

TP n° 03 – Algorithmes et programmation

Exercice 1. Assigner une valeur à une variable.

Au départ, on a assigné à deux variables les valeurs suivantes :

`a=2`

`b=3`

L'objectif est d'échanger les valeurs des deux variables. Pour chaque exemple, prédire la valeur finale de chaque variable et vérifier avec Python.

<code>a=b</code>		<code>a=x</code>		<code>x=a</code>		<code>(a,b)=(b,a)</code>
<code>b=a</code>	<code>a+3=b</code>	<code>a=b</code>	<code>a=b</code>	<code>a=b</code>	<code>b=x</code>	
		<code>b=x</code>		<code>b=x</code>		

Exercice 2. De l'importance de l'indentation

On considère les deux algorithmes suivants :

```
if a>=5:
    a=a-2
if a<4:
    a=a+5
```

Algorithme n°1

```
if a>=5:
    a=a-2
    if a<4:
        a=a+5
```

Algorithme n°2

Dans les deux cas, déterminer la valeur finale de `a` pour `a = 0, 1, 2, ..., 10`.

Exercice 3. Quelle est la différence entre les deux fonctions suivantes ? Déterminer une valeur a telle que $f(a) \neq g(a)$.

```
def f(a):
    if a<-10:
        b=a-2
    elif a>15:
        b=2*a
    else :
        b=a
    return(b)
```

```
def g(a):
    if a<-10:
        b=a-2
    if a>15:
        b=2*a
    else :
        b=a
    return(b)
```

Exercice 4. Soit $f: \mathbb{R} \rightarrow \mathbb{R}$ la fonction définie par :

$$f: x \mapsto \begin{cases} 2x & \text{si } x \leq 0 \\ x+1 & \text{si } 0 < x \leq 1 \\ x^2 & \text{si } x > 1 \end{cases}$$

1. Programmer la fonction f en utilisant `if`, `elif`, `else`.
2. Programmer à nouveau la fonction f sans utiliser `elif` mais en utilisant deux tests `if` imbriqués. On commencera par faire l'organigramme correspondant.

Exercice 5. Dans le programme ci-dessous, `a` et `b` sont des entiers positifs.

```
while a>=b:
    a=a-b
```

1. Partant de `a=17` et `b=4`, notez à chaque étape la valeur de `a` et de `b`. Quelle est la valeur finale de `a` ?
2. En général, que vaut `a` à la fin du programme ?

Rappel. Lorsqu'on travaille avec des entiers naturels, la fonction `a//b` renvoie le quotient de la division euclidienne de `a` par `b` et `a%b` renvoie le reste de cette division euclidienne.

Exercice 6. Premières boucles `for`.

Recopier le programme suivant et exécuter-le.

```
for i in range(10):  
    print(i)
```

1. Modifier le programme pour qu'il affiche les nombres de 0 à 20 inclus.
2. Modifier le programme pour qu'il affiche les nombres de 0 à 20 dans l'ordre décroissant.
3. Modifier le programme pour qu'il affiche les nombres de 0 à 20 qui sont pairs.

Remarque. Sous l'environnement IDLE, vous pouvez obtenir des informations sur un objet Python. Dans l'interpréteur interactif, par exemple taper `help(range)` puis Entrée. Il apparaît soit directement un texte d'aide, soit une boîte sur laquelle il est possible de double-cliquer pour dérouler un texte d'aide. La partie utile pour vous est souvent au début.

Une autre méthode pour les fonctions : une info-bulle contenant aide minimale rappelant la syntaxe de la fonction apparaît automatiquement après la saisie de la parenthèse gauche après une fonction.

Exercice 7. Que fait `range(2,40,3)` ?

Refaites la question 3 de l'exercice 6 en utilisant la fonction `range` avec trois paramètres.

Exercice 8. Premiers termes d'une suite.

On considère le programme suivant :

```
u=0  
for i in range(10):  
    u=2*u+3
```

1. Faire tourner le programme.
2. Le programme calcule les dix premiers termes d'une suite définie par récurrence. Modifier le programme pour qu'il affiche tous les termes calculés.
3. Compléter la définition de la suite :

$$\begin{cases} u_0 = \dots \\ u_{n+1} = \dots \end{cases}$$

4. On considère maintenant la suite de Fibonacci définie par :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall n \in \mathbb{N}, \quad F_{n+2} = F_{n+1} + F_n \end{cases}$$

Ecrire un programme qui permet de calculer les dix premiers termes de la suite $(F_n)_{n \in \mathbb{N}}$.

Exercice 9. Dans cet exercice, n est une variable déjà initialisée, entière. Que fait le programme suivant :

```
while n>0:
    r=n%10
    n=n//10
```

Modifier le programme pour qu'il retourne la somme des chiffres d'un entier.

Exercice 10. On considère le programme suivant :

```
1 import random
2
3 nombre=random.randrange(10)           # la variable 'nombre' est un entier
4                                       # pris au hasard entre 0 et 9
5 essai=input('entrer une valeur entre 0 et 9 ') # la variable 'essai' est un nombre
6                                       # entré par l'utilisateur
7 while essai!=nombre:
8     essai=input('entrer une nouvelle valeur ')
9 print('Vous avez gagné')
```

1. Recopier le programme dans un fichier Python (sans les commentaires)
2. Que fait le programme ?
3. Modifier le programme pour qu'il affiche le nombre d'essais qui ont été nécessaires pour trouver le nombre.

Exercice 11. Voici deux programmes. n est une variable entière, déjà initialisée. L'un des deux compte le nombre de diviseurs positifs de n . Ce nombre est stocké dans la variable p . Que fait l'autre algorithme ?

```
1 p=0
2 k=1
3 while k<=n :
4     if n%k == 0 :
5         p = p+1
6     k = k+1
```

Algorithme n°1

```
1 p=0
2 k=1
3 if n%k == 0 :
4     while k<=n :
5         k = k+1
6     p=p+1
```

Algorithme n°2

Exercice 12. Crible d'Eratosthène.

Le crible d'Eratosthène est un algorithme qui étant donné un entier n renvoie les nombres premiers compris entre 2 et n . Il fonctionne de la manière suivante :

- on inscrit tous les entiers entre 2 et n .
- 2 est premier. On stocke cette valeur et on élimine tous les multiples de 2.
- on prend le premier entier qui vient après 2 qui n'a pas été éliminé : c'est 3. On stocke cette valeur et on élimine tous les multiples de 3.
- on réitère l'opération jusqu'à n .

1. Testez l'algorithme à la main pour déterminer les nombres premiers entre 2 et 14.
2. En Python, l'algorithme est codé par le programme suivant. La liste `premier` contient la réponse.

```

1 premiers=[]
2 nombres = []
3 for i in range(2,n+1):
4     nombres.append(True)
5     # nombres=[True,True,...]
6     #On y stockera l'information suivante :
7     #True : le nombre est premier, False : il ne l'est pas
8     for i in range(2,n+1):          # i parcourt les entiers de 2 à n
9         if nombres[i-2]==True:
10             premiers.append(i)
11     # si i est marqué comme True,
12     # c'est un nombre premier : on le stocke dans la liste 'premier'
13         for j in range(2*i,n+1,i):
14             nombres[j-2] = False
15     # les multiples de i qui sont compris entre 2i et n
16     # sont alors marqués comme False

```

3. Vérifier que l'algorithme donne les mêmes valeurs que celles calculées en 1) pour $n=14$.
4. Dans cet algorithme, i parcourt tous les entiers entre 2 et n . Or, si i n'est pas premier, alors l'un de ces facteurs est au moins inférieur à \sqrt{n} ¹. Donc, dans l'algorithme, à partir de $i = \sqrt{n}$ (ou plutôt l'entier directement supérieur à \sqrt{n}), les nombres non premiers sont déjà marqués **False**. Modifier le programme pour qu'il affiche encore les nombres premiers inférieurs à n mais en évitant que i parcourt les entiers supérieurs à \sqrt{n} .

1. On suppose que $i \leq n$ n'est pas premier. Il se décompose en $i = pq$. Si $p > \sqrt{n}$ et $q > \sqrt{n}$, alors $i > n$. On dépasse alors n .