Programmation en Python

Anne Garcia-Sanchez

M2i cyber dev - CFA CCI Avignon
12 octobre 2023

Séquences

séquences: ensemble de types

chaînes de caractères, listes, tuples, bytes, ...

`str`, `list`, `tuple`, `bytes`

ensemble fini et ordonné de n éléments indicés de 0 à $n-1$

Séquences

opérations communes à toutes les séquences:

- accès par notation **crochet** []
- nombre d'éléments: **len**
- test d'appartenace: **in**
- concaténation: +
- recherche de la première occurence: **index**
- nombre d'occurrence d'une valeur: **count**
- **min**, **max**, *

Séquences

exemple avec une chaîne de caractères

```
>>> m = 'message'
>>> len(m)
7
>>> m[0]
'm'
>>> m[6]
'e'
>>> 'sa' in m
True
>>> m.index('a')
4
>>>
```

Séquences - slicing

`m[debut:fin:pas]` retourne un nouvel objet

`debut` inclus - `fin` exclue

chaîne	m	e	s	s	a	g	e
indice	0	1	2	3	4	5	6

```
>>> m
'message'
>>> m[2:]
'ssage'
>>> m[2:4]
'ss'
>>> m[-2:]
'ge'
>>> m[::2]
'msae'
>>> m[::-2]
'easm'
>>> m[::-1]
'egassem'
>>>
```

Séquences - slicing

`[ : ]` retourne une copie

Séquences - slicing

indice en dehors de la séquence provoque une exception

```
>>> m = 'message'
>>> m[100]
Traceback (most recent call last):
  File "/usr/lib/python3.8/idlelib/run.py", line 55
    exec(code, self.locals)
  File "<pyshell#4>", line 1, in <module>
IndexError: string index out of range
>>>
```

Chaînes de caractères

```
s = "message"
```

```
s = 'message'
```

nombreuses méthodes: renvoient nouvel objet chaîne de caractères

```
>>> 'message'.upper()
'MESSAGE'
```

Chaînes de caractères

objets immuables

```
>>> m = 'message'
>>> m[0] = 'a'
Traceback (most recent call last):
  File "/usr/lib/python3.8/idlelib/run.py", line 559, in runcode
    exec(code, self.locals)
  File "<pyshell#1>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

Listes

séquence d'objets hétérogènes

```
>>> mylist = [12, 'message', True, 4.5]
>>> mylist
[12, 'message', True, 4.5]
>>> type(mylist)
<class 'list'>
>>> mylist[0]
12
>>>
```

Listes

ne stocke pas les objets mais les références vers ces objets

objet mutable

pas besoin de faire une copie pour modifier

efficace au niveau mémoire

```
>>> mylist
[12, 'message', True, 4.5]
>>> mylist[0] = 0
>>> mylist
[0, 'message', True, 4.5]
>>>
```

Listes

opération sur un élément d'une liste directement

```
>>> mylist = [12, 'message', True, 4.5]
>>> mylist
[12, 'message', True, 4.5]
>>> mylist[0] = mylist[0] + 10
>>> mylist
[22, 'message', True, 4.5]
>>>
```

Listes

objet mutable

objet qui peut être modifié en place

```
>>> mylist
[0, 'message', True, 4.5]
>>> mylist.reverse()
>>> mylist
[4.5, True, 'message', 0]
>>>
```

Listes

ATTENTION

la méthode ne renvoie rien: la modification est faite en place

```
>>> mylist = [12, 'message', True, 4.5]
>>> mylist
[12, 'message', True, 4.5]
>>> mylist = mylist.reverse()
>>> mylist
>>> type(mylist)
<class 'NoneType'>
>>>
```

Listes

slicing et opérations sur les slices

```
>>> mylist
[22, 'message', True, 4.5]
>>> mylist[1:3]
['message', True]
>>> mylist
[22, 'message', True, 4.5]
>>> mylist[1:3] = [1, 2, 3, 4]
>>> mylist
[22, 1, 2, 3, 4, 4.5]
>>>
```

insertion d'éléments à la place des éléments effacés

Listes

permet d'effacer des éléments

```
>>> mylist
[12, 'message', True, 4.5]
>>> mylist[1:3] = []
>>> mylist
[12, 4.5]
>>>
```

utilisation de del

```
>>> mylist
[12, 4.5]
>>> del mylist[1:2]
>>> mylist
[12]
>>>
```

Opérations sur les listes

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '
__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__
lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reverse
d__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
>>> help(list.append)
Help on method_descriptor:

append(self, object, /)
    Append object to the end of the list.

>>>
```

append

```
>>> mylist
[12, 'message', True, 4.5]
>>> mylist.append('hello')
>>> mylist
[12, 'message', True, 4.5, 'hello']
>>>
```

tris de listes

en place avec la méthode `sort`:

```
>>> list1 = [3, 2, 1, 3, 5, 2]
>>> list1
[3, 2, 1, 3, 5, 2]
>>> list1.sort()
>>> list1
[1, 2, 2, 3, 3, 5]
>>>
```

économie d'allocation mémoire

tris de listes

tri sur une copie de la liste avec la fonction `sorted`:

```
>>> list1 = [3, 2, 1, 3, 5, 2]
>>> list2 = sorted(list1)
>>> list1
[3, 2, 1, 3, 5, 2]
>>> list2
[1, 2, 2, 3, 3, 5]
>>>
```

Listes et chaînes de caractères

```
>>> c = 'ma chaine de caractères'
>>> list1 = c.split()
>>> list1
['ma', 'chaine', 'de', 'caractères']
>>> "".join(list1)
'machainedecaractères'
>>> " ".join(list1)
'ma chaine de caractères'
>>> list1
['ma', 'chaine', 'de', 'caractères']
```

chaînes de caractères: objets immuables

```
>>> str1 = 'ab'
>>> str2 = str1
>>> str1 += 'cd'
>>> str1
'abcd'
>>> str2
'ab'
>>>
```

listes: objets mutables

```
>>> list1 = ['a', 'b']
>>> list2 = list1
>>> list1 += ['c', 'd']
>>> list1
['a', 'b', 'c', 'd']
>>> list2
['a', 'b', 'c', 'd']
>>>
```