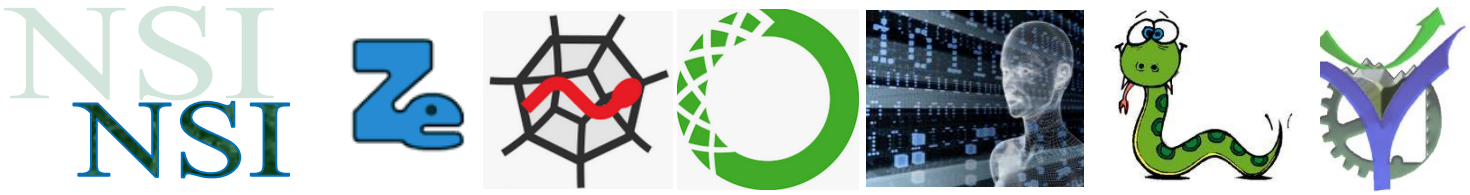
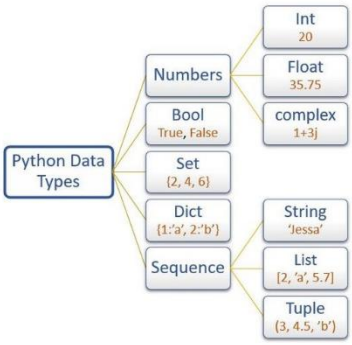


La complexité des types de bases en Python un aspect pratique



Résumé :

Nous utilisons dans nos scripts les types de données mis à disposition par Python. Ce petit travail va nous permettre de découvrir que chacun de ces types possède son propre comportement en termes de rapidité.

Aussi en fonction ce que nous voulons faire il est important de connaître le comportement de ces différents types pour les utiliser à bon escient.

Les réponses aux questions posées seront données dans le document réponse joint.

Sommaire :

1 Le challenge

On souhaite réaliser une fonction qui retourne une table de valeurs entières, une *list* python, dans le but de tester des algorithmes de tris. La contrainte consiste à n'avoir aucunes valeurs identiques dans cette table.

Cette table est correctement formée :

1	8	12	23	47	7	9	15	16	3

Cette table ne l'est pas :

1	8	12	23	16	7	9	15	16	3

1.1 Une première analyse

Q1. Proposez un algorithme pour réaliser cette fonction en pseudo langage.

1.2 Tirage aléatoire d'une valeur entière en Python

Pour utiliser des nombres aléatoires nous utiliserons la bibliothèque *random* et plus particulièrement la méthode *randrange*.

Q2. Expliquez le fonctionnement de la ligne de code ci-dessous :

```
nouvelle_valeur = random.randrange(0,valeur_max)
```

2 La complexité du fonctionnement de quelques types construits de Python[1]

Les tableaux ci-dessous résument la complexité du comportement des types *List*, *Dict* et *Set* de Python pour différentes opérations.

Q3. Rappeler brièvement ce que signifie la complexité d'un algorithme.

Q4. Quelle est son utilité ?

Q5. Que signifie une complexité en $O(1)$?

Q6. Que signifie une complexité en $O(n)$?

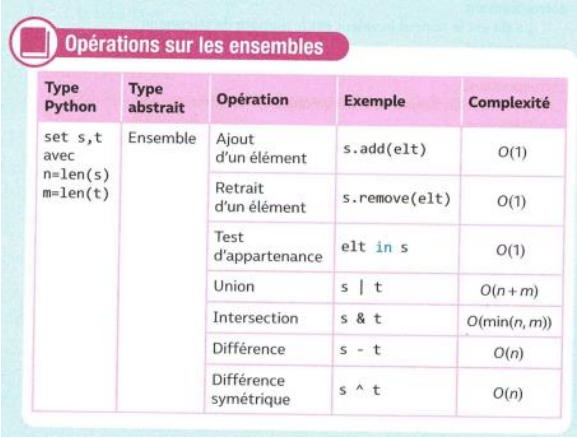
Opérations sur les listes/piles

Type Python	Type abstrait	Opération	Exemple	Complexité
list lst avec n=len(lst)	Tableau dynamique	Ajout à la fin	`lst.append(x)`	$O(1)$
		Accès à un élément	`lst[i]`	$O(1)$
		Modification d'un élément	`lst[i] = x`	$O(1)$
		Effacement d'un élément	`del lst[i]`	$O(n)$
		Insertion d'un élément	`lst.insert(i,x)`	$O(n)$
		Recherche d'un élément	`x in lst`	$O(n)$
	Pile	*push*	`lst.append(x)`	$O(1)$
		pop	`lst.pop()`	$O(1)$

Opérations sur les dictionnaires

Type Python	Type abstrait	Opération	Exemple	Complexité
dict d avec n=len(d)	Tableau associatif	Ajout d'un élément	`d[key] = val`	$O(1)$
		Modification d'un élément	`d[key] = val`	$O(1)$
		Effacement d'un élément	`del d[key]`	$O(1)$
		Accès à un élément	`d[key]`	$O(1)$
		Recherche d'une clé	`key in d`	$O(1)$
		Recherche d'une valeur	`val in d.values()`	$O(n)$

[1] Les tableaux sont issus de Prépabac NSI Hatier.



Opérations sur les ensembles

Type Python	Type abstrait	Opération	Exemple	Complexité
`set s,t` avec `n=len(s)` `m=len(t)`	Ensemble	Ajout d'un élément	`s.add(elt)`	$O(1)$
		Retrait d'un élément	`s.remove(elt)`	$O(1)$
		Test d'appartenance	`elt in s`	$O(1)$
		Union	`s | t`	$O(n+m)$
		Intersection	`s & t`	$O(\min(n,m))$
		Différence	`s - t`	$O(n)$
		Différence symétrique	`s ^ t`	$O(n)$

3 Création des tables étude comparative de deux fonctions

3.1 L'algorithme utilisé

Vous aller utiliser le script Python ci-dessous qui créé les tables de deux manières différentes. Voilà un exemple de résultats pour deux tables de dimension 20 :

tout_est_dans_les_details.py

```
-----------------
CREATION DE TABLE
-----------------

Donnez la dimension de la table :

Création d'une table méthode basique de      20 éléments

La durée de la création de la table est de           20 µs

[54, 112, 151, 186, 73, 156, 5, 122, 67, 58, 22, 97, 116, 52, 160, 173, 164, 172, 118, 132]

Création d'une table méthode évoluée de      20 éléments

La durée de la création de la table est de           12 µs

[115, 154, 67, 132, 184, 40, 72, 64, 128, 89, 167, 177, 98, 176, 125, 87, 137, 76, 126, 194]
```

3.2 Fonctionnement comparatif

Script1. Faites fonctionner le script et relever les résultats dans le tableau de la feuille réponse.

Q7. Déterminez la complexité de la solution basique et la complexité de la solution avancée à partir des résultats obtenus.

Q8. En comparant le code des deux fonctions proposer une explication de la différence de comportement.