UNIVERSITÉ COTE D'AZUR
L1
2023-2024

Julien Deantoni

Introduction à l'Informatique par le Web

TD numéro 2

Variable et mémoire, premiers algorithmes

Objectif

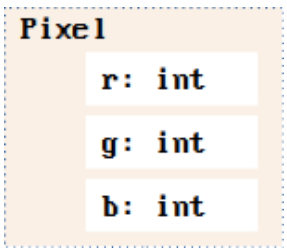
Le premier objectif de ce TD est de comprendre comment évolue les valeurs dans la mémoire en fonction d'algorithmes que l'on vous donne. Le second objectif est de vous faire écrire, en pseudo-code, des algorithmes permettant de répondre à des problèmes particuliers. Les algorithmes que l'on vous donne sont écrits en Python exécutable.

Rappels

- Il existe des variables mutables et immutables. Toutes les variables dans la mémoire ont deux parties. Un premier emplacement contenant la valeur de la variable et un deuxième emplacement contenant le nom de la variable. Le nom permet d'accéder à la valeur (slides 30 à 43 du cours #2)
- Rappelez vous qu'il est primordiale de comprendre comment l'ordre des pas d'exécution se plaque sur l'ordre des lignes dans le fichier (slides 75 à 82 du cours #2 entre autres)

1 Comprendre un programme en comprenant comment les valeurs dans la mémoire évoluent

Comprenez ce que font les programmes suivants. Pour ce faire vous donnerez la valeur des variables à chaque étape ainsi qu'une représentation (artistique ;)) des éléments dans la mémoire. Si vous pensez qu'une erreur est produite, expliquez pourquoi, ignorez la ligne et continuez l'exécution.
Ces exemples reposeront sur le type de donnée `Pixel` qui est un type structuré composé de 3 variables `r`, `g`, `b` de type entier naturels positifs entre 0 et 255.
la représentation de ce type structuré est la suivante:



Pour rappel, le type Pixel peut s'écrire en python comme ceci:

Le type structuré `Pixel`

```
from dataclasses import dataclass

@dataclass
class Pixel:
        r: int
        g: int
        b: int
```

On crée 3 variables de type `Pixel` et on crée un liste dans laquelle on met ces 3 variables:

définition d'une liste de `Pixel`

```
p1: Pixel = Pixel(0,0,255)
p2: Pixel = Pixel(255,255,255)
p3: Pixel = Pixel(r=255,g=0,b=0)

allPixels: list[Pixel] = [p1, p2, p3]
```

basé sur les variables définies dans le code précédent on définit 4 algorithmes. Détaillez ce que font ces 4 algorithmes différents et l'impact sur la liste `allPixels` après chacun des algorithmes. Notez que la liste est réinitialisée aux valeurs ci dessus avant l'exécution de chaque nouvel algorithme.

algorithme #0

```
p1.r = 255
```

algorithme #1

```
index: int = int(0)
while (index < len(allPixels)):
        p: Pixel = allPixels[index]
        p.r = 0
        p.g = 0
        p.b = 0
        index = index + 1
```

algorithme #2

```
index: int = int(0)
while (index < len(allPixels)):
        p: Pixel = allPixels[index]
        p = Pixel(0,0,0)
        index = index + 1
```

algorithme #3

```
index: int = int(0)
while (index < len(allPixels)):
        allPixels[index] = Pixel(0,0,0)
        index = index + 1
```

algorithme #4

```
black: Pixel = Pixel(0,0,0)

index: int = int(0)
while (index < len(allPixels)):
        allPixels[index] = black
        index = index + 1
```

Que ce passe t'il si je rajoute la ligne suivante à la suite de l'algorithme 4:

complément algorithme #4

```
1 black.r=42
```

algorithme #5

```
1 allPixels = allPixels + [Pixel(1,2,3), Pixel(2,3,4), Pixel(127,89,64)]
2
3 index: int = int(len(allPixels))
4 while (index > 0):
5         index = index - 1
6         p: Pixel = allPixels[index]
7         if ((index % 2) == 0):
8                 p.r = 255
9         if ((index %4) == 0):
10                p.b = 255
```

2 Écrire un algorithme

2.1 recopie d'une liste

Soit une liste de pixels nommée `inputPixels`. Écrire un algorithme permettant de créer une nouvelle liste nommée `outputPixels` contenant des pixels ayant la même valeur que ceux de la liste `inputPixels`. Cependant assurer vous que la modification des éléments de `inputPixels` n'impacte pas la liste `outputPixels`.

2.2 Il faut moyenner

Soit une liste d'entiers naturels nommée `lin`. Écrire un algorithme permettant de créer une nouvelle liste nommée `lout`. Le premier élément de `lout` sera la moyenne des 3 premiers éléments de `lin`. La deuxième valeur de `lout` sera la moyenne des 3 prochains éléments dans `lin`, etc, jusqu'à ce qu'il y ait moins de trois éléments non utilisés dans `lin`

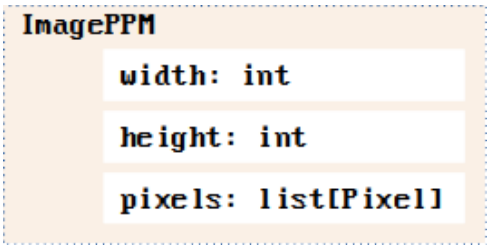
2.3 Position d'un pixel

Soit une liste de pixels nommée `pixels` représentant les pixels d'un image. Soient 2 entiers naturels nommés `largeur` et `hauteur` donnant en nombre de pixels la dimension de l'image. Écrire un algorithme permettant d'afficher la position de que pixel sous la forme suivante: `"pixels[i] => (x ; y)"`.

2.4 Modification des pixels d'une image

2.4.1 Niveaux de gris

Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est un type structuré défini comme ceci:



`pixels` est une liste contenant des variables de type `Pixel` tel que défini précédemment. Écrire un algorithme permettant transformer l'image en niveaux de gris. Plus précisément, pour chaque pixel $p \in pixels$, changer la valeur de ses variables internes `r`,`g` et `b` par une même valeur v obtenue par la formule suivante: $v = 0,2126 * p.r + 0,7152 * p.g + 0,0722 * p.b$

2.4.2 Changement de couleur

Soit une variable nommée `img` de type `ImagePPM`. `ImagePPM` est le type définit précédemment.
Soit une variable nommée `pRef` de type `Pixel`. Soit une autre variable nommée `Pnew` de type `Pixel`.
Écrire un algorithme permettant changer tous les pixels de `img` qui sont égaux à `pRef` par des pixels égaux à `pNew`.

3 Exercice optionnel

3.1 Éclaircissements des pixels autour d'une image

Soit une variable nommée `img` de type `Image`.
Écrire un algorithme permettant de modifier les pixels se trouvant sur les bords de l'image. Plus précisément, pour tous les pixels étant à une distance inférieure à 15 d'un bord de l'image, on désir appliquer un éclaircissement de celui-ci. Pour éclaircir un pixel, nous allons augmenter la valeur de chacune de ses variables internes `r`,`g` et `b` de 100; tout en s'assurant de ne pas dépasser 255.