Romain LUCAS 27/10/2022

PARTIEL DE PAPY

Partie 1

1.1 Connaissances de Python

1. Un objet est dit mutable si on peut en changer ses valeurs sans changer l'adresse méoire. On a donc des effets de bord. Des exemples d'objets mutables sont la liste et le dictionnaire.

Au contraire un objet immuable est affecté à une valeur et changer cette valeur revient à changer l'adresse mémoire. Des exemples sont les integers ou encore les tuples.

2.

Changement de la deuxième valeur

Pour un tuple, c'est un objet immuable donc on est obligé de réaffecter l'ensemble de l'objet :

```
v = (v[0], new_val, v[2])
```

On récupère les anciennes premières et troisièmes valeurs du tuple pour les placer dans un nouvel objet.

Pour une liste, c'est un objet mutable donc on peut simplement changer la valeur du deuxième élément de la liste :

```
v[1] = new_val
```

Cela modifiera la liste v toute entière.

Allongement du vecteur

Pour un tuple, il faut lui affecter un nouveau tuple de longueur 4 :

```
v = (v[0], v[1], v[2), new_val)
```

Pour une liste on peut utiliser la méthode append :

```
v.append(new_val)
```

3. Un dictionnaire représente un ensemble de couple (clé, valeur) où chaque clé est unique et renvoie à la valeur associée.

Un dictionnaire est créé en utilisant la syntaxe

```
d = {clé1 : val1, clé2 : val2)
```

Si on veut récupérer la valeur associée à la clé 1

```
val = d[clé1]
```

4. Un docstring est un string que l'on place en début de fonction, de méthode, de classe ou de module afin d'en créer une documentation. Usuellement un docstring indique ce qué fait l'objet qui lui est associé, par exemple pour une fonction en détaillant paramètres et résultats.

Ce docstring peut être affiché en utilisant la fonction help(fonction_qui_contient_le_docstring) ou en utilisant l'attribut __doc__.

Un docstring est contenu entre deux """.

5.

Après la première étape a vaut 3

Après la deuxième étape a vaut 3, b vaut 2.

A la troisième étape, c prend la valeur actuelle de a, c'est à dire 3. Cependant les int étant des objets immuables, c est bien associé à sa propre case mémoire valant 3, ainsi si on modifie la valeur de a, c étant dans une autre case mémoire ne sera pas affecté

Après la troisième étape, a vaut 3, b vaut 2 et c vaut 3.

Après la quatrième étape, a vaut 4, b vaut 2 et c n'ayant pas été affecté par le changement impactant a, vaut toujours 3.

Finalement on print la valeur de c, c'est à dire **3**.

6.

1.

    L renvoie vers la case mémoire contenant la liste [1,2,3].

    L2 = L fait que L2 renvoie aussi vers la case mémoire contenant la liste [1, 2, 3].

    Lorsque l'on fait L2 = 'a' on fait que L2 renvoie vers la case mémoire contenant 'a'. Mais on ne touche pas à la case mémoire contenant [1,2,3]. L renvoyant toujours vers cette case, elle n'est pas affectée.

    Donc après ces opérations, L vaut toujours [1,2,3]

2.

    Cette fois l'opération L2[1] = 'a' affecte directement la case mémoire qui ne contient plus [1,2,3] mais [1, 'a', 3]. L revoie vers cette case mémoire, sa nouvelle valeur, tout comme celle de L2 est donc [1, 'a', 3]

7. Un module est un programme python qui a déjà été écrit et que l'on peut importer dans un programme afin d'y ajouter les fonctionnalités disponibles dans le dit module. Ce module peut être créé directement par le programmeur ou être créé par la communauté et réutiliser dans le programme.

8. On peut faire de différentes manières :

```
|	import numpy as np
```

Ici on a importé l'ensemble du module numpy sous un alias np, on peut alors utiliser la fonction linspace :

```
|	np.linspace(0, 10, 0.01)
```

Une autre manière de faire important seulement linspace est :

```
|	from numpy import linspace
```

Ici on a seulement importé linspace, on peut directement l'utiliser dans le programme comme telle :

```
|	linspace(0, 10, 0.01)
```

1.2 Être un bon développeur

1

L'utilisation d'environnements en python permet de créer un python isolé adapté à un ou plusieurs projets, avec ses propres modules installés sans qu'ils soient impactés ni ne puissent être impactés par la version de python que l'on utilise de base.

Cette isolation permet notamment de ne perdre qu'un environnement et non toute l'installation de python en cas de problèmes, notamment de concurrence entre différents modules.

En outre on s'assure ainsi que le projet fonctionne bien indépendamment sans être affecté par des modules qui n'avaient pas été prévus à la base.

2

On utilisera principalement deux outils pour l'installation de bibliothèques, conda et pip :

```
|	conda install numpy
|	pip install numpy
```

On notera que la méthode conda permet d'installer moins de bibliothèques que l'outil le plus communément utilisé qui est pip. Cependant les distributions conda permettent une meilleure stabilité car les bibliothèques sont validées.

Dans le cas de numpy, ce point de détail n'est pas un soucis car c'est un outil suffisamment utilisé pour être présent partout.

3

Un décorateur est une fonction d'ordre supérieur qui prend en entrée une fonction afin d'en modifier le comportement.

On crée un décorateur comme une fonction prenant en entrée une fonction et renvoyant une fonction modifiée :

```
|       def deco(function)
|               def new_function(args de function) :
|                       actions du decorateur
|                       return function(args de function)
|               return new_function
```

Une fois ce décorateur créé, on peut l'appeler en le plaçant avec un @ au dessus d'une fonction auquel on veut l'appliquer

```
|       @deco
|       def function…
```

4

Le fait qu'un code ne renvoie pas d'erreur ne signifie pas qu'il a le comportement attendu. Pour s'assurer qu'il fonctionne comme prévu il nous faut tester le code en proposant quelques cas dont on connaît le résultat et en vérifiant ce fonctionnement.

Pour cela on utilisera la librairie doctest (qu'il faut donc importer) qui s'appuie sur les docstrings. Dans ceux-ci il est de bon usage de donner des exemples de fonctionnement, par exemple :

```
|       def sum(a,b) :
|       """returns a + b
|       >>> sum(1,1)
|       2
|       >>> sum(-1, 4)
|       3
```

On peut ensuite appeler la fonction doctest() qui ira chercher ces différents exemples dans les docstring et vérifiera qu'ils donnent le résultat attendu.

5

Une constante en python est usuellement représenté par un nom en majuscule, on propose donc pour cette variable l'affectation suivante :

```
|       GRAVITATION = 9.81
```

1.3 Commentaire de code

1

La première valeur de entry est -1. log(-1) n’étant pas défini, la ligne 4 soulèvera une erreur. On pourra insérer dans la fonction un except permettant de proposer un comportement dans cette situation, par exemple en renvoyant -Inf.

2

On pourra, à chaque entrée de la ligne de code réaliser une assertion que entry est strictement positif.

3

L’optimisation vise à rendre le programme plus efficace et moins coûteux tant en matière de mémoire qu’en matière de temps. On pet pour cela identifier les parties les plus critiques du programme et chercher à les corriger. Une méthode peut être de chercher à diminuer la complexité du programme mais cela ne fonctionne pas toujours aussi l’optimisation reviendra souvent à pré compiler certaines parties du programme les plus coûteuses.

4

```
import numba
|       @numba.jit(noPython = True)
|       def LogSum(array) :
|               res = 0
|               for entry in array :
|                       try :
|                               assert  entry > 0
|                       except AssertionError :
|                               print(f"Error log : log({entry}) is not defined. Returns -Inf")
|                               return -Inf
|                       res += log(entry)
|               return res
```