3 = dict.fromkeys(tupla)
print(dict3) # {'nombre': None, 'apellido': None, 'edad': None}
```

```python
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75,  nota3 = 7.25)
# get()
print(dict4.get('nota2')) # 9.75
print(dict4.get('apellido')) # None

# items()
print(dict4.items())
# dict_items([('alumno', 235645), ('nota1', 8.5), ('nota2', 9.75), ('nota3', 7.25)])
```

```python
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75,  nota3 = 7.25)
# keys()
print(dict4.keys()) # dict_keys(['alumno', 'nota1', 'nota2', 'nota3'])

# pop()
print(dict4.pop('alumno')) #235645
print(dict4) # {'nota1': 8.5, 'nota2': 9.75, 'nota3': 7.25}

dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75,  nota3 = 7.25)
# popitem()
print(dict4.popitem()) # ('nota3', 7.25)
print(dict4) # {'alumno': 235645, 'nota1': 8.5, 'nota2': 9.75}
```

```python
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75,  nota3 = 7.25)
# setdefault()
alumno = dict4.setdefault('alumno') # la clave existe
print(alumno) # 235645
apellido = dict4.setdefault('apellido') # la clave no existe
print(apellido) # None
nombre = dict4.setdefault('nombre', 'Eduardo') # la clave no existe, pero damos dato
print(nombre) # Eduardo
print(dict4)
# {'alumno': 235645, 'nota1': 8.5, 'nota2': 9.75, 'nota3': 7.25,
# 'apellido': None, 'nombre': 'Eduardo'}
```

```python
dict4 = dict(alumno = 235645, nota1 = 8.50, nota2 = 9.75,  nota3 = 7.25)
# update
otro_dict = dict(nombre = 'Álvaro')
dict4.update(otro_dict)
print(dict4)
# {'alumno': 235645, 'nota1': 8.5, 'nota2': 9.75,
#'nota3': 7.25, 'nombre': 'Álvaro'}

# values()
print(dict4.values()) # dict_values([235645, 8.5, 9.75, 7.25, 'Álvaro'])
```

.3.c) Funciones

⑩ **len():** devuelve la cantidad de elementos de un diccionario.

```python
dict5 = dict(nota1 = 8.50, nota2 = 9.75,  nota3 = 7.25)

# len()
print(len(dict5)) # 3
```

Tipos de datos compuestos en Python IV

.4 Conjuntos

Un conjunto es una colección no ordenada y sin elementos repetidos. Los usos básicos de este tipo de datos son la verificación de pertenencia y eliminación de entradas duplicadas.

.4.a) Set

Es de tipo mutable, desordenado y no contiene duplicados.

.4.b) Frozenset

Es de tipo inmutable, desordenado y no contiene duplicados.

.4.c) Métodos

- **add():** agrega un elemento a un conjunto mutable.
- **clear():** remueve todos los elementos de un conjunto mutable.
- **copy():** devuelve una copia de un conjunto mutable o inmutable.
- **difference():** devuelve la diferencia entre dos conjuntos mutables o inmutables.
- **difference_update():** actualiza un tipo conjunto mutable con la diferencia de los conjuntos.
- **discard():** remueve un elemento de un conjunto mutable.
- **intersection():** devuelve la intersección entre los conjuntos mutables o inmutables.
- **intersection_update():** actualiza un conjunto mutable con la intersección de ese mismo y otro conjunto mutable.
- **isdisjoint():** devuelve True si no hay elementos comunes entre conjuntos mutables o inmutables.
- **issubset():** devuelve True si el conjunto mutable es un subconjunto del conjunto mutable o inmutable del argumento.
- **issuperset():** devuelve True si el conjunto mutable o inmutable es un superset (contiene) del conjunto mutable argumento.
- **pop():** elimina aleatoriamente un elemento del conjunto mutable.
- **remove():** elimina arbitrariamente un elemento de un conjunto mutable.
- **symmetric_difference():** decuelve todos los elementos que están en un conjunto mutable e inmutable u otro, pero no en ambos.
- **symmetric_difference_update():** actualiza a un conjunto mutable con el contenido de la diferencia simétrica.
- **union():** devuelve un conjunto mutable e inmutable con todos los elementos que están en alguno de los conjuntos mutable e inmutables
- **update():** agrega elementos desde un conjunto mutable pasado como argumento.

```python
mi_set = set([4, 3, 11, 7, 5, 2, 1, 4])
print(mi_set) # {1, 2, 3, 4, 5, 7, 11}

# add()
mi_set.add(22)
print(mi_set) # {1, 2, 3, 4, 5, 7, 11, 22}

# copy()
mi_set2 = mi_set.copy()
print(mi_set == mi_set2) # True

# clear()
mi_set2.clear()
print(mi_set2) # set()
```

```python
mi_set = set([3, 11, 7, 5, 2, 1, 4, 23])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 55, 70])

# difference
print(mi_set.difference(mi_set2)) # {23}
print(mi_set2.difference(mi_set)) # {70, 55}

# difference_update
mi_set2.difference_update(mi_set)
print(mi_set2) # {70, 55}

# discard
mi_set.discard(23)
print(mi_set) # {1, 2, 3, 4, 5, 7, 11}
```

```python
mi_set = set([3, 11, 7, 5, 2, 1, 4, 23])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 55, 70])

# intersection()
print(mi_set.intersection(mi_set2)) # {1, 2, 3, 4, 5, 7, 11}

# intersection_update()
mi_set.intersection_update(mi_set2)
print(mi_set) # {1, 2, 3, 4, 5, 7, 11}
```

```python
mi_set = set([3, 11, 7, 5, 2, 1, 4])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 55, 70])

# isdisjoint()
print(mi_set.isdisjoint(mi_set2)) # False

# issubset()
print(mi_set.issubset(mi_set2)) # True

# issuperset()
print(mi_set2.issuperset(mi_set)) # True
```

```python
mi_set = set([3, 11, 7, 5, 2, 1, 4])

# pop()
print(mi_set.pop()) # 1
print(mi_set) # {2, 3, 4, 5, 7, 11}

# remove()
mi_set.remove(7)
print(mi_set) # {2, 3, 4, 5, 11}

mi_set2 = { 55, 112}
# update()
mi_set.update(mi_set2)
print(mi_set) # {2, 3, 4, 5, 11, 112, 55}
```

```python
mi_set = set([3, 11, 7, 5, 2, 1, 4, 55, 77])
mi_set2 = set([3, 11, 7, 5, 2, 1, 4, 100, 123])

# symmetric_difference()
print(mi_set.symmetric_difference(mi_set2)) # {100, 77, 55, 123}

# union()
print(mi_set.union(mi_set2)) # {1, 2, 3, 4, 5, 100, 7, 11, 77, 55, 123}

# symmetric_difference_update()
mi_set.symmetric_difference_update(mi_set2)
print(mi_set) # {100, 77, 55, 123}
```

Slicing: rebanadas

Un slice es un subconjunto en una lista de elementos. La función slice() devuelve un objeto slice (una rebanada de una lista o tupla).

Sintaxis

```
x = slice(inicio, final, step)
```

```
a = [2, 5, 8, 11, 15, 18, 20, 22, 50]
x = slice(2,9,3)
print(a[x])  # [8, 18, 50]
```

Notación

```
a[inicio:final] # desde el elemento 'inicio' hasta 'final'-1
a[inicio:] # desde el elemento 'inicio' hasta el final del array
a[:final] # desde el primer elemento hasta elemento 'final'-1
a[:] # todos los elementos del array
a[-1] # selecciona el último elemento del array
a[-2:] # selecciona los dos últimos elementos del array
a[:-2] # selecciona todos los elementos excepto los dos últimos
```

```
a = [2, 5, 8, 11, 15, 18, 20, 22]
print(a[1:4]) # [5, 8, 11]
print(a[5:]) # [18, 20, 22]
print(a[:4]) # [2, 5, 8, 11]
print(a[:]) # [2, 5, 8, 11, 15, 18, 20, 22]
print(a[-1]) # 22
print(a[-2:]) # [20, 22]
print(a[:-2]) # [2, 5, 8, 11, 15, 18]
```