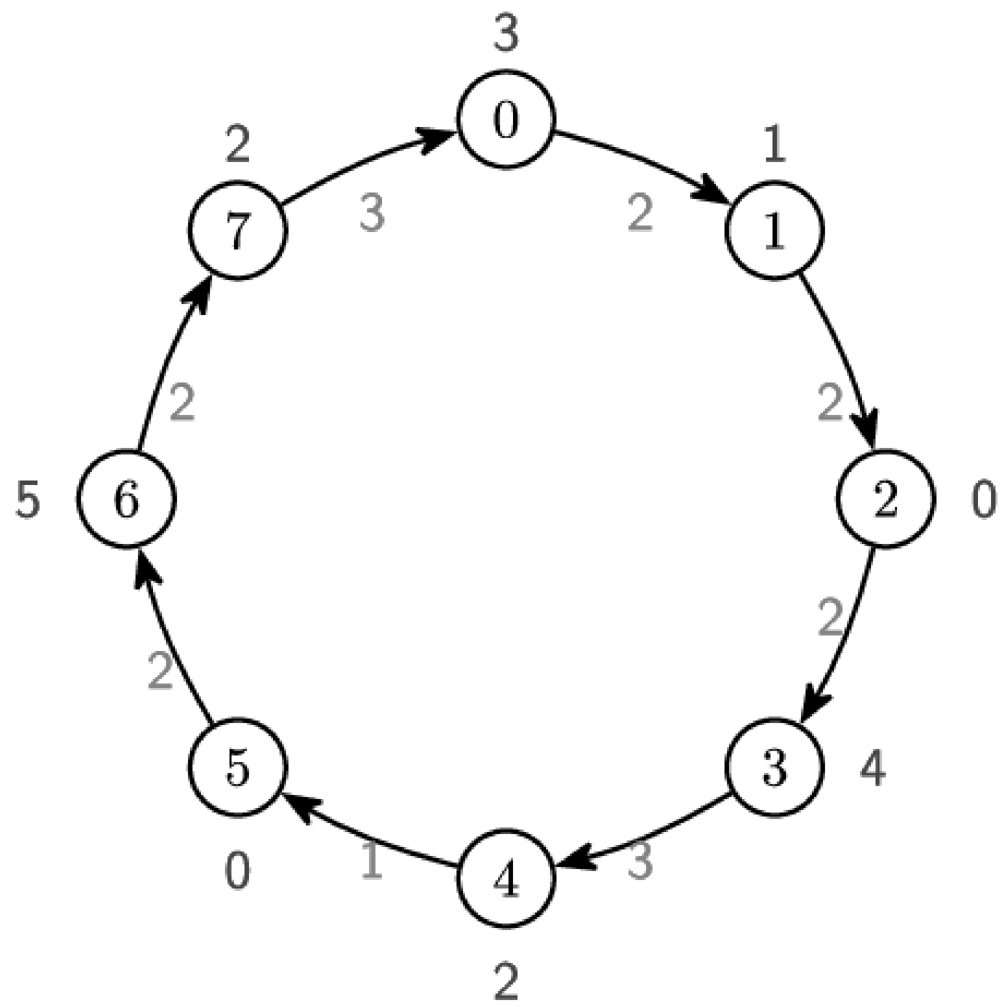
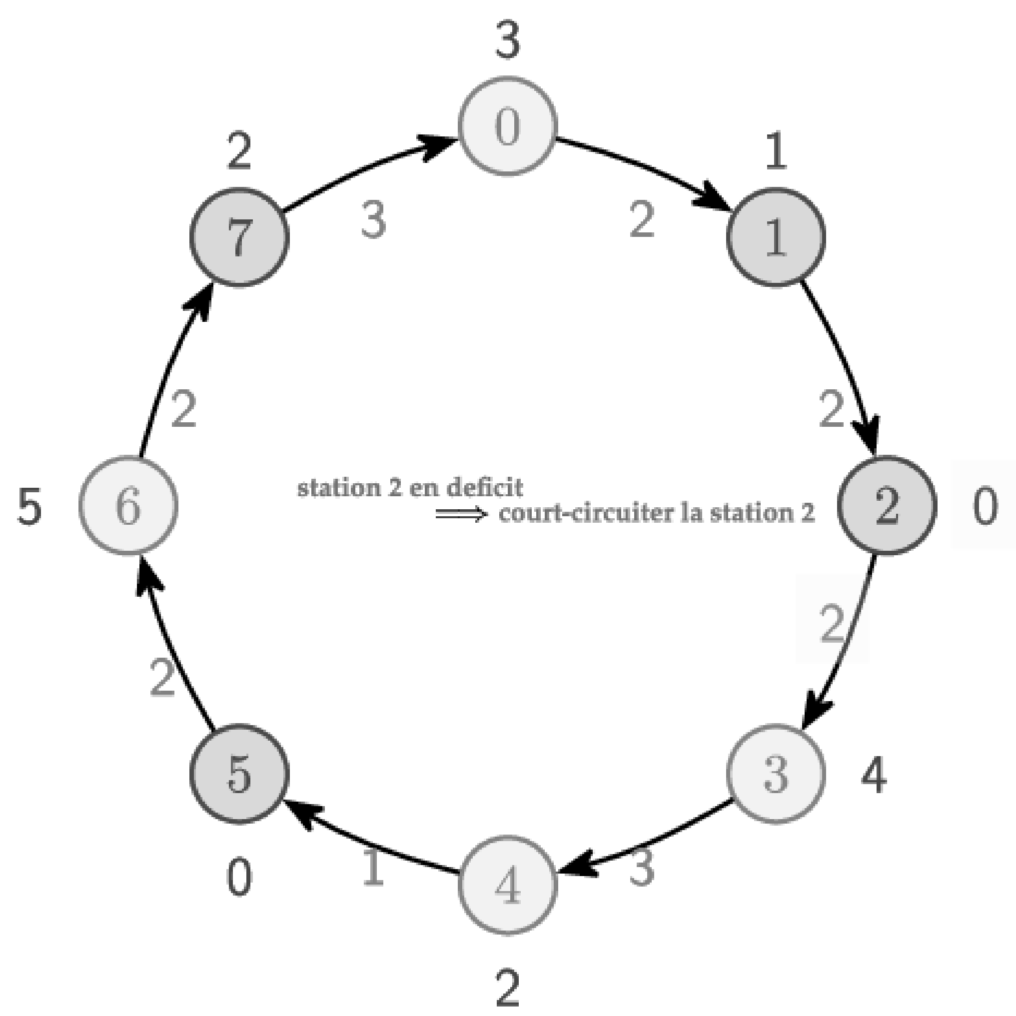
new idea

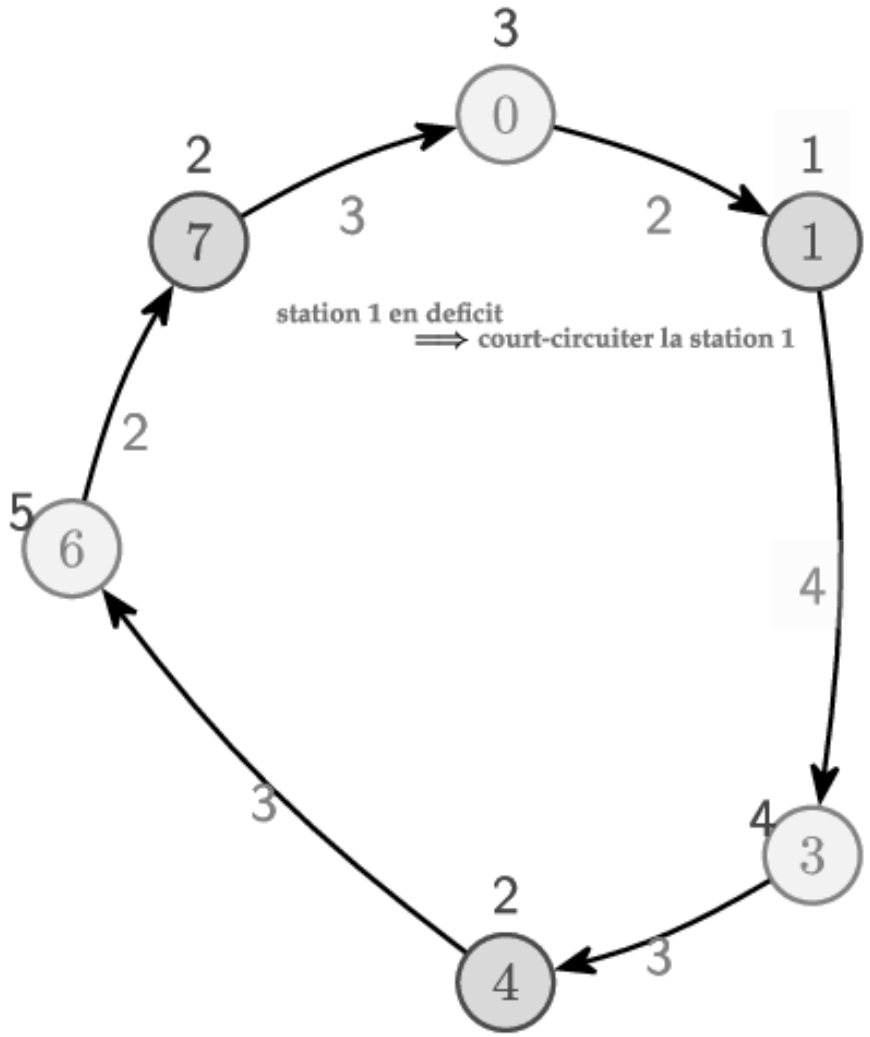
SOLUTION

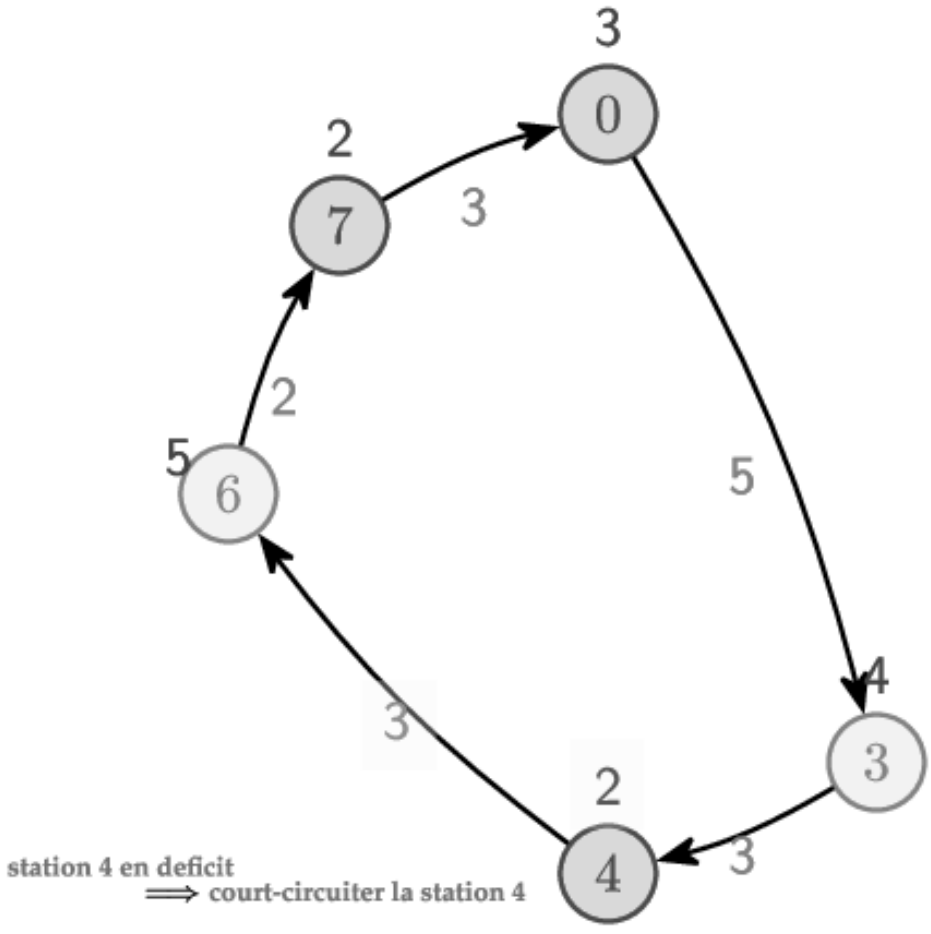
Puzzle Gas Stations (recursion)

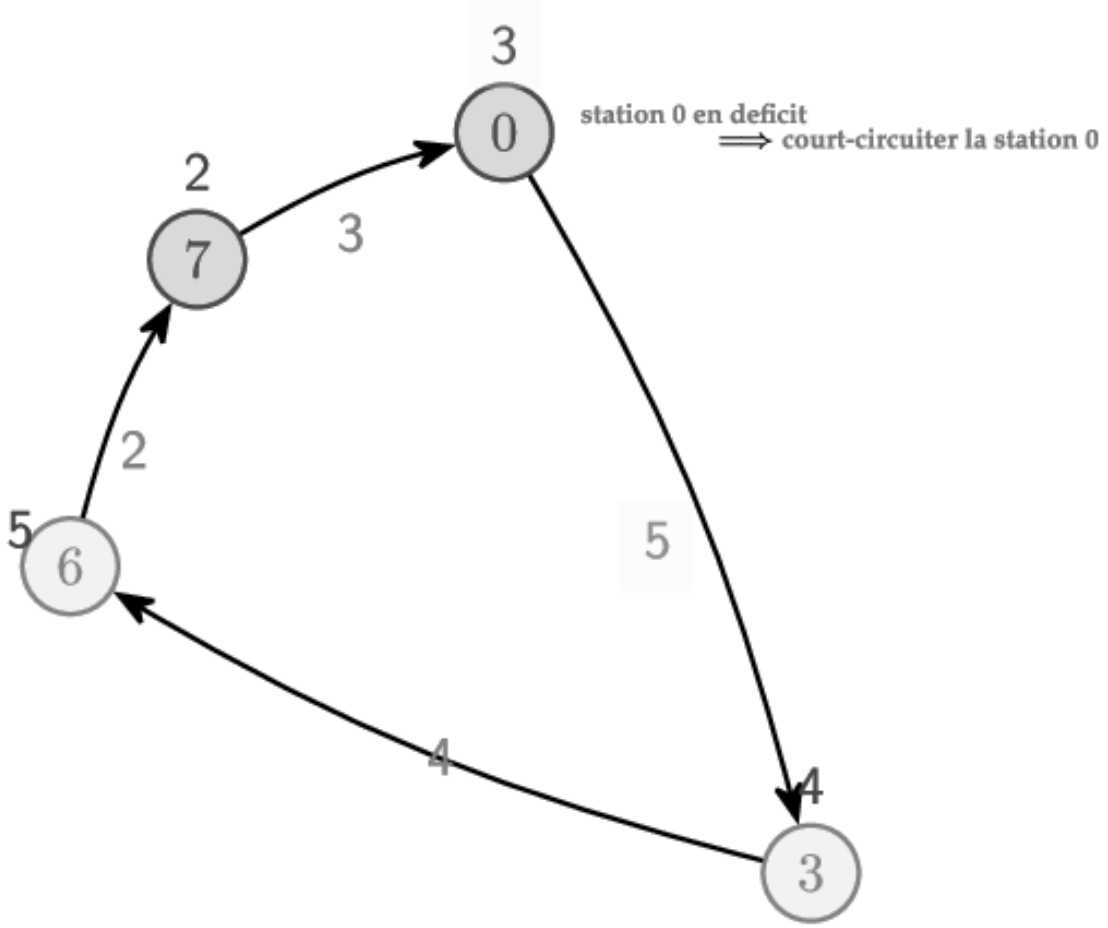
```
def find_valid_start(gas, cost):
    n = len(gas)

    if n == 1:
        return 0

    for j in range(0, n):
        if cost[j] > gas[j]:
            cost_p = cost[:j] + cost[j+1:]
            cost_p[j-1] = cost[j-1] + cost[j] - gas[j]

            gas_p = gas[:j] + gas[j+1:]

            sol = find_valid_start(gas_p, cost_p)
            if sol >= j: sol += 1
            return sol
    return 0
```

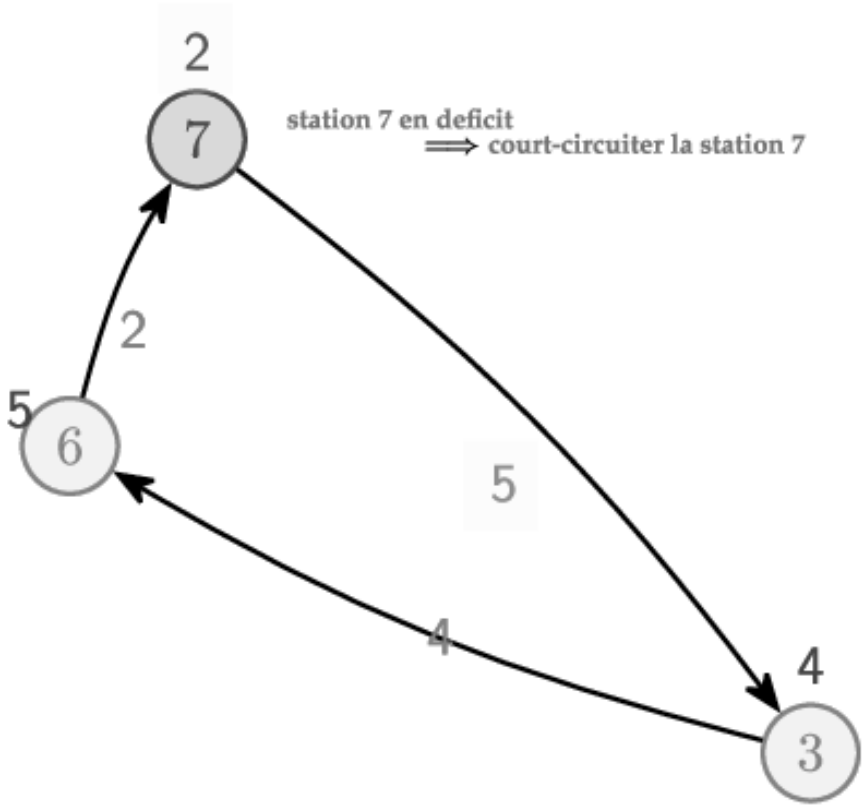


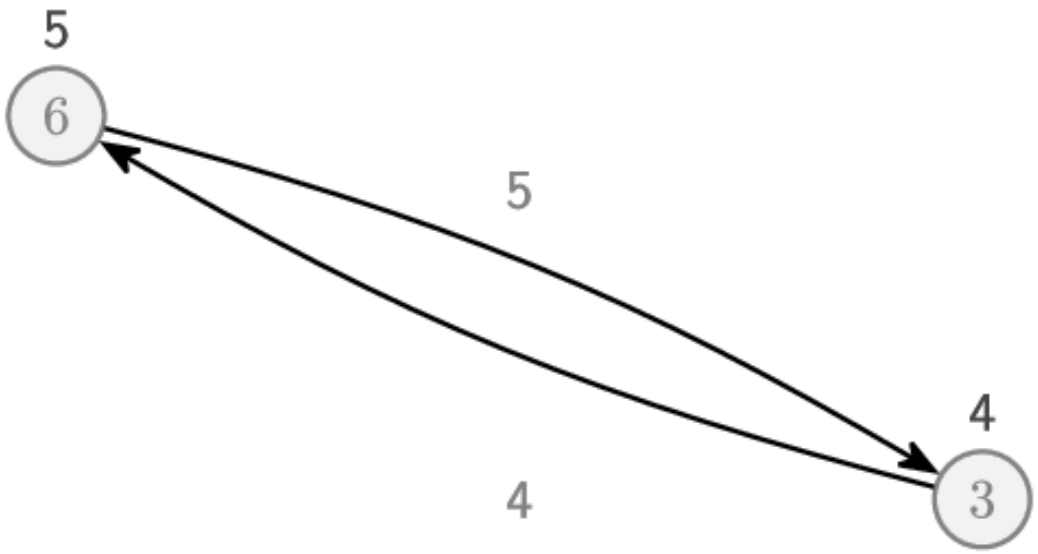

station 2 en deficit
⟹ court-circuiter la station 2


3
0
2
7
3
2
1
1
station 1 en deficit
⟹ court-circuiter la station 1
2
5
6
4
3
4
3
2
4
3


3
0
2
7
3
2
5
6
5
4
3
3
2
4
station 4 en deficit
⟹ court-circuiter la station 4


3
0
station 0 en deficit
⟹ court-circuiter la station 0
2
7
3
2
5
6
5
4
4
3


2
7
station 7 en deficit
⟹ court-circuiter la station 7
2
5
6
5
4
4
3



pas des deficits

$\Longrightarrow$

Donc on peut commencer à station 3 ou station 6!

new idea

SET

Set: une collection des éléments, **sans doublons ou ordre**

l'ordre n'a aucune importance

$$\{1, 5, `hello'\} == \{`hello', 5, 1\}$$

Pas de doublons

$$\{1, 5, 1, `hello'\} == \{`hello', 5, 1\}$$

Set: une collection des éléments, **sans doublons ou ordre**

```
A = {4, 7, 1, 5}

print(type(A))
print(A)
```

```
<class 'set'>
{1, 4, 5, 7}
```

Pour créer un nouveau `set` vide

```
A = set()

print(type(A))
print(A)
```

```
<class 'set'>
set()
```

Attention : ne faites pas A = { } ... ici, `A` n'est pas un set !

Convertir une liste à un set avec `set()`

```
A = [4, 7, 1, 5, 1]
B = set(A)

print(type(B))
print(B)
```

```
<class 'set'>
{1, 4, 5, 7}
```

Taille avec `len()`

```
A = {4, 7, 1, 5, 1}

print( len(A) )
```

```
4
```

Convertir un set à une list avec `list()`

```
A = {4, 7, 1, 5, 1}
B = list(A)

print(type(B))
print(B)
```

```
<class 'list'>
[1, 4, 5, 7]
```

List: `A = [ ..... ]`

Set: `A = { ..... }`

```
A = {4, 7, 1, 5}
B = [4, 7, 1, 5]

if A == B:
    print("Equal!")
else:
    print("No!")
```

No!

```
A = {4, 7, 1, 5}
B = [4, 7, 1, 5, 7]

if A == set(B):
    print("Equal!")
else:
    print("No!")
```

Equal!

new idea

SET OPÉRATIONS

Adhésion : `in`

```
A = {4, 7, 1, 5}

if 4 in A:
    print("4 in A.")

if 3 not in A:
    print("3 no way in A.")
```

```
4 in A.
3 no way in A.
```

Union : `|`

```
A = {4, 7, 1, 5}
B = {4, 8, 9}

print(A)
print(B)

C = A | B

print(C)
```

```
{1, 4, 5, 7}
{8, 9, 4}
{1, 4, 5, 7, 8, 9}
```

Intersection : &

```
A = {4, 7, 1, 5}
B = {4, 8, 9}

print(A)
print(B)

C = A & B

print(C)
```

```
{1, 4, 5, 7}
{8, 9, 4}
{4}
```

Difference : -

```
A = {4, 7, 1, 5}
B = {4, 8, 9}

C = A - B
D = B - A

print(C)
print(D)
```

```
{1, 5, 7}
{8, 9}
```

Nous ne pouvons pas utiliser les indices :

```
A = {4, 7, 1, 5}

print(A[0])
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 3, in <module>
    print(A[0])
          ~~~^
TypeError: 'set' object is not subscriptable
```

Set: parcourir les éléments en utilisant la boucle `for` :

```
A = {4, 7, 1, 5}

for x in A:
    print(x + 5)
```

```
6
9
10
12
```

```
A = {4, 7, 1, 5}

for i in range(0, len(A)):
    print(A[i] + 5)
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 4, in <module>
    print(A[i] + 5)
          ~~~^
TypeError: 'set' object is not subscriptable
```

Ajouter un élément: `add()`

```
A = {4, 7, 1, 5}

A.add("hello")

print(A)
```

```
{1, 4, 5, 7, 'hello'}
```

Supprimer un élément: `remove()`

```
A = {4, 7, 1, 5}

A.remove(7)
print(A)
```

```
{1, 4, 5}
```

```
A = {4, 7, 1, 5}

A.remove(8)
print(A)
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py",
   line 3, in <module>
    A.remove(8)
KeyError: 8
```

Supprimer un élément: `discard()`

```
A = {4, 7, 1, 5}

A.discard(7)
print(A)
```

```
{1, 4, 5}
```

```
A = {4, 7, 1, 5}

A.discard(8)
print(A)
```

```
{1, 4, 5, 7}
```

new idea

SET: accéder un élément

1. d'accéder à un élément d'un ensemble: `iter() + next()`

```
A = {1, 2, 3, 4, 5}

it = iter(A)

element = next(it)
print(element)

element = next(it)
print(element)
```

```
1
2
```

2. d'accéder à un élément d'un ensemble: Convertir en liste et indicer

```
A = {1, 2, 3, 4, 5}

element = list(A)[0]
print(element)

element = list(A)[1]
print(element)
```

```
1
2
```

3. d'accéder à un élément d'un ensemble: Utiliser min() ou max()

```
A = {1, 2, 3, 4, 5}

element = min(A)
print(element)

element = max(A)
print(element)
```

```
1
5
```

4. d'accéder à un élément d'un ensemble: Utiliser un boucle

```
A = {1, 2, 3, 4, 5}

for element in A:
    print(element)
```

```
1
2
3
4
5
```

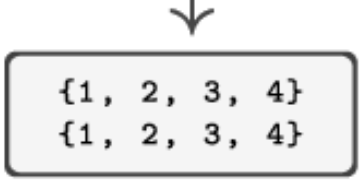
new idea

SET: OPÉRATEURS =

```
A = {6, 18, 30, 41, 48}
B = A
```

Un autre nom de la **même** set !

```
A = {1,2,3}

B = A

A.add(4)

print(A)
print(B)
```



```
{1, 2, 3, 4}
{1, 2, 3, 4}
```

Pour faire une copie distincte, utilisez `set()`

```
A = {1,2,3}

B = set(A)

A.add(4)

print(A)
print(B)
```



```
{1, 2, 3, 4}
{1, 2, 3}
```

new idea

SET vs LISTES

List: `A = [ ..... ]`

Set: `A = { ..... }`

Liste peut contenir

- int
- float
- bool
- string
- list
- set

Set peut contenir

- int
- float
- bool
- string

```
A = [1, 5, [3,6]]

print(A)
```

↓

```
[1, 5, [3, 6]]
```

```
A = {1, 5, [3,6]}

print(A)
```

↓

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 1, in <module>
    A = {1, 5, [3,6] }
         ^^^^^^^^^^^^^
TypeError: unhashable type: 'list'
```

new idea

TUPLE

Tuple: une collection des éléments, **avec ordre**

```
A = (4, 7, 1, 5)

print(type(A))
print(A)
print(len(A))
```

```
<class 'tuple'>
(4, 7, 1, 5)
4
```

Convertir une liste à un tuple avec `tuple()`

```
A = [4, 7, 1, 5]

B = tuple(A)

print(type(B))
print(B)
```

```
<class 'tuple'>
(4, 7, 1, 5)
```

List: `A = [ ..... ]` Tuple: `A = ( ..... )`

Différence entre `list` et `tuple` : impossible de modifier les tuples !

```
A = (6,1,34,5)

A[0] = 40
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 3, in <module>
    A[0] = 40
    ~~~~
TypeError: 'tuple' object does not support item assignment
```

Différence entre `list` et `tuple` : impossible de modifier les tuples !

```
A = (6,1,34,5)

A.append(50)
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 3, in <module>
    A.append(50)
    ^^^^^^^^
AttributeError: 'tuple' object has no attribute 'append'
```

Packing and unpacking tuples

```
A = (3,5,1,4)

a,b,c,d = A

print(a,b,c,d)
```

```
3 5 1 4
```

Lorsque vous définissez des tuples avec un seul élément :

```
A = (3)
print(type(A))
print(A)

B = (3,)
print(type(B))
print(B)
```

```
<class 'int'>
3

<class 'tuple'>
(3,)
```

Packing and unpacking tuples

```
A = (3,5,1,4)

a,b,c = A

print(a,b,c)
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 3, in <module>
    a,b,c = A
    ^^^^^
ValueError: too many values to unpack (expected 3)
```

Plusieurs valeurs sont renvoyées par les fonctions sont les tuples

```
def myfunction():
    return 0,5,13,14

A = myfunction()

print(type(A))
print(A)
```

```
<class 'tuple'>
(0, 5, 13, 14)
```

new idea

MODIFIABLE ITERABLES

List: `A = [ ..... ]`

Set: `A = { ..... }`

Liste peut contenir

int
float
bool
string
list
set
tuple

Set peut contenir

int
float
bool
string
tuple

Voici les propriétés générales que nous avons déjà croisées.

Un container peut être :

Ordonné

il y a un ordre précis des éléments ;
ce même ordre est utilisé lorqu'on itère dessus ;

Indexable

on peut retrouver un élément par son indice (i.e. sa position) ;
général, tout container indexable est ordonné ;

Itérable

un itérable est une structure de données permettant d'accèder à ses éléments un par un ; on peut faire une boucle dessus.

Non Modifiable (parfois dit 'Immuable')

une fois créé, Python ne permet plus de le modifier par la suite.

	Ordonné	`A[ i ]` Indexable	`for x in A` Itérable	`A[i] = 'a'` Modifiable
list:	✓	✓	✓	✓
string:	✓	✓	✓	✗
tuple:	✓	✓	✓	✗
set:	✗	✗	✓	✓

List: `A = [ ..... ]`

Set: `A = { ..... }`

contient uniquement des objets non modifiable!

Liste peut contenir

- int
- float
- bool
- string
- list
- set
- tuple

Set peut contenir

- int
- float
- bool
- string
- tuple