rappels_algo_seconde

September 16, 2019

1 Rappels de seconde sur les algorithmes et le langage Python

1.1 I - Affectations et variables

1.1.1 Définition

> Un algorithme est une suite finie d'opérations élémentaires, à appliquer dans un ordre déterminé, à des données.

Dans un algorithme, on utilise des **variables**. Elles représente des nombres ou d'autres objets (listes, chaînes de caractère, ...) et on utilise une lettre ou un mot pour les désigner. On modifie leur valeur lors d'**affectations**.

On peut se représenter mentalement une variable comme une boîte pouvant contenir des objets et munit d'une étiquette : son nom. Ici deux variables a et b.

La variable a est vide et la variable b contient le nombre 3 :

On effectue par exemple les affectations suivantes :

$a \leftarrow 4$
$b \leftarrow a + b$

Voici le résultat de ces affectations en mémoire :

Les algorithmes utilisent chacun des instructions très diverses, mais on peut ranger ces instructions en quatre grandes familles, à découvrir tout au long de l'année : - *entrée/sortie* ou encore *saisie/affichage*: permettent à l'utilisateur d'interagir avec l'algorithme en précisant une valeur lors de l'utilisation et désignant le résultat obtenu ; - *affectation de variables* : définissent ou modifient la valeur d'une variable ; - *instructions conditionnelles* : elles permettent de tester des conditions et proposer des choix ; - *boucles* : elles permettent de répéter des instructions.

1.1.2 Exemple

On considère l'algorithme de calcul suivant : - Choisir un nombre entier n. - Lui ajouter 4. - Multiplier la somme obtenue par le nombre choisi. - Ajouter 4 à ce produit. - Écrire le résultat r.

Il est utile d'écrire les algorithmes dans un langage proche de celui utilisé par les ordinateurs. Ici, l'algorithme serait :

Saisir n
$r \leftarrow n + 4$
$r \leftarrow r \times n$
$r \leftarrow 4 + r$
Afficher r

Voici la traduction de cet algorithme en Python :

```
In [3]: # Pour executer une cellule on appuie simultanément sur MAJ et ENTRER
        %load_ext tutormagic
```

```
In [ ]: %%tutor --lang python3 --height 500
        # Pour voir l'exécution du script pas à pas
        n = eval(input("Saisir n : "))
        r = n + 4
        r = r * n
        r = 4 + r
        print(r)
```

1.1.3 Exercice 1

Écrire les scripts Python, sur le modèle précédent, correspondants aux expressions suivantes (il ne doit y a voir qu'une seule opération élémentaire par ligne) :

1) $7(x+2)^2$;

```
In [1]: x=eval(input("x = "))
        y=x+2
        print(y)
```

```
x = 5
7
```

2) $(7x+2)^2$;

```
In [ ]:
```

3) $7x^2+2$;

```
In [ ]:
```

4) $(7x)^2+2$;

```
In [ ]:
```

5) $\frac{7x}{2}+3$;

```
In [ ]:
```

6) $\frac{7x+2}{2}$.

```
In [ ]:
```

1.1.4 Exercice 2

On considère l'algorithme ci-dessous :

Saisir p
$c \leftarrow p - 1$
$p \leftarrow p + 1$
$p \leftarrow p \times p - c \times c$

1. Qu'obtient-on à la fin de l'algorithme pour : $p = 2$? pour $p = 5$?
2. Le traduire en Python ci-dessous pour vérifier vos réponse :

```
In [ ]:
```

1.1.5 Exercice 3

On considère l'algorithme ci-dessous :

Saisir a et b $n \leftarrow 10 \times a + b$ **Afficher** n $c \leftarrow a$ $a \leftarrow b$ $b \leftarrow c$ $n \leftarrow 10 \times a + b$ **Afficher** n

1. Utiliser un tableau, dont les entrées sont les variables de cet algorithme, afin de le tester pour $a = 2$ et $b = 3$.

Variables	-	-	-	-
a	2			
b	3			
c				
n				

2. Reprendre le travail avec un autre couple d'entiers compris entre 0 et 9.
3. Expliquer l'importance de la variable c.
4. Le traduire en Python ci-dessous pour vérifier vos réponse :

```
In [ ]:
```

1.1.6 Exercice 4

On considère deux points dans un repère orthonormé du plan.

Compléter le script suivant afin qu'il affiche les coordonnées du milieu I d'un segment $[AB]$ et le carré de la longueur de ce segment.

```
In [ ]: # Pour saisir les données (ne rien modifier dans cette partie)
        A = input("Saisir les coordonnées de A séparées d'une virgule : ").split(',')
        x_A, y_A = int(A[0]), int(A[1])
        B = input("Saisir les coordonnées de B séparées d'une virgule : ").split(',')
        x_B, y_B = int(B[0]), int(B[1])

        # Compléter la suite
        x_I, y_I = ...
```

```
AB_carre = ...
print(...)
```

1.2 Instructions conditionnelles

1.2.1 Définition

La structure `si ... alors ... sinon ...` (qui se traduit par `if ... then ... else` en anglais) permet de définir une condition: **si** cette condition est remplie, **alors** on effectuera certaines instructions ; **sinon** on effectuera d'autres instructions.

La structure générale est la suivante :
Si Condition **alors**
> Traitement 1
sinon
> Traitement 2

La condition est soit *vraie* soit *fausse*, si elle est vraie le Traitement 1 est effectué, si elle est fausse c'est le Traitement 2 qui est effectué. L'intruction `sinon` n'est pas obligatoire.

Sa traduction en Python est la suivante :

```
In [ ]: if Condition:
            Traitement1
        else:
            Traitement2
```

1.2.2 Exercice 5

Pour les résultats du baccalauréat, l'ordinateur indique : - `admis` si l'élève a obtenu à l'écrit une moyenne supérieure ou égale à 10 ; - `oral` si sa moyenne appartient à l'intervalle $[8;10[$; - `recalé` sinon.

Écrire un script Python affichant les résultats suivant la note saisie par l'utilisateur :

```
In [ ]:
```

1.2.3 Exercice 6

La fonction f est définie sur l'intervalle $[-5;6]$ par l'algorithme ci-dessous :

Saisir x
Si $x \leq 1$ **alors**

$y \leftarrow -3x - 7$

Sinon Si $x \geq 3$

$y \leftarrow -\frac{1}{3}x + 5$

Sinon > $y \leftarrow 2x - 2$

Afficher $f(x) = y$

1) Qu'affiche cette fonction pour les valeur -4, -1, 0, 3 et 6.

2) Écrire le script Python correspondant ci-dessous (Sinon Si se traduit par elif) :

```
In [ ]: # Pour afficher le résultat utiliser la ligne suivante
        print(f"f({x})={y}")
```

1.3 Les boucles

1.3.1 Définition

Une **boucle bornée** est utilisée lorsque l'on veut **répéter un certain nombre de fois** les mêmes instructions.

Remarque

Une variable sera utilisée pour **compter** le nombre de répétitions, on dit parle ainsi **d'itérations**. En général, elle prendra des valeurs entières. À chaque itération, la variable est incrémentée d'une unité (sauf indication contraire).

La structure générale est la suivante :
Pour i **allant de** 0 **jusqu'à** 9 **faire** > Instructions
FinPour
Dans cette boucle, les *instructions* sont répétées 10 fois.

Remarque

La variable i prendra successivement les valeurs : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Sa traduction en Python est la suivante :

```
In [21]: for i in range(10):
             Instructions

17.0
```

1.3.2 Exercice 7

On considère l'algorithme ci-dessous :
Saisir n
s $\leftarrow$ 0
Pour i **allant de** 1 **jusqu'à** n **faire** > s $\leftarrow$ s + i
FinPour
Afficher s

1. Que retourne cet algorithme pour $n = 5$?
2. Que fait cet algorithme ?
3. Traduire cet algorithme en Python :

```
In [ ]:
```

1.4 Les fonctions

1.4.1 Information

Les fonctions permettent de décomposer un algorithme complexe en une série de sous-algorithmes plus simples, lesquels peuvent à leur tour être décomposés en fragments plus petits, et ainsi de suite. Une fonction **retourne** en **sortie** un ou des objet(s) et/ou exécute une tâche, pour cela on peut lui fournir un ou des **arguments** en **entrée**.
Une fonction est ensuite **appelée** dans le cur de l'algorithme.

La structure générale est la suivante :
Définition Fonction(paramètres):
> Instructions
> Retourne objets
FinDéfinition
...
`# Appel de la fonction`
A ← Fonction(valeurs des paramètres)
Sa syntaxe Python est la suivante :

```
In [ ]: def Fonction(paramètres):
            Instruction
            return objets

        # Appel de la fonctio
        A = Fonction(valeurs)
```

1.4.2 Remarques :

- Il faut bien distinguer la définition de la fonction et son appel à l'intérieur de l'algorithme.
- Lors de cet appel, l'objet retourné par la fonction est affecté à la variable A.

1.4.3 Exercice 8

Dans la fonction définie ci-dessous, les paramètres a et b sont positifs et $b \neq 0$:
Définition mystère(a,b):
> $r \leftarrow a$
> $q \leftarrow 0$
> **Tant que** $r \geqslant b$ **faire**
> > $r \leftarrow r - b$
> > $q \leftarrow q + 1$
> > **FinTantque**
> Retourne r
FinDéfinition

Remarque

Lorsque l'instruction **Retourne** est rencontrée, on sort de la fonction

1. Tester la fonction avec les paramètres 15 et 5, puis, avec 17 et 6.

2. Expliquer pourquoi la boucle se termine toujours.

3. Montrer qu'avant le début de la boucle ainsi qu'à la fin de chaque itération de la boucle, on a toujours $a = bq + r$.

4. Vérifier que la valeur retournée est toujours strictement inférieur à b. Finalement, que fait cette fonction ?

5. Traduire cette fonction en Python.

```
In [ ]:
```

6. Vérifier vos résultats de la question **1.** en appelant cette fonction avec les valeurs correspondantes :

```
In [ ]:
```

```
In [ ]:
```

1.4.4 Définition

> On appelle nombre premier, un nombre entier naturel ayant exactement deux diviseurs entiers naturels : 1 et lui-même.

7. Écrire une fonction `premier` qui retourne `True` si le nombre est premier et `False` sinon.

```
In [ ]:
```