1- Les Tuples

2 - Les dictionnaires

3 - les ensembles

4- Les chaines de caractères

Les Tuples

Définition

- Python propose un type de données appelé ***tuple***, qui est assez semblable à une liste mais qui **n'est pas modifiable** (**immuables**).
- Du point de vue de la syntaxe, un tuple est une collection d'éléments séparés par des virgules :

```
>>> tup='a','z','e','r','t','y'
>>> tup
('a', 'z', 'e', 'r', 't', 'y')
>>> tup=('a','z','e','r','t','y')
>>> tup
('a', 'z', 'e', 'r', 't', 'y')
```

- Il est vivement conseillé de mettre le tuple en évidence en l'enfermant dans une paire de parenthèses
- Les opérations qu'on peut faire sur un Tuple sont identiques à ce que l'on effectue sur les liste excepté qu'un Tuple n'est pas modifiable

Exemple

```
>>> tup=('a','z','e','r','t','y')
>>> tup[1:3]
('z', 'e')
```

```
>>> tup1=('a','z','e','r','t','y')
>>> tup2=tup1[1:3]
>>> tup2
('z', 'e')
>>> tup3=tup2*3
>>> tup3
('z', 'e', 'z', 'e', 'z', 'e')
```

Un tuple n'est pas modifiable

```
>>> tup=('a','b','c','d')
>>> tup[0]='A'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    tup[0]='A'
TypeError: 'tuple' object does not support item assignment
```

Les dictionnaires

Les dictionnaires

- Les dictionnaires sont un type composite, semblables aux listes dans une certaine mesure (ils sont modifiables comme elles), mais ce ne sont pas des séquences
- Un dictionnaire est composé de paires (clé, valeur)
- La clé permettra d'accéder à la valeur
- Une clé pourra être:
  - Alphabétique
  - Numérique
  - Un type composite
- Les valeurs associées à ces clés peuvent êtres de n'importe quel type

Création d'un dictionnaire

- Puisque le type dictionnaire est un type modifiable, nous pouvons commencer par créer un dictionnaire vide, puis le remplir petit à petit.
- Du point de vue syntaxique, on reconnaît une structure de données de type dictionnaire au fait que ses éléments sont enfermés dans **une paire d'accolades**
- **Exemple:** **fiche**

num	nom	prenom	Date_naissance
12345678	Ziad	Lina	12-05-1989

```
>>> fiche={}
>>> fiche['num']=12345678
>>> fiche['nom']='Ziad'
>>> fiche['prenom']='Lina'
>>> fiche['date_naissance']='12-05-1989'
>>> fiche
{'prenom': 'Lina', 'num': 12345678, 'date_naissance': '12-05-1989', 'nom': 'Ziad'}
>>> fiche['nom']
'Ziad'
>>> fiche['date_naissance']
'12-05-1989'
>>> fiche ['prenom'] #accès à un élement
'Lina'
```

Remarque

- Chaque clé d'un dictionnaire doit être unique

```
>>> ang_fr={'one':'un','two':'deux','three':'trois'}
>>> ang_fr
{'one': 'un', 'two': 'deux', 'three': 'trois'}
>>> ang_fr={'one':'un','two':'deux','three':'trois','one':'Un'}
>>> ang_fr
{'one': 'Un', 'two': 'deux', 'three': 'trois'}
```

Opérations sur les dictionnaires

```
>>> fiche['nom']='Ziadi' # modifier un élément
>>> fiche
{'prenom': 'Lina', 'num': 12345678, 'date_naissance': '12-05-1989', 'nom': 'Ziadi'}
>>> len(fiche) # le nombre d'élément
4
>>> del(fiche['num']) #supprimer un élément
>>> fiche
{'prenom': 'Lina', 'date_naissance': '12-05-1989', 'nom': 'Ziadi'}
>>> fiche['filière']='GI' # ajouter un élément
>>> fiche
{'prenom': 'Lina', 'date_naissance': '12-05-1989', 'nom': 'Ziadi', 'filière': 'GI'}
```

les méthodes spécifiques aux dictionnaires

- **keys()** :renvoie la liste des clés utilisées dans le dictionnaire :

```
>>> fiche={}
>>> fiche['num']=12345678
>>> fiche['nom']='Ziad'
>>> fiche['prenom']='Lina'
>>> fiche['date_naissance']='12-05-1989'
>>> fiche.keys()
dict_keys(['prenom', 'num', 'date_naissance', 'nom'])
```

- **values() :renvoie la liste des valeurs mémorisées dans le dictionnaire :**

```
>>> fiche.values()
dict_values(['Lina', 12345678, '12-05-1989', 'Ziad'])
```

- **items()** : extrait du dictionnaire une liste équivalente de tuples :

```
>>> fiche.items()
dict_items([('prenom', 'Lina'), ('num', 12345678), ('date_naissance', '12-05-1989'), ('nom', 'Ziad')])
```

les méthodes spécifiques aux dictionnaires

- **copy()** :permet d'effectuer une vraie copie d'un dictionnaire

```
>>> fiche2=fiche.copy()
>>> id(fiche2),id(fiche)
(48812240, 48812080)
```

- À ne pas confondre avec une affectation

```
>>> fiche3=fiche
>>> id(fiche3),id(fiche)
(48812080, 48812080)
```

Parcours d'un dictionnaire

- Le parcours d'un dictionnaire pose quelques difficultés. Comme ce n'est pas une séquence, il n'y a pas de relation d'ordre pré-établie entre les éléments afin de parcourir toutes les données d'un dictionnaire.

```
>>> fiche ={"num":12345678, "nom":'ziad', "prenom":'lina'}
>>> for cle in fiche:
        print (cle)


prenom
num
nom
```

- Pour effectuer un traitement sur les valeurs, il suffit de récupérer chacune d'elles à partir de la clé correspondante

```
>>> for cle in fiche:
        print (cle,fiche[cle])


prenom lina
num 12345678
nom ziad
```

Parcours d'un dictionnaire

- Il est recommandé de faire appel à la méthode **items()**

```
>>> for cle,valeur in fiche.items():
        print (cle,valeur)


prenom lina
num 12345678
nom ziad
```

Exercice

- Ecrire un programme qui demande à l'utilisateur de saisir un texte , ensuite il affiche le nombre d'occurrence de chaque caractère dans le texte, par exemple:

```
Tapez votre texte : Ecole nationale des sciences appliquées kénitra
{'E': 1, 'c': 3, 'o': 2, 'l': 3, 'e': 6, ' ': 5, 'n': 4, 'a': 4, 't': 2, 'i': 4, 'd'
: 1, 's': 4, 'p': 2, 'q': 1, 'u': 1, 'é': 2, 'k': 1, 'r': 1}
```

Exécution

```
 RESTART: /Users/oumaira/Documents/Python 2019/cours 2020/exemples/histogram.py
Tapez votre texte : Python is an interpreted, high-level, general-purpose programmin
g language. Created by Guido van Rossum and first released in 1991, Python's design
philosophy emphasizes code readability with its notable use of significant whitespac
e. Its language constructs and object-oriented approach aim to help programmers writ
e clear, logical code for small and large-scale projects.
{'P': 2, 'y': 5, 't': 20, 'h': 11, 'o': 21, 'n': 19, ' ': 49, 'i': 22, 's': 22, 'a':
28, 'e': 35, 'r': 21, 'p': 13, 'd': 12, ',': 4, 'g': 13, '-': 4, 'l': 17, 'v': 2, 'u
': 7, 'm': 8, '.': 3, 'C': 1, 'b': 4, 'G': 1, 'R': 1, 'f': 4, '1': 2, '9': 2, "'": 1
, 'z': 1, 'c': 12, 'w': 3, 'I': 1, 'j': 2}
```

Les ensembles

Les ensembles (set)

- Un ensemble est une collection non ordonnée d'éléments.
- Chaque élément est unique (pas de doublons) et doit être immuable (ce qui ne peut pas être changé).

Cependant, l'ensemble lui-même est mutable. Nous pouvons ajouter ou supprimer des éléments.

Création d'un ensemble

- Un ensemble est créé en plaçant tous les éléments (éléments) entre accolades **{},** séparés par une virgule ou en utilisant la fonction intégrée **set()**.
- Il peut avoir n'importe quel nombre d'éléments et ils peuvent être de types différents (int, float, tuple, string, etc.). Mais un ensemble ne peut pas avoir un élément mutable, comme une liste, un ensemble ou un dictionnaire.

Exemple

```python
# création d'un ensemble de 4 éléments
e1={1,3,5,7}
print("le type de e1 est : ",type(e1))
print("les éléments de e1 sont : ",e1)

e2={"Mostafa", 1.78, 32 }
print("e2 : ",e2)

# ensemble avec doublons
e3 = {3, 5, 7, 2, 3, 5}
print("e5 : ", e5)
```

```
le type de e1 est : < class 'set'>
les éléments de e1 sont : {1, 3, 5, 7}
e2 : {32, 1.78, 'Mostafa'}
e3 : {2, 3, 5, 7}
```

Différence entre liste et ensemble

- Chaque élément à l'intérieur d'un ensemble doit être unique.
- Les éléments d'un ensemble ne sont stockés dans aucun ordre particulier.

Remarque:

- Si votre application ne se soucie pas de la façon dont les éléments sont stockés, utilisez des ensembles plutôt que des listes, car en ce qui concerne la recherche des éléments, un ensemble est beaucoup plus efficace qu'une liste.

Création d'un ensemble

- Nous pouvons également utiliser la fonction intégrée **set()** pour créer des ensembles.

```
1  # créer un ensemble à partir d'un ensemble
2  e1 = set({77, 23, 91, 271})
3  print("e1 : ",e1)
4
5  #créer un ensemble à partir d'une chaîne
6  e2 = set("123abc")
7  print("e2 : ",e2)
8
9  # créer un ensemble à partir d'une liste
10 e3 = set(['Mostafa', 'Sedoki', 'Meknes', 32, 1.78])
11 print("e3 : ",e3)
12
13 #créer un ensemble à partir d'un tuple
14 e4 = set(("Meknes", "Marrakech", "Essaouira"))
15 print("e4 : ",e4)
```

```
e1 : {91, 271, 77, 23}
e2 : {'3', 'b', '1', '2', 'a', 'c'}
e3 : {32, 1.78, 'Sedoki', 'Mostafa', 'Meknes'}
e4 : {'Marrakech', 'Essaouira', 'Meknes'}
```

Modifier un ensemble

- Les ensembles sont mutables. Mais comme ils ne sont pas ordonnés, l'indexation n'a pas de sens.
- Nous ne pouvons pas accéder à un élément d'un ensemble ni le modifier à l'aide de l'indexation ou du slicing. Set ne le supporte pas.
- Nous pouvons ajouter un seul élément à l'aide de la méthode **add()** et plusieurs éléments à l'aide de la méthode **update().** La méthode **update()** peut prendre pour argument des tuples, des listes, des chaînes ou d'autres ensembles. Dans tous les cas, les doublons sont évités.

Modifier un ensemble

Exemple 3 :

```
1  e={1,3}
2
3  # ajouter un élément
4  e.add(5)
5  print("e : ",e)
6
7  # ajouter une liste
8  e.update([9,4,7])
9  print("e : ",e)
10
11 # ajouter une liste et un ensemble
12 e.update([50,51],{100,200})
13 print("e : ",e)
```

```
e : {1, 3, 5}
e : {1, 3, 4, 5, 7, 9}
e : {1, 3, 4, 5, 100, 7, 200, 9, 50, 51}
```

Supprimer un élément d'un ensemble

- Un élément particulier peut être supprimé de la série à l’aide des méthodes, **discard()** et **remove().**

  La seule différence entre les deux est que,en utilisant **discard()** si l'élément n'existe pas dans l'ensemble, il reste inchangé. Mais **remove()** lève une erreur dans une telle condition.

Exemple

Exemple 4 :

```
1  e = {1, 3, 6, 7}
2
3  e.discard(3)
4  print("e : ", e)
5
6  e.remove(7)
7  print("e : ", e)
8
9  e.discard(9)
10 print("e : ", e)
11
12 e.remove(9)
13 print("e : ", e)
```

```
e : {1, 6, 7}
e : {1, 6}
e : {1, 6}
Traceback (most recent call last):
File "prog.py", line 12, in < module>
e.remove(9)
KeyError: 9
```

Union et intersection

```
A = {0, 2, 4, 6, 8};
B = {1, 2, 3, 4, 5};
print("Union :", A | B)

print("Intersection :", A & B)

print("Difference :", A - B)
```

```
Union : {0, 1, 2, 3, 4, 5, 6, 8}
Intersection : {2, 4}
Difference : {0, 8, 6}
```

Recommandation

- Si vous voulez faire un test d'appartenance sur une séquence (liste par exemple), convertissez cette séquence en set et faites le test d'appartenance sur ce nouveau set.

Les chaines de caractères

Chaînes de caractères

- Une chaine de caractères est une structure de données non modifiable

```
>>> message = 'bonjour à tous'
>>> message[0] = 'B'
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    message[0] = 'B'
TypeError: 'str' object does not support item assignment
```

```
>>> message='bonjour'
>>> message='B'+message[1:]
>>> message
'Bonjour'
```

- On peut modifier la valeur de la variable avec une affectation, mais pas l'un de ses caractères individuellement
- Nous ne modifions pas la chaîne message. Nous en re-créons une nouvelle avec le même nom à la deuxième ligne du script (à partir d'un morceau de la précédente : il s'agit bien d'une nouvelle chaîne

Code ASCII en python

- La fonction **ord(ch)** accepte n'importe quel caractère comme argument. elle retourne le code ASCII correspondant à ce caractère.
- La fonction **chr(num)** fait exactement le contraire. L'argument qu'on lui transmet doit être un entier compris entre 0 et 255. En retour, on obtient le caractère ASCII correspondant

```
>>> ord('a')
97
>>> ord('z')
122
>>> chr(97)
'a'
>>> chr('122')
```

Les chaînes sont comparables

- Tous les opérateurs de comparaison fonctionnent aussi avec les chaînes de caractères
- les caractères forment une suite bien ordonnée dans le code ASCII, on peut donc les comparer
- Exemple: Ecrire une fonction qui renvoie « vrai » si le paramètre qu'on lui passe est une minuscule :

```
>>> def minuscule(ch):
    if 'a' <= ch <= 'z' :
        return True
    else:
        return False

>>> minuscule('B')
False
>>> minuscule('z')
True
```

Opérations sur les chaines de caractères

- Len(): retourne la taille de la chaine
- +: Concaténation
- *: Répétition

```
>>> s='azerty'
>>> len(s)
6
>>> s1,s2='Bonjour','les Ensakistes'
>>> s=s1+s2
>>> s
'Bonjourles Ensakistes'
>>> s='stop'
>>> s5=s*4
>>> s5
'stopstopstopstop'
```

Méthodes des chaines de caractères

- On peut classer les méthodes en plusieurs catégories :
  - Gestion des espaces
  - Gestion des majuscules et minuscules
  - Test de la nature d'une chaine
  - Recherche et remplacement
  - Découpage et collage de chaînes
- Une nouvelle chaine est créée et renvoyée par les méthodes
- **L'appel à une méthode ne modifie pas la chaine en elle-même**

Gestion des majuscules et des minuscules

Méthode	Effet
upper()	Remplace les minuscules par des majuscules
lower()	Remplace les majuscules par des minuscules
capitalize()	Passe la 1ère lettre de la chaine en majuscule et les autres en minuscules
title()	Formate comme un titre (1ère lettre de chaque mot en majuscule, les suivantes en minuscules)
swapcase()	Inverse les majuscules et minuscules

Exemples

```
>>> s="école nationale des sciences appliquées"
>>> s1=s.upper()
>>> s1
'ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES'
>>> s2=s1.lower()
>>> s2
'école nationale des sciences appliquées'
>>> s3=s.capitalize()
>>> s3
'École nationale des sciences appliquées'
>>> s4=s3.swapcase()
>>> s4
'éCOLE NATIONALE DES SCIENCES APPLIQUÉES'
>>> s5=s.title()
>>> s5
'École Nationale Des Sciences Appliquées'
```

Gestion des espaces

Méthode	Effet
strip()	Supprime les espaces en début et fin de chaine
rstrip()	Supprime les espaces en fin de chaine
lstrip()	Supprime les espaces en début de chaine
isspace()	Renvoie True s'il n'y a que des espaces

```
>>> s1=' bonour tout le monde  '
>>> s2=s1.strip()
>>> s2
'bonour tout le monde'
```

Tests sur la nature de la chaîne

Méthode	Effet
startswith(prefix)	True si la chaine commence par prefix
endswith(prefix)	True si la chaine se termine par prefix
isupper()	True si la chaine ne contient que des majuscules
islower()	True si la chaine ne contient que des minuscules
istitle()	True si la chaine est formatée comme un titre
isdigit()	True si la chaine ne contient que des caractères numériques
isalpha()	True si la chaine ne contient que des caractères alphabétiques
isalnum()	True si la chaine ne contient que des caractères alphanumériques

```
>>> s='1234'
>>> s1=s.isdigit()
>>> s1
True
```

Recherche et remplacement

Méthode	Effet
index(sub)	Renvoie l'index de la 1ère occurrence de la chaine sub, exception si pas trouvée
rindex()	Idem pour la dernière occurrence
find(sub)	Comme index mais renvoie -1 si la chaine n'est pas trouvée
rfind()	Idem pour la dernière occurrence
count(sub)	Retourne le nbr d'occurrence de la sous-chaine
replace(old, new [,max])	Remplace la sous-chaine old par new, au plus max fois (par défaut, remplace toutes les occurrences)

Exemples

```
>>> s="école nationale des sciences appliquées"
>>> s.index('e')
4
>>> s.rindex('e')
37
>>> s.find('z')
-1
>>> s.index('z')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    s.index('z')
ValueError: substring not found
>>> s.rfind('e')
37
>>> s.count('e')
6
>>> s.replace('é','E')
'Ecole nationale des sciences appliquEes'
>>> s.replace('é','E',1)
'Ecole nationale des sciences appliquées'
```

Découpage et collage

Méthode	Effet
split(sep)	Crée une liste de tous les mots de la chaîne en utilisant sep comme séparateur
Lien.**join**(list)	Fusionner les membres d'une liste avec le mot de liaison **lien**

```
>>> saisons='été,hiver,automne,printemps'
>>> saisons.split(',')
['été', 'hiver', 'automne', 'printemps']
>>> saisons='été:hiver:automne:printemps'
>>> saisons.split(':')
['été', 'hiver', 'automne', 'printemps']
```

```
>>> liste=['plate','forme']
>>> '-'.join(liste)
'plate-forme'
```

Exemple

```
>>> s='12/03/2019'
>>> s=s.split('/')
>>> s
['12', '03', '2019']
>>> s='-'.join(s)
>>> s
'12-03-2019'
```

Accès et extraction

Accès

- s="python"
- s[0] → 'p'
- s[5] → 'n'
- s[-1] → 'n'
- s[-3] → 'h'

Extraction

- s[1:5] → 'ytho' # 5 non compris
- s[-3:] → 'hon' #-3 compris jusqu'à la fin
- s[2:] → 'thon' #2 compris jusqu'à la fin