Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques de Monastir
Département Sciences de l'Informatique
Ingénieur en Informatique -I1 – 2023/2024

Hands on training
Structures de données avancées en Python

Enseignant du cours : Mariem Gzara
Hands on training préparé par : Mariem Gzara (MC Info ISIMM) et Azza Chebil (Doctorant Info)

Objectif du "Hands on training" : L'objectif de cette activité est de manipuler les principales structures de données de base utilisées en programmation python et de comprendre les situations dans lesquelles ellessont utilisées.

Partie 1 : tuples (n-uplets)

- Séquence immuable hétérogène :
  - Les objets immuables (ou « immutables ») ne peuvent pas être directement modifiés. Parmi les objets immuables, on retrouve notamment :
    - Les nombres entiers (int)
    - Les nombres décimaux (float)
    - Les chaînes de caractères (str)
    - Les booléens (bool)
    - Les tuples (tuple)
- Entouré de parenthèses plutôt que de crochets

Création de tuples, création de tuple avec 0 ou bien un seul élément, suppression d'un tuple, indexation négative dans un tuple, modifier un tuple, tuples imbriqués, affecter tuple à objet, affecter les éléments d'un tuple aux objets, utilité des tuples.

Question 1 : Saisir les commandes suivantes et observer le résultat.

```
a = (1.85,75,"homme")
type(a)
print(a)
print(type(a))
print(len(a))
print(a[0])
print(a[len(a)-1])
b=(1.62,80,"femme")
#tuples imbriqués
groupe=(a,b,(1.70,90,"homme"))
print(groupe)
# virgule de fin obligatoire pour tuple à 1 seul élément
seul=(1,) print(seul)
print(type(seul))
unique=(2)
print(type(unique))
print(unique)

#Les parenthèses ne sont pas obligatoires
c=12,15.25,"bonjour"
print(c)
print(type(c))
```

```
prenom,moyenne=("Wiem",12.45)
print(prenom)
print(moyenne)
#création de tuple à partir d'une liste
li=[15,12,4,7]
print(li)
li=tuple(li)
print(li)
#création de tuple à partir de chaines de caractères
cours="Machine Learning"
print(cours)
cours=tuple(cours)
print(cours)
```

Question 2 : comment créer un tuple vide ou bien à une valeur ?

- Notez que dans le cas où on souhaite créer un tuple vide, on utilisera une paire de parenthèses vides.
- Si on souhaite créer un tuple avec une seule valeur, alors il faudra faire suivre cette valeur d'une virgule.

Question 3 : Comment itérer sur les éléments d'un tuple ?

- Comme une liste, il est possible de parcourir un tuple avec une boucle **for**.

Question 4 : Que veut dire déballage de séquence ou aussi déballer un tuple dans plusieurs variables.

Question 5 : Exécuter et interpréter le cas d'un tuple contenant une liste.

```
events = ('29/05/2019', ['anniversaire', 'soirée'])
events[1].append('rdv coiffeur')
events
```

```
('29/05/2019', ['anniversaire', 'soirée', 'rdv coiffeur'])
```

Partie 2 : Les listes

- Série de valeurs de même type ou de types différents

Question 1 : Saisir les commandes suivantes et observer le résultat.

```
#especes des fleurs iris especes=["virginica","setosa","versicolor"]
print(type(especes))
print(especes)
print(especes[0])
print(especes[2])
print(len(especes))
#largeur et longeur des sépales et des pétales
sepal=[0.45,12.3]
print(sepal)
petal=[1.45,14.5]
print(petal)
#les attributs de iris
iris=sepal+petal print(iris)
iris.append(especes[1])
print(iris)
#initialisation d'une liste vide
a=[]
a=a+[6]
print(a)
```

```
a.append(8)
print(a)
#ATTENTION : indices positifs et négatifs d'une liste
scores=[45,62,25,75,99,87,12,15,56,68]
print(len(scores))
scores[0] scores[5]
scores[len(scores)-1]
print(scores)
print(scores[len(scores)-1],scores[-1])
print(scores[len(scores)-2],scores[-2])
print(scores[len(scores)-3],scores[-3]) #!!!!!!!!!!
print(scores[0],scores[-len(scores)]) print(25 in scores)
print(71 in scores)
print(scores!=[])
print([1,2,3]==[1,2,3])
print(max(scores),min(scores),sum(scores)/len(scores))
```

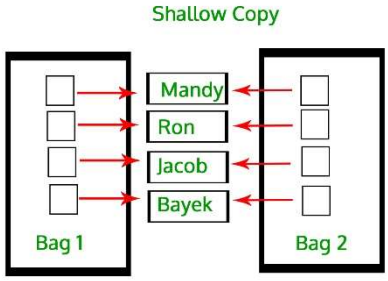
Question 2 : Que fait chacune des fonctions suivantes. Certaines instructions génèrent des erreurs. Expliquer pourquoi.

```
scores.count(75)
scores.count(32)
scores.index(25)
scores.index(75)
scores.index(63)
scores.index(75, 4)
scores.reverse()
scores.append(84)
scores.sort() scores.pop()
matrix = [ [1, 2, 3, 4],[5, 6, 7, 8],[5,8,2,14]]
```

Question 3 : Quelle est la différence entre copy et deepcopy ?

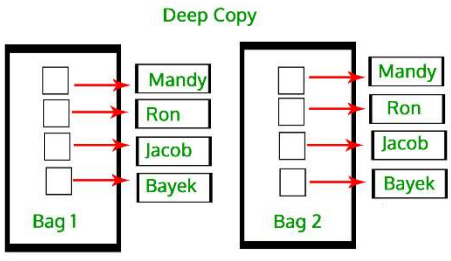
Copy()

- Créer un nouvel objet composé, puis référence les objets contenus dans l'original ➔ construit un nouvel objet de collection, puis le remplit avec des références aux objets enfants trouvés dans l'original
- Toutes les modifications apportées à une copie d'un objet sont reflétées dans l'objet d'origine



Deepcopy()

- Sert à créer un nouvel objet composé avant d'y insérer des copies des éléments trouvés dans l'objet original d'une manière récursive . Une copie de l'objet est copiée dans un autre objet. Toute modification apportée à une copie de l'objet ne se reflète pas dans l'objet d'origine.



Partie 3 : Les dictionnaires

- Chaque élément se compose d'une paire *clé: valeur*

- Objets muables et dynamiques :
  - Un objet muable (aussi appelé « mutable ») est un objet qui peut être modifié.
  - Des exemples d'objets muables incluent des listes, des dictionnaires et des ensembles.
- Collections d'objets non ordonnées
- Contenir des objets de n'importe quel type
- Inclure d'autres dictionnaires

Question 1 : Saisir les commandes suivantes et observer le résultat.

```
"""
En théorie
d = {
            clé: valeur,
            clé: valeur,
            clé: valeur,
              ...
           clé: valeur
}
"""


# Dictionnaire vide
dictio = {} print(dictio)
di=dict()
# Dictionnaire dont les clés ne sont que des chaînes de caractères
personne = {
            "prenom": "Mira",
            "nom": "Haddad",
            "age": 20
  }
print(personne)
print(type(personne)) #accés par clé
print(personne["age"])
print(personne.get("age"))
print(personne.get("profession")) # None#ajout
suppression des éléments
personne["profession"]="Developpeur"
personne["age"]=28
# Ou la valeur par défaut que l'on passe en deuxième argument print(personne.get("profession",
"Cette clé n'existe pas...")) # "Cetteclé n'existe pas..."
print(personne.values())
print(personne.keys())
print(personne.items())
# Dictionnaire dont les clés sont des objets de différents types
patient= {
            1: "0000205xxk","jour": "lundi",
            ("temp","sang","tension"): (38.2,"A+",14.3),4.5: "admission
            4 heures 30 du matin"
  }
print(patient)
#création de dictionnaire
a = {}
a["nom"] = "Rayan"
a["prenom"] = "Chouchen"
```

```
print(a)
```

Question 2 : Comment vérifier la présence d'une clé dans un dictionnaire ?

- Utiliser la méthode haskey pour vérifier la présence d'une clé qu'on veut chercher : nomdict.has_key("nom_key")
- Le moyen préféré et le plus rapide de vérifier l'existence d'une clé dans un dictionnaire est avec le `in` opérateur. Il revient `True` si la clé est dans le dictionnaire, `False` Par ailleurs.

```
dictionary = {'A': 1, 'B': 2, 'C': 3}
key = 'B'
if key in dictionary:
  print("Key", key, "exists in the dictionary")
else:
  print("Key doesn't exist in the dictionary")
```

- L'opérateur in implémente la fonction intégrée `__contains__`. Cependant, il est déconseillé d'appeler `__contains__` directement.
- Une autre approche courante pour rechercher une clé dans un dictionnaire consiste à utiliser la `get()` fonction. Cela renvoie la valeur de la clé si elle est trouvée ; sinon, la valeur spécifiée ou la valeur par défaut `None` est retourné.

```
dictionary = {'A': 1, 'B': 2, 'C': 3}
key = 'B'
value = dictionary.get(key)
if value:
  print("Key", key, "exists in the dictionary")
else:
  print("Key doesn't exist in the dictionary")
```

Question 3 : Comment supprimer une clé d'un dictionnaire.

- L'instruction **del** efface des éléments d'un dictionnaire en fonction de leur clé.
- La méthode **clear** efface tous les éléments d'un dictionnaire.
- Utiliser la fonction **dict.pop().** L'avantage de cette méthode est qu'elle permet de spécifier la valeur par défaut à retourner si la clé demandée n'existe pas dans le dictionnaire.

Question 4 :

- Créer un dictionnaire puis tester les 3 méthodes de suppression d'une clé d'un dictionnaire.
- Essaye de supprimer un élément par une clé qui n'est pas présente dans le dictionnaire.

Question 5 : créer deux dictionnaires de votre choix puis comparer les.

Question 6 : Donner différentes méthodes pour supprimer un élément du dictionnaire

```
d={'server': 'isim', 'uid': 'ig', 'database': 'ingenieur', 42: 'algo', 'retrycount': 3}
del d[42]
print(d)
d.clear()
print(d)
```

Question 7 : Comment supprimer un dictionnaire

Question 8 : saisir les commandes suivantes et observer les résultats

```
dict_1={"a":2,"b":3,"c":5}
dict_2={"h":14,"e":5.24}
#fusion de dict print(dict_1|dict_2)
print({**dict_1,**dict_2})
# Merge dictionaries using update()
dict_1.update(dict_2) print(dict_1)
# Merge dictionaries using dict()
d0 = {"a": 0, "b": 1}
```

```
d1 = {"b": 2, "c": 3}
d2 = dict(d0, **d1)
print(d2)
#vérifier l'existence d'une clé
print("a" in dict_1)
#supprimer une clé print(dict_1.pop('c', 'Missing
Key'))# loop over keys
my_dict={"a":4.5,"g":6.2,"k":14.2,"r":15}
for key in my_dict: print(key,
    my_dict[key])
# loop over keys
for key in my_dict.keys():print(key)
# loop over values
for value in my_dict.values():
    print(value)
# loop over keys and values
for key, value in my_dict.items():print(key,
    value)
```

Partie 4 : Les ensembles

- Set (ou ensemble en français) est une collection d'éléments non ordonnée.
- Eléments de type différent
- Pas de doublons
- Non immuable (valeurs non modifiées)
- Un set est modifiable : supprimer/ajouter des éléments
- Effectuer des opérations mathématiques comme l'union, l'intersection, la différence symétrique, etc.

Question 1 : Saisir les commandes suivantes et observer le résultat.

```
#initialiser un set
vrac={"cahier",12,15.3,True}
mySet = {"A", "B", "C"}
print(mySet)
print(type(mySet))
#longeur d'un set
print(len(mySet))
#ajouter un élément à un set
mySet.add("D")
print(mySet)
#ajoute plusieurs éléments à un Set
mySet.update(["E","F"]) print(mySet)
#ajouter un élément déjà dans le set
#observez le résultat: pas de doublons dans un set
mySet.add("A")
print(mySet)
#vérifier la présence d'un élément dans le set
print("A" in mySet)
print("h" in mySet)
#parcours d'un set
for element in mySet:
    print(element)
#supprimer un élément du set
mySet.remove("A")
```

```
print(mySet)

"""
#ATTENTION: tenter de supprimer un élément non présent
#keyError
mySet.remove("A")
print(mySet)
discard supprime un élément du set mais ne déclenche
pas une erreur si l'élémentn'est pas présent
"""
mySet.discard("A")
print(mySet) #supprimer un
élémentelem=mySet.pop()
print(elem)
#max de set si éléments comparables
print(max(mySet)) print(min(mySet))
#copier un set
myNewSet=mySet.copy()
print(myNewSet==mySet)
myNewSet.add("R")
print(myNewSet==mySet)
#vider le set mySet.clear()
print(mySet)
print(type(mySet))
del mySet
```

Question 2 : Quelle est la différence entre les méthodes suivantes : remove, display, pop

- Pop = supprime aléatoirement un element de l'ensemble
- Remove, discard = supprime les valeurs ou l'objet de l'ensemble à l'aide de la valeur

Question 3 : Quelle est la différence entre les méthodes clear et del

- Clear = supprimer tous les éléments de l'ensemble = set vide
- Del = supprimer l'ensemble définitivement de la mémoire, devient non définie

Question 4 : Quelle est la différence entre ces deux instructions : a={} et a=set()

- Il n'y a pas de différence, les deux instructions permettent de créer un ensemble (set) vide

Question 5 : y a-t-il une erreur ? laquelle ? corriger

```
mon_ensemble = {1, 2, 2, 3, 3, 3, 4, 4, 4, 4}
print(mon_ensemble)
# affiche {1, 2, 2, 3, 3, 3, 4, 4, 4, 4}
```

Question 6 : Est-il possible d'accéder aux éléments d'un ensemble avec l'opérateur [].

- Non car set objects are unordered and not subscriptable

Question 7 : Opérations sur les ensembles : union, intersection, différence

```
e1 = {1, 2}
e2 = {2, 3}
e3 = e1.union(e2)
print(e3)
e4= e1 | e2
print(e4)
e5 = e1.intersection(e2)
print(e5)
e6 = e1 & e2
print(e6)
e7 = e1 - e2
```

```
print(e7)
print(e8)
e8 = e1.symmetric_difference(e2)
```

Question 8: Qu'en dites-vous des possibilités de conversion entre string, tuple, set, list et dict. Illustrer avec des exemples.

Partie 5 : Python : indexing and slicing

- Le terme anglais de slice est associé à l'idée de découpage.
- Un slice permet le découpage de structures de données séquentielles.
- a[start:stop] # items start through stop-1
- a[start:]# items start through the rest of the array
- a[:stop] # items from the beginning through stop-1
- a[:] # a copy of the whole array
- a[start:stop:step] # start through not past stop, by step
- Fonction slice :
    - Syntax: slice(start, stop, step)
    - start: Starting index where the slicing of object starts.
    - stop: Ending index where the slicing of object stops.
    - step: It is an optional argument that determines the increment between each index for slicing.
    - Return Type: Returns a sliced object containing elements in the given range only.

Le tableau ci-dessus résume ce que l'on utilise des slices dans 80% de la pratique.

Action	Code	Exécution (s = "ABCDEFGHIJKL")
Extraction	s[2:7]	CDEFG
Les 4 premiers	s[:4]	ABCD
Les 4 derniers	s[-4:]	IJKL
Tous sauf les 4 premiers	s[4:]	EFGHIJKL
Tous sauf les 4 derniers	s[:-4]	ABCDEFGH
Partitionner	s[:3], s[3:7], s[7:]	('ABC', 'DEFG', 'HIJKL')
De 3 en 3	s[::3]	ADGJ
De 3 en 3 à partir de la fin	s[::-3]	LIFC
Les indices pairs	s[::2]	ACEGIK
Les indices impairs	s[1::2]	BDFHJL
Copie superficielle	s[::]	ABCDEFGHIJKL
Copie à l'envers	s[::-1]	LKJIHGFEDCBA

Partie 6: La fonction Lambda

- Le mot-clé lambda est utilisé pour déclarer une fonction anonyme, raison pour laquelle ces fonctions sont appelées « fonctions lambda ».
    - Une fonction anonyme se réfère à une fonction déclarée sans nom.
- Les fonctions lambda se comportent de la même manière que les fonctions régulières qui sont déclarées en utilisant le mot-clé def.
- Les fonctions Lambda incluent toujours une instruction return implicite (la fonction lambda renvoie automatiquement le résultat de l'expression dans la fonction une fois exécutée.)
- Syntaxe d'une fonction lambda en python
  lambda arguments : expression
- La fonction lambda peut comporter n'importe quel nombre d'argument mais une seule expression.

Fonction classique	Fonction Lambda

X = 5
 	
Y = 10
 	
def sum_classic(a , b):
 	
  return a + b
 	
  print(sum_classic(x,y))
	a = 5
	b = 10
	sum_lambda = lambda a, b : a + b

- Les fonctions anonymes sont utilisées :
  - lorsqu'il n'est pas nécessaire de nommer une fonction. Il y a aucun intérêt de nommer une fonction qu'on utilisera qu'une fois.
  - lorsqu'on a besoin de lisibilité dans notre code.
  - lorsqu'il faut effectuer des petites tâches avec moins de code (big data, data analysis)

Question 1 : Exécuter les codes dans les exemples suivants et interpréter les résultats.

Exemple 1 :

```
r = lambda a : a + 15
print(r(10))
r = lambda x, y : x * y
print(r(12, 4))
```

Exemple 2 :

```
subject_marks = [('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]
print("Original list of tuples:")
print(subject_marks)
subject_marks.sort(key = lambda x: x[1])
print("\nSorting the List of Tuples:")
print(subject_marks)
```

Exemple 3 :

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Original list of integers:")
print(nums)
print("\nEven numbers from the said list:")
even_nums = list(filter(lambda x: x%2 == 0, nums))
print(even_nums)
print("\nOdd numbers from the said list:")
odd_nums = list(filter(lambda x: x%2 != 0, nums))
print(odd_nums)
```

Exemple 4:

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Original list of integers:")
print(nums)
print("\nSquare every number of the said list:")
square_nums = list(map(lambda x: x ** 2, nums))
print(square_nums)
print("\nCube every number of the said list:")
cube_nums = list(map(lambda x: x ** 3, nums))
print(cube_nums)
```

Exemple 5:

```
import datetime
```

```
now = datetime.datetime.now()
print(now)
year = lambda x: x.year
month = lambda x: x.month
day = lambda x: x.day
t = lambda x: x.time()
print(year(now))
print(month(now))
print(day(now))
print(t(now))
```

Exemple 6:

```
from functools import reduce
fib_series = lambda n: reduce(lambda x, _: x+[x[-1]+x[-2]],                    range(n-2), [0, 1])
print("Fibonacci series upto 2:")
print(fib_series(2))
print("\nFibonacci series upto 5:")
print(fib_series(5))
print("\nFibonacci series upto 6:")
print(fib_series(6))
print("\nFibonacci series upto 9:")
print(fib_series(9))
```

Exemple 7 :

```
array_nums = [-1, 2, -3, 5, 7, 8, 9, -10]
print("Original arrays:")
print(array_nums)
result = sorted(array_nums, key = lambda i: 0 if i == 0 else -1 / i)
print("\nRearrange positive and negative numbers of the said array:")
print(result)
```

Partie 7 : Les fonctions prédéfinies les plus utilisées

Question 1: Exécuter les fonctions les plus utilisées sur les structures de données set,tuple, list et dict.

Les tuples		
S.no	Method	Description
1	Cmp ()	compare les éléments de deux tuples. Syntaxe : cmp(tuple1, tuple2)
2	Len ()	retourne le nombre d'éléments dans le tuple. Syntaxe : len(tuple)
3	Max ()	retourne la valeur maximale des éléments d'un tuple. Syntaxe : max (tuple)
4	Min ()	retourne la valeur minimale des éléments d'un tuple. Syntaxe : min(tuple)
5	Tuple ()	La méthode tuple () permet de convertir une liste d'éléments en tuple. Syntaxe : tuple(nom_liste)
6	Sum ()	La méthode Sum () permet de calculer la somme des éléments d'un tuple. Syntaxe Sum(tuple, valeur_de_début) NB : valeur_de_début (optionnelle) : Cette valeur est ajoutée à la somme des éléments du tuple. La valeur de départ par défaut est 0.
7	Del	Permet de supprimer la liste entière non modifiable. Syntaxe : del nom_tuple
up8	Count ()	interroger la fréquence à laquelle un élément se présente dans le tuple Python. Syntaxe : nom_tuple.count("element_existant_dans_tuple ")
9	Index ()	interroger l'index d'une valeur spécifique avec index. Syntaxe : nom_tuple.index("element_existant_dans_tuple ")

List in python		
S.no	**Method**	**Description**
1	append()	Used for appending and adding elements to the end of the List.
2	copy()	It returns a shallow copy of a list
3	clear()	This method is used for removing all items from the list.
4	count()	These methods count the elements
5	extend()	Adds each element of the iterable to the end of the List
6	index()	Returns the lowest index where the element appears.
7	insert()	Inserts a given element at a given index in a list.
8	pop()	Removes and returns the last value from the List or the given index value.
9	remove()	Removes a given object from the List.
10	reverse()	Reverses objects of the List in place (or list = list[::-1])
11	sort()	Sort a List in ascending, descending, or user-defined order
12	min()	Calculates the minimum of all the elements of the List
13	max()	Calculates the maximum of all the elements of the List
14	zip()	Combines lists and results a list of tuples.
15	deepcopy()	It returns a deep copy of the list.

Les dictionnaires	
Méthode	**Description**
clear	Supprime toutes les données du dictionnaire.
copy	Duplique (shallow copy) un dictionnaire.
fromkeys	Permet de créer un nouveau dictionnaire à partir d'un jeu de clés.
get	Permet de retrouver une donnée du dictionnaire à partir de sa clé.
items	Renvoie l'ensemble des paires clé/valeur, sous forme d'une collection de tuples.
keys	Renvoie l'ensemble de clés du dictionnaire.
pop	Permet de supprimer et de retourner une donnée du dictionnaire à partir de sa clé.
popitem	Supprime une association du dictionnaire (en mode LIFO) et vous la renvoie sous forme d'un tuple.
setdefault	Insert une nouvelle entrée dans le dictionnaire, si la clé n'existe, et renvoie la valeur associée à la clé.
update	Fusionne dans le dictionnaire courant les entrées présentes dans un autre dictionnaire.
values	Renvoie l'ensemble de valeurs du dictionnaire.
Len	calcule la taille d'un dictionnaire. Retourne le nombre d'associations clé/valeur.
sorted	renvoie une liste triée des clés ou des valeurs constitutives d’un dictionnaire.

- # List slicing
- L = [1, 2, 3, 4, 5]
- s1 = slice(3)
- s2 = slice(1, 5, 2)
- print("List slicing")
- print(L[s1])
- print(L[s2])

Pour les listes :

En python del est mot-clé et remove(), pop() sont des méthodes intégrées. Le but de ces trois méthodes est le même mais le comportement est différent . La méthode remove() supprime les valeurs ou l'objet de la liste à l'aide de la valeur et del et pop() supprime les valeurs ou l'objet de la liste à l'aide d'un index.

Del	**Remove**	**Pop**
del est un mot clé.	C'est une méthode.	pop() est une méthode.
Pour supprimer la valeur, il utilise l'index.	Pour supprimer une valeur, cette méthode utilise la valeur comme paramètre.	Cette méthode utilise également l'index comme paramètre à supprimer.
Le mot-clé del ne renvoie aucune valeur.	La méthode remove() ne renvoie aucune valeur.	pop() renvoie la valeur supprimée.
Le mot-clé del peut supprimer la valeur unique d'une liste ou supprimer la liste entière à la fois.	À la fois, il supprime une seule valeur de la liste.	À la fois, il supprime une seule valeur de la liste.
Il génère une erreur d'index si l'index n'existe pas dans la liste.	Il génère une erreur de valeur si la valeur n'existe pas dans la liste.	Il génère une erreur d'index dans le cas où un index n'existe pas dans la liste.

2. Les arrays (vecteurs multidimentionnels)
3. Les series
4. Les dataframes
5. Compréhension in python
6. Les fonctions Lambda
7. La programmation orienté objet
8. Les piles avec listes
9. Les files avec listes
10. Linked list in python

Comparaison des plusieurs structures de don

11. nées

Partie X :

Université de Monastir
Institut Supérieur d'Informatique et de Mathématiques de Monastir
Département Sciences de l'Informatique
Ing1 – 2022/2023
Enseignante du cours : Mariem Gzara
Enseignante de TP : Azza Chebil

➔ https://www.techiedelight.com/fr/check-key-exists-dictionary-python/#:~:text=Le%20moyen%20pr%C3%A9f%C3%A9r%C3%A9%20et%20le,le%20dictionnaire%2C%20False%20Par%20ailleurs.&text=La%20in%20l%27op%C3%A9rateur%20impl%C3%A9mente,appeler%20__contains__%20directement.

Performance des structures de données Python

Objectif :

L'objectif du TP est de comparer la performance de quelques opérations sur les structures de données list, set, dict et array. Le choix des structures de données python (set, list, array, dict, series) adaptées pour implémenter d'autres structures de données plus complexes comme les graphes, les files de priorités et les tables de hachage dépendra du contexte d'utilisation et de la performance de ces structures.

Partie I : Travail d'implémentation :

Question 1 : Performance des opérations sur la structure de données List.

Opération	Performance	Production de graphiques
Copy	Ecrire une fonction Perf_copy qui : 1- Crée une liste L de taille n entiers tous égaux à 1. 2- Copie la liste L dans une liste LL. 3- Répète k fois l'étape 2. 4- Retourne le temps d'exécution.	Faites varier n puis tracer le courbe temps d'exécution en fonction de n.
Append	Ecrire une fonction Perf_append qui : 1- Crée une liste L vide. 2- Ajoute l'entier 1 à la liste L. 3- Répète n fois l'étape 2. 4- Retourne le temps d'exécution.	Faites varier n puis tracer le courbe temps d'exécution en fonction de n.
Pop last	Ecrire une fonction Perf_popLast qui : 1- Crée k listes de n entiers tous égaux à 1. 2- Supprime le dernier élément de chaque liste. 3- Retourne le temps d'exécution de k fois l'instruction 2.	Faites varier k puis tracer le courbe temps d'exécution en fonction de n.
Pop intermediate	Ecrire une fonction Perf_popInter qui : 1- Crée k listes de n entiers tous égaux à 1. 2- Supprime le pème élément de chaque liste. 3- Retourne le temps d'exécution de k fois l'instruction 2.	Faites varier k puis tracer le courbe temps d'exécution en fonction de n.

Question 2 : Performance des opérations sur la structure de données Set.

Opération	Average case	Production des graphiques
Union s|t	Ecrire une fonction Perf_Union qui : 1- Crée 2 sets A et B. A contient les entiers de 1 à n et B contient les entiers de n+1 à m. 2- Construire le set C qui est l'union de A et B. 3- Répéter k fois l'étape 2. 4- Retourne le temps d'exécution de k fois l'instruction 2.	Faites varier n puis tracer le courbe temps d'exécution en fonction de n.
Intersection s&t	Ecrire une fonction Perf_Intersect qui : 1- Crée 2 sets A et B. A contient les entiers de 1 à n et B contient les entiers de n/2 à n. 2- Construire le set C qui est l'intersection de A et B. 3- Répéter k fois l'étape 2. Retourne le temps d'exécution de k fois	Faites varier n puis tracer le courbe temps d'exécution en fonction de n.

	l'instruction 2.	

Question 3 : Performance des opérations sur la structure de données dict.

Opération	Performance	Production de graphiques
Copy	Ecrire une fonction Perf_dictcopy qui : 1- Crée un dict L de taille n entiers tous égaux à 1 comme valeurs et les clés sont les entiers de 1 à n. 2- Copie le dict L dans un dict LL. 3- Répète k fois l'étape 2. 4- Retourne le temps d'exécution.	Faites varier n puis tracer le courbe temps d'exécution en fonction de n.
Set Item	Ecrire une fonction Perf_hash qui : 5- Crée un dict D vide. 6- Insère une clé et val=1 7- Répète n fois l'étape 2. 8- Retourne le temps d'exécution.	Faites varier n puis tracer le courbe temps d'exécution en fonction de n.
Delete Item	Ecrire une fonction qui : 1- Crée k dict de n entiers tous égaux à 1. 2- Supprime le dernier élément de chaque liste. 3- Retourne le temps d'exécution de k fois l'instruction 2.	Faites varier k puis tracer le courbe temps d'exécution en fonction de n.
Pop intermediate	Ecrire une fonction Perf_popInter qui 1- Crée k listes de n entiers tous égaux à 1. 2- Supprime le pème élément de chaque liste. 3- Retourne le temps d'exécution de k fois l'instruction 2.	Faites varier k puis tracer le courbe temps d'exécution en fonction de n.

Partie II : Compte rendu

Question 6 : Commenter les graphiques obtenus dans les questions précédentes.

Question 7 : Dresser un tableau qui répertorie les principales opérations sur les structures de données list, set, dict, array, tuple et donner la complexité temporelle de ces opérations. (citer vos sources d'information).

Notes

1- List

The Average Case assumes parameters generated uniformly at random.

Internally, a list is represented as an array; the largest costs come from growing beyond the current allocation size (because everything must move), or from inserting or deleting somewhere near the beginning (because everything after that must move). If you need to add/remove at both ends, consider using a collections.deque instead.

Opération	Average Case	Amortized Worst Case
Copy	O(n)	O(n)
Append[1]	O(1)	O(1)
Pop last	O(1)	O(1)
Pop intermediate[2]	O(n)	O(n)
Insert	O(n)	O(n)
Get Item	O(1)	O(1)
Set Item	O(1)	O(1)
Delete Item	O(n)	O(n)
Iteration	O(n)	O(n)
Get Slice	O(k)	O(k)
Del Slice	O(n)	O(n)
Set Slice	O(k+n)	O(k+n)
Extend[1]	O(k)	O(k)
Sort	O(n log n)	O(n log n)
Multiply	O(nk)	O(nk)
x in s	O(n)	
min(s), max(s)	O(n)	
Get Length	O(1)	O(1)

2- Collections (deque)

A deque (double-ended queue) is represented internally as a doubly linked list. (Well, a list of arrays rather than objects, for greater efficiency.) Both ends are accessible, but even looking at the middle is slow, and adding to or removing from the middle is slower still.

Operation	Average Case	Amortized Worst Case
Copy	O(n)	O(n)
append	O(1)	O(1)
appendleft	O(1)	O(1)
pop	O(1)	O(1)
popleft	O(1)	O(1)
extend	O(k)	O(k)
extendleft	O(k)	O(k)
rotate	O(k)	O(k)
remove	O(n)	O(n)

3- Set

See dict -- the implementation is intentionally very similar.

Operation	Average case	Worst Case	notes
x in s	O(1)	O(n)	
Union s|t	O(len(s)+len(t))		
Intersection s&t	O(min(len(s), len(t)))	O(len(s) * len(t))	replace "min" with "max" if t is not a set
Multiple intersection s1&s2&..&sn		(n-1)*O(l) where l is max(len(s1),..,len(sn))	
Difference s-t	O(len(s))		
s.difference_update(t)	O(len(t))		
Symmetric Difference s^t	O(len(s))	O(len(s) * len(t))	
s.symmetric_difference_update(t)	O(len(t))	O(len(t) * len(s))	

- As seen in the source code the complexities for set difference s-t or s.difference(t) (set_difference()) and in-place set difference s.difference_update(t) (set_difference_update_internal()) are different! The first one is O(len(s)) (for every element in s add it to the new set, if not in t). The second one is O(len(t)) (for every element in t remove it from s). So care must be taken as to which is preferred, depending on which one is the longest set and whether a new set is needed.
- To perform set operations like s-t, both s and t need to be sets. However you can do the method equivalents even if t is any iterable, for example s.difference(l), where l is a list.

4- Dict

The Average Case times listed for dict objects assume that the hash function for the objects is sufficiently robust to make collisions uncommon. The Average Case assumes the keys used in parameters are selected uniformly at random from the set of all keys.

Note that there is a fast-path for dicts that (in practice) only deal with str keys; this doesn't affect the algorithmic complexity, but it can significantly affect the constant factors: how quickly a typical program finishes.

Operation	Average Case	Amortized Worst Case
k in d	O(1)	O(n)
Copy[3]	O(n)	O(n)
Get Item	O(1)	O(n)
Set Item[1]	O(1)	O(n)
Delete Item	O(1)	O(n)
Iteration[3]	O(n)	02.3 O(n)