Python : Attendus au programme

Traits généraux

Le typage dynamique

Au contraire des langages comme le C/C++, qui eux sont à typage statique, le Python associe aux objets un type à la volée, lors de l'exécution du code. C'est ce qu'on appelle le **typage dynamique**. Affecter un type aux différents objets permet de leur allouer respectivement un taille adéquate dans la mémoire.

Rappel: Nommer des objets

Un notation d'affectation peut commencer par:

- *Une lettre (min,MAJ)*
- *Un underscore (_)*

et peut contenir des entiers.
⚠ *Elle ne peut pas commencer par un entier. Ex: "5Id_Min" n'est pas valide.*

Lors de l'exécution du code, l'interpréteur:

- Caractérise l'objet et y accorde une place en mémoire dans l'espace des objets
- Ajoute la dénomination de celui-ci dans l'espace des variable
- Crée la référence entre les deux

Syntaxe

→ ***del (Variable)*** *supprime la variable dans la mémoire.*

→ ***type(Objet/Variable)*** *retourne le type du paramètre donné.*

En Python, il n'est pas nécessaire de munir les arguments d'une fonction de leur type; toutefois par soucis de lisibilité et pour déboguer, cela peut s'avérer pratique.

```
1 def f( x : int) -> int :
2 # Hyp: Aucune ici
3 # Retourne x²
4     return x**2
```

Portée lexicale

Lorsqu'une expression fait référence à une variable à l'intérieur d'une fonction, Python cherche la valeur définie à l'intérieur de la fonction et à défaut la valeur dans l'espace global du module.

Appel de fonction par valeur

L'exécution de f(x) évalue d'abord x puis exécute f avec la valeur calculée.

Définition: Passage par valeur

> *Le passage d'un paramètre par valeur copie la valeur du paramètre dans une variable locale à la fonction.*

Définition: Passage par référence

> *Une variables est passée par référence lorsque l'on donne l'adresse de celle-ci en argument de la fonction. Dès lors, on agit concrètement sur la variable, même à l'intérieur de la fonction.*

⚠ A la différence du C où l'on peut utiliser des pointeurs afin de passer les objets par référence ; en python, il faut distinguer ceux qui sont de type **mutable** ou non.

Définition: Variable non mutable

> *Une variable de type non mutable ne peut être modifiée. Une opération sur une variable de ce type entraîne nécessairement la création d'une autre variable du même type, même si cette dernière est temporaire.*

```
x=2          # On déclare une variable de type int
f(2)=4       # On opère à partir de celle-ci
print(x)     # On regarde si x a été modifiée
>> 2
```

Ainsi on donne une qualification des différentes types ci-dessous :

Mutables	Non mutable
Ensemble, Liste, Dictionnaire	Tuple, Int, Float, Bool, Frozenset

Types de base

Type int / float

Affecter un entier à une variable initialise celle-ci avec un type **int**. Lui affecter un réel au format "x.y" lui donne un type **float**. On donne les opérations suivantes sur les entiers :

+ -	Addition, soustraction
+= -=	Addition, soustraction et affectation
* /	Multiplication, division
*= /=	Multiplication, division et affectation
//	Division entière
%	Modulo

Syntaxe

float(objet) *Converti l'objet en réel.*
int(objet) *Converti l'objet en entier.*
⚠ *Parfois il faut avoir recours à des transtypages en amont ; comme dans le cas str -> float -> int où la la chaine de caractère vaut par exemple "3.5".*

Type bool

Syntaxe

True, False
not, and , or
==, !=, <, >, <=, >=.

Evaluation paresseuse: sur un test if (cond1) and (cond2):, on détermine d'abord le résultat du premier test. S'il est faux, le second n'est pas évalué. C'est notamment utile lorsqu'on doit vérifier que l'indice dans un tableau est acceptable. Si L est un tableau de taille n, pour s'assurer qu'on n'aura pas d'erreur Index out of range, on peut utiliser : if (i<n) and (L[i]...) .

Type structuré

Structures indicées immuables

	Chaîne	N_uplet
Type	str	tuple
Accès	s[i]	t[i]

Définition: Séquence

Un objet séquentiel ou séquence est une collection itérable, ordonné et indexable. Les objets séquentiels sont les listes, les chaînes de caractères, les objets de type range, ainsi que les tuples (cf. plus bas).
Ie. on peut faire une boucle dessus, chaque élément est indicé, et la création de la séquence détermine l'ordre des objets dans la collection

Listes

Initialisation

Syntaxe

*[**e for x in s**], où e est une expression, x une variable à valeur dans s*
*[x_0] *** n**, où x_0 est la valeur d'initialisation d'une liste de taille n*
liste.append()
***list(range(début, fin, pas))** où range génère des entiers de début à fin selon le pas donné*

Méthodes

Indiçage

```
                        ["A", "B", "C", "D", "E", "F"]
    #indice positif :    0    1    2    3    4    5
    #indice négatif :   -6   -5   -4   -3   -2   -1

```

Opérations

Syntaxe

sum(), min(), max(), del(liste[i]), list.insert(i, x)
***list.pop([i])** : Enlève de la liste l'élément situé à la position indiquée et le renvoie en valeur de retour. Si aucune position n'est spécifiée, **list.pop()** enlève et renvoie le dernier élément de la liste*
***list.sort()** ordonne les éléments, **list.sorted()** le fait en renvoyant une nouvelle liste et en ne modifiant par la première*
***list.reverse** inverse l'ordre de la liste*

Usage

On peut implémenter les piles à l'aide de liste, en utilisant les méthodes: **list.append() / list.pop()**
On peut implémenter les files en utlisant le module `from collections import deque`.

Syntaxe

***list.popleft()** pour défiler et **list.append()** pour enfiler*

Chaînes,Tuples & Listes : Opérations communes

Syntaxe

***len(objet)** retourne la taille de l'objet*
+ opérateur de concaténation
***** opérateur d'itération*
***obj[n,m]** extraction par tranche. On utilise **:** pour "aller jusqu'au bout"*
***enumerate(objet)** permet d'itérer sur (position, élément) (Hors programme)*
***zip**($obj_1, obj_2, ...$) permet d'itérer sur plusieurs séquences en même temps*

Exercice

Créer deux listes, nom et âge, pour 4 individus et afficher "X à Y ans" à l'aide d'une seule boucle.

Type ensemble

```python
#Créer un ensemble vide
E=set()

# On crée un ensemble à partir d'une liste
liste = [1, 2, 3, 4, 2, 5]
nb_set = set(liste)

# Ajouter un élément
E.add(5)
E.update(liste) # On peut remplacer liste par tout objet convertible en ensemble

# On peut créer un ensemble par compréhension
a = {x for x in 'abracadabra' if x not in 'abc'}
# Sortie : a={'r', 'd'}

```

On peut convertir en ensemble : un tuple, une chaine de caractère etc... ou-bien un dictionnaire, un ensemble ainsi qu'un itérrable.

Syntaxe

> ***&*** *fait l'intersection,* ***|*** *fait l'union,* ***-*** *fait la différence,* ***^*** *fait la différence symétrique*
> ***set.remove(elem)*** *retire eleme de set, erreur sinon*
> ***set.clear()*** *vide set*
> ***set.copy()***

Rappel:

> *{} ne crée pas un ensemble vide, mais un dictionnaire !*

0.1 Exercice

Créer $\{n \in [0..20] \cap 2\mathbb{N}\}$ d'au moins trois manière différentes ; en au plus trois lignes de code.

0.2 Exercice

Créer l'ensemble des entiers jusqu'à 20 ; puis, y retrancher les entiers impairs.
Procéder à un affichage.

Dictionnaire

Lorsque les listes ne se révèlent pas suffisamment efficientes pour traiter un ensemble de données, on peut avoir recourt aux dictionnaires.

Définition: Dictionnaire

> *Ensemble de données au format* ***Clé -> Valeur*** *où à la clé est un identifiant unique repérant la valeur qu'il indexe. Subséquemment, les clés sont des objets non mutables, comme les entiers, les chaînes de caractères ou encore les n_uplets d'objets eux-mêmes non mutables.*

Exemple

```
animal = {"nom": "singe", "poids": 70, "taille": 1.75}
```

Opérations & Méthodes

- Accéder à un élément à partir de sa clé: `dico[clé]`
- Ajouter un élément dans le dictionnaire: `dico[new_clé]=élément`
- Supprimer un élément du dictionnaire: `del dico[clé]`
- Tester la présence d'une clé: `clé in dico #Renvoie True/False`
- Obtenir la liste des clés: `dico.keys()`
- Obtenir la liste des valeurs: `dico.values()`
- Obtenir la liste des clés,valeurs: `dico.items()`
- Obtenir la valeur indexée par une clé: `dico.get(clé)`

Manipulation des méthodes

Les méthodes **.keys() .values() .items()** sont intérables, mais pas indexables.
Ainsi, l'instruction `dico.keys()[i]` produit une erreur.

- Trier par clés: `sorted(dico)` Retourne la liste triée des clés.
- Trier par valeurs: `sorted(dico,key=dico.get)` Retourne la liste des clés triées relativement à l'ordre sur les valeurs, **si tant est que cela soit possible** ⚠
- Obtenir la clé de la valeur max `max(dico, key=dico.get)`, respectivement min.
- Convertir un objet séquentiel contenant d'autres objets séquentiels de 2 éléments : `dict(objet)`

Complexité: comparatif Liste/Dictionnaire

- Le test d'appartenance est en O(1) pour un dictionnaire, alors qu'il peut être en O(n) pour une liste.
- Les temps d'accès à un élément donné est en O(1) pour les deux.
- Le temps de suppression d'un élément est en O(1) pour un dictionnaire et en O(n) pour une liste (sauf si c'est le dernier élément qu'on retire avec pop())

Exercices

1 Les bases

1.1 Opérations simples sur les listes

Étant donné une liste une liste d'entiers ; implémenter les fonctions suivantes :

- appartient_L(l : list, x : int) -> bool
- position(l : list, x : int) -> int
- nb_occurences(l : list, x : int) -> int
- max_L(l : list) -> int

1.2 Traitement du langage

Étant donné deux mots, u et v tels que $|u| \leq |v|$, implémenter la fonction nb_occ_motif qui retourne le nombre d'occurence de u dans v.

2 kNN

Implémenter l'algorithme des k plus proches voisins à partir du fichier .py donné en suivant les étapes ci-dessous:

- Initialiser une liste de N abscisses (X) aléatoires dans [0..60]. De même pour les ordonnées (Y), aléatoirement générées dans [0..60].
- Initialiser S avec un couple de coordonnées aléatoirement tirées dans [0..60]x[0..60].
- Procéder au partitionnement spatial des coordonnées en initialisant Tc, le tableau des coordonnées (abs,ord), par la même occasion. Cette étape donne lieu au remplissage de Classe.
- Implémenter la fonction `dist(a:tuple, b:tuple)` qui retourne la distance euclidienne.
- Implémenter la fonction `k_NN( k : int , T : list, s : tuple, n : int) -> int`
  On implémentera l'ensemble des k sommets les plus proches de S par un dictionnaire, où les clés seront les sommets indexé par [1..N] et les valeurs la distance `dist(S,T[i])` . Ceci permettra d'utiliser les fonctions vues précédemment, notamment pour la recherche de l'argmax.