

Container

A container is a data structure that can store and organize multiple values or objects.

The common types of container : lists, strings, dictionaries, sets, tuples

propriétés :

Capacité à supporter le test d'appartenance, for example, 'to' in 'toto' return True

Capacité à supporter la fonction len()

renvoyant la longueur du container.

ordonné/ordered; indexable/subscriptable;

itérable/iterable

itérable/iterable

immuable, identifiant id(), hachable

(hasable, hashability) hash()

Tuplet

iterable, ordered, indexable, similar to lists but immutable.

Avoid containing mutable variables, such as list, dictionary

```
(1,2,3)
```

```
tuplet()
```

```
# create a new tuplet and add element
```

```
(different id)
```

```
set1 = (1, 2, 3)
```

```
set1 = set1 + (2,)
```

```
# operators
```

```
>>> (1, 2) + (3, 4)
```

```
(1, 2, 3, 4)
```

```
>>> (1, 2) * 4
```

```
(1, 2, 1, 2, 1, 2, 1, 2)
```

```
# iteration
```

```
dic1.items() return a list contains tuplets,
```

```
each tuplet contains key/value pair
```

```
similar to enumerate()
```

```
>>> for bidule in enumerate([75, -75, 0]):
```

```
... print(bidule, type(bidule))
```

Tuplet (cont)

```
...
```

```
(0, 75) <class 'tuple'>
```

```
(1, -75) <class 'tuple'>
```

```
(2, 0) <class 'tuple'>
```

Dictionary

```
dic = {key1: value1, key2: value2, ...}
```

```
# iterable by key
```

```
dic.it ems() & dic.keys() &
```

```
dic.va lues()
```

```
# ordered by key or value
```

```
sorted (dic) # by key
```

```
sorted (dic, revers e=True)
```

```
sorted (dic, key=di c.get) #
```

```
by value
```

```
min(dic, key=di c.get) &
```

```
max(dic, key=di c.get)
```

```
# return the key with maximun or
```

```
minimun value
```

```
# get value
```

```
dic[key] & dic.ge t(key)
```

```
# if the key exist, dic[key]
```

```
return error
```

```
# modify value / add new key-
```

```
value pair / remove key
```

```
dic[ke y] = value
```

```
del dic[key]
```

```
dict.p op (key)
```

```
dic.up dat a({key: value})
```

```
#duplicate a dictionary (same to
```

```
lists)
```

```
dic2 = dic1.c opy()
```

```
# transform list of list to
```

```
dictionary
```

```
dic1 = dir([[ 'a', 1], [ 'b', 2]])
```

```
# list of dictionary & iteration
```

```
>>> animaux = [ani1, ani2]
```

```
>>> animaux
```

```
[{'n': 'g', 'p': 1, 't': 5},
```

```
{ 'n': 's', 'p': 7, 't': 1}]
```

```
>>> for ani in animaux:
```

```
... print( ani ["n"])
```

```
...
```

Dictionary (cont)

```
> girafe
```

```
singe
```

```
>>>len(animaux)
```

```
2 # length of dictionaries in a list
```

les objets utilisés comme clé doivent être hachables

Si un des sous-éléments a plus de 2 éléments (ou moins), Python renvoie une erreur

Set

iterable, mutable, unordered, indexable, a list without duplicated element

```
{1,2,3}
```

```
# transforme a list to a set
```

```
set()
```

```
# add the new element at the end
```

```
set.add()
```

```
# remove element
```

```
set.discard()
```

```
set.remove()
```

remove() raises an exception if the element is not present, but discard() does not

```
# add multiple elements to a set
```

```
set.update()
```

This method takes any iterable object as an argument, such as another set, a list, a tuple, or a dictionary. For example, set s = {1, 2, 3}, add {4, 5} and [6, 7] to it by calling s.update({4, 5}, [6, 7]). This will result in s = {1, 2, 3, 4, 5, 6, 7}

```
# operations of sets
```

```
set(list1) & set(list2) # elements
```

```
set(list1) | set(list2) # union
```

```
set(list1) - set(list2) # difference
```

Range

range(start, stop, step)
step could be negative, for example
range(5, 1, -1)
similar to lists, but immutable/hashable
transform range to list
list(range())

C

By **Theo666**

cheatography.com/theo666/

Published 16th October, 2023.

Last updated 9th October, 2023.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>