Algorithmique et Programmation 1

Lundi 21 octobre 2024

L1 Mathématiques - L1 Informatique
Semestre 1

CC1 : Comment et quoi ?

Horaire

Lundi 4 novembre 13h-15h (tiers-temps : 15h40) en A1, A2, A5

CC1 : Comment et quoi ?

Horaire

Lundi 4 novembre 13h-15h (tiers-temps : 15h40) en A1, A2, A5

Contenu

- Tout ce qu'on a vu, y compris les séances de cette semaine
- Les slides et les notes de cours
- QCM
- Questions de programmation écrites
- Mini-problème

CC1 : Comment et quoi ?

Horaire

Lundi 4 novembre 13h-15h (tiers-temps : 15h40) en A1, A2, A5

Contenu

- Tout ce qu'on a vu, y compris les séances de cette semaine
- Les slides et les notes de cours
- QCM
- Questions de programmation écrites
- Mini-problème

Sur la triche

- L'examen est **individuel**
- Tolérance zéro

Questions ouvertes et mini-problème

Voir le sujet de l'an dernier

Approfondissements : les listes (de listes (de listes (...)))

Et après ?

Liste de listes (de listes de listes de ....)

Et si on faisait un jeu de « Morpion » ?

Démo Thonny : brouillon_morpion.py

Revenons à notre morpion

Démo sur Thonny

Revenons à notre morpion

Démo sur Thonny

Les leçons à en tirer

- Une liste de listes se comporte comme une matrice : `grille[i][j]` permet d'accéder à la case (i, j)
  (et la modifier)
- Lorsqu'on utilise plusieurs fois une liste, c'est la même liste même si on a l'impression de l'avoir copiée
- Factoriser le code en fonctions permet de le rendre plus lisible
- Ces fonctions peuvent modifier la liste donnée en argument

La liste de listes comme matrice

```
grille = [
  #   0    1    2
     ["a", "b", "c"], # 0
     ["d", "e", "f"], # 1
     ["g", "h", "i"]  # 2
]
```

Accéder à une ligne et à une case

```
l = grille[1]
>>> ["d", "e", "f"]
l[2]
>>> "f"
```

```
(grille[1])[2]
>>> "f"
grille[1][2]
>>> "f"
```

La liste de listes comme matrice

```
grille = [
  #   0    1    2
    ["a", "b", "c"], # 0
    ["d", "e", "f"], # 1
    ["g", "h", "i"]  # 2
]
```

Modifier une case

```
l = grille[1]
>>> ["d", "e", "f"]
l[2] = "j"
print(grille)
>>> [['a', 'b', 'c'], ['d', 'e', 'j'], ['g', 'h', 'i']]
```

La liste de listes comme matrice

```
grille = [
  #   0    1    2
     ["a", "b", "c"], # 0
     ["d", "e", "j"], # 1
     ["g", "h", "i"]  # 2
]
```

Modifier une case (suite)

```
(grille[2])[2] = "k"
print(grille)
>>> [['a', 'b', 'c'], ['d', 'e', 'j'], ['g', 'h', 'k']]
grille[0][1] = "l"
>>> [['a', 'l', 'c'], ['d', 'e', 'j'], ['g', 'h', 'k']]
```

La copie superficielle

```
ligne = ["a", "b", "c"]
grille = [ligne, ligne, ligne]
print(grille)
>>> [["a", "b", "c"], ["a", "b", "c"], ["a", "b", "c"]]
```

Modifier une (des ?) case

```
ligne[1] = "d"
print(grille)
```

La preuve sur Python Tutor.

La copie superficielle

```
ligne = ["a", "b", "c"]
grille = [ligne, ligne, ligne]
print(grille)
>>> [["a", "b", "c"], ["a", "b", "c"], ["a", "b", "c"]]
```

Modifier une (des ?) case

```
ligne[1] = "d"
print(grille)
>>> [['a', 'd', 'c'], ['a', 'd', 'c'], ['a', 'd', 'c']]
```

La preuve sur Python Tutor.

La copie superficielle

```
ligne = ["a", "b", "c"]
grille = [ligne, ligne, ligne]
print(grille)
>>> [["a", "b", "c"], ["a", "b", "c"], ["a", "b", "c"]]
```

Modifier une (des ?) case

```
ligne[1] = "d"
print(grille)
>>> [['a', 'd', 'c'], ['a', 'd', 'c'], ['a', 'd', 'c']]
grille[0][2] = "e"
```

La preuve sur Python Tutor.

La copie superficielle

```
ligne = ["a", "b", "c"]
grille = [ligne, ligne, ligne]
print(grille)
>>> [["a", "b", "c"], ["a", "b", "c"], ["a", "b", "c"]]
```

Modifier une (des ?) case

```
ligne[1] = "d"
print(grille)
>>> [['a', 'd', 'c'], ['a', 'd', 'c'], ['a', 'd', 'c']]
grille[0][2] = "e"
>>> [['a', 'd', 'e'], ['a', 'd', 'e'], ['a', 'd', 'e']]
```

La preuve sur Python Tutor.

Les fonctions sur les listes

Mutabilité, immutabilité

Rappel

Une fonction ne peut pas modifier[1] une variable de type `int` donnée en argument :

```
def essaie_de_modifier_pour_voir(x):
    x = 5

x = 12
essaie_de_modifier_pour_voir(x)
print(x)
>>> 12
```

C'est aussi vrai pour tous les types "primitifs" :

`int`, `float`, `str`, `bool`.

On dit que les types primitifs sont immutables.

1.

Mutabilité, immutabilité

Rappel

Une fonction ne peut pas modifier[1] une variable de type `int` donnée en argument :

```
def essaie_de_modifier_pour_voir(x):
    x = 5

x = 12
essaie_de_modifier_pour_voir(x)
print(x)
>>> 12
```

C'est aussi vrai pour tous les types "primitifs" :

`int`, `float`, `str`, `bool`.

On dit que les types primitifs sont immutables.

1. Sauf en utilisant les mots-clés `global` et `nonlocal`, mais ne faites pas ça svp.

Mutabilité, immutabilité

Une fonction peut modifier la liste donnée en argument :

```
def essaie_de_modifier_pour_voir_liste(lst):
    lst[0] = 5

lst = [12]
essaie_de_modifier_pour_voir_liste(lst)
print(lst)
>>> [5]
```

On dit que les listes sont mutables.

Les itérables

En général : parcours de gauche à droite

Élément par élément

OU

Indice par indice

- Listes
- Chaînes
- *Tuples*
- *Ensembles*
- *Fichiers...*

SEULEMENT POUR LES LISTES ???

NON : « ITERABLES »