TP n° 03 – Algorithme 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 chaques variables et vérifier avec Python.

```
a=b
b=a
```

```
a+3=b
```

```
a=x
a=b
b=x
```

```
x=a
a=b
b=x
```

```
(a,b)=(b,a)
```

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` $= 0,\ 1,\ 2, \cdots 10$.

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 $f\colon \mathbb{R} \to \mathbb{R}$ la fonction définie par :

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

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 Spyder, vous pouvez obtenir des informations sur un objet Python. Par exemple, taper `range` dans l'éditeur ou la console. Sélectionner-le puis taper **Ctrl i**. Dans la fenêtre en haut à droite, il apparait une documentation (en anglais) sur la fonction, ainsi que les différentes variables dont elle dépend.
Une autre méthode : l'aide apparaît automatiquement après la saisie d'une parenthèse gauche après un objet.

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

```
import random

nombre=random.randrange(10)                    # la variable 'nombre' est un entier
                                               # pris au hasard entre 0 et 9
essai=input('entrer une valeur entre 0 et 9 ') # la variable 'essai' est un nombre
                                               # entré par l'utilisateur
while essai!=nombre:
   essai=input('entrer une nouvelle valeur ')
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 ?

```
p=0
k=1
while k<=n :
   if n%k == 0 :
      p = p+1
   k = k+1
```

Algorithme n°1

```
p=0
k=1
if n%k == 0 :
   while k<=n :
      k = k+1
   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.

```
premiers=[]
nombres = []
for i in range(2,n+1):
        nombres.append(True)
# nombres=[True,True,...]
#On y stockera l'information suivante :
#True : le nombre est premier, False : il ne l'est pas
for i in range(2,n+1):              # i parcourt les entiers de 2 à n
        if nombres[i-2]==True:
                premiers.append(i)
# si i est marqué comme True,
# c'est un nombre premier : on le stocke dans la liste 'premier'
                for j in range(2*i,n+1,i):
                        nombres[j-2] = False
# les multiples de i qui sont compris entre 2i et n
# 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.

Correction TP n° 03 – Algorithme et programmation

Solution 1.
1. `a=3` et `b=3`
2. message d'erreur : ne peut pas assigner avec un opérateur
3. message d'erreur : `x` n'est pas définie
4. `a=3` et `b=2`. On a échangé les valeurs.
5. `a=3` et `b=2`. On a échangé les valeurs.

Solution 2. Algorithme n°1 :

Valeur initiale de a	0	1	2	3	4	5	6	7	8	9	10
Valeur finale de a	5	6	7	8	4	8	4	5	6	7	8

Algorithme n°2 :

Valeur initiale de a	0	1	2	3	4	5	6	7	8	9	10
Valeur finale de a	0	1	2	3	4	8	4	5	6	7	8

Solution 3. Dans le premier cas, le test `else` est vérifié pour `a` qui n'est ni dans $]-\infty,-10[$, ni dans $]15,+\infty[$. Donc, `a` vérifie ce test pour $a \in [-10,15]$.
Dans le deuxième cas, `else` est le contraire du dernier `if`, donc le test est vérifié pour `a` qui n'est pas dans $]15,+\infty[$. Donc, `a` vérifie ce test pour $a \in]-\infty,15]$.
Par exemple, $f(-11)$ renvoie -13 mais $g(-11)$ renvoie -11.

Solution 4.
1.
```
def f(x):
    if x <= 0 :
        y = 2*x
    elif x > 0 and x <= 1 :
        y = x+1
    else :
        y = x**2
    return(y)
```

D'autres solutions sont possibles, comme :

```
def f(x):
    if x <= 0 :
        y = 2*x
    elif x > 1 :
        y = x**2
    else :
        y = x+1
    return(y)
```

2.
```
def f(x):
    if x<=0 :
        y = 2*x
    else :
        if x <= 1 :
            y = x+1
         else :
            y = x**2
    return(y)
```

Solution 5. 1.

Etape	1	2	3	4	
Valeur de `a` avant	17	13	9	5	1
Le critère est-il vérifié ?	oui	oui	oui	oui	non
Valeur de `a` après	13	9	5	1	

La valeur finale de `a` est 1.

2. `a%b`

Solution 6. 1.

```
for i in range(21):
    print(i)
```

2.
```
for i in range(21):
    print(20-i)
```

3.
```
for i in range(11):
    print(2*i)
```

Solution 7.

```
for i in range(0,21,2):
    print(i)
```

Solution 8. 1. .

2.
```
u=0
for i in range(9):
    u=2*u+3
```

3. $\begin{cases} u_0 = 0 \\ u_{n+1} = 2u_n + 3 \end{cases}$

4.
```
u=0
v=1
for i in range(10):
    (u,v)=(v,u+v)
print(v)
```

Solution 9. Le programme affiche les chiffres du nombre `n`.

```
s=0                 # on initialise la somme a zero
while n>0:
    r=n%10          # r est le dernier chiffre de n
    s=s+r           # on ajoute r a la somme
    n=n//10         # on recommence avec n auquel on a retiré le dernier chiffre
```

Solution 10.

```
import random

nombre=random.randrange(10)                    # la variable 'nombre' est un entier
                                               # pris au hasard entre 0 et 9
compteur=1                                     # on initialise le compteur a 1
essai=input('entrer une valeur entre 0 et 9 ') # la variable 'essai' est un nombre
                                               # entré par l'utilisateur
while essai!=nombre:
   essai=input('entrer une nouvelle valeur ')
   compteur=compteur+1                         # a chaque coups, compteur augmente de 1
print('Vous avez gagné en ',compteur,' coups')
```

Solution 11. Le premier programme compte le nombre de diviseurs de n.

```
p=0
k=1                          # k va parcourir tous les entiers de 1 a n
while k<=n :                 # pour k <=n
   if n%k == 0 :             # si k divise n,
      p = p+1                # on ajoute 1 aux nombres de diviseurs de k
   k = k+1                   # on passe a l'entier suivant
```

Le deuxième programme renvoie 1 :

```
p=0
k=1
if n%k == 0 :        # k=1, donc k verifie la condition
   while k<=n :      # tant que k<n
      k = k+1        # k augmente de 1
   p=p+1             # on sort de la boucle while. p augmente de 1, il vaut maintenant 0+1
```

Solution 12. 1. Nombres premiers entre 2 et 14 :
$2, 3, \underline{4}, 5, \underline{6}, 7, \underline{8}, \underline{9}, \underline{10}, 11, \underline{12}, 13, \underline{14}$.

2.
```
n=100
premiers=[]
nombres = []
N=int(sqrt(n))+1                    # N est l'entier strictement superieur à racine de n

for i in range(2,n+1):
    nombres.append(True)            # nombres=[True,True,...] : on y stocke l'information
                                    # True : le nombre est premier ou False : il ne l'est pas
for i in range(2,N+1):              # i parcourt les entiers de 2 à N
    if nombres[i-2]==True:          # si i n'est pas False,
        premiers.append(i)          # c'est un nombre premier, on le stocke dans la liste
        for j in range(2*i,n+1,i):  # les multiples de i entre 2i et n
            nombres[j-2] = False    # sont alors marqués comme False


for i in range(N+1,n+1):            # on ajoute a la liste 'premiers' les nombres superieurs a
    if nombres[i-2]==True:          # N qui sont marqués True
        premiers.append(i)
print(premiers)
```