TP – Algorithmique de base en Python

Table des matières

TP 1 – Fonctions et appels de fonctions

Implémentez et testez les fonctions ci-dessous.

1. $f(x) = x + 1$

   Correction.

```
def f(x):
    return x+1

print(f(3))
```

2. $g(x) = 2x + 3$

3. $h(x) = x^2 + 1$

4. $i(x) = 5x^2 + 3x + 1$

5. $j(x) = 3x^3 + 5x^2 + 3x + 4$

6. $k(x) = x^{10} - x^3 + 3$

7. $l(x) = 3x^{10} + 4x^2 + 5$

8. $m(x) = 10x^{15} + 3x + 1$

9. $n(x) = a(x) + b(x)$ où

   $a(x) = x + 3$
   $b(x) = x^2 - 2$

10. $p(x) = c(d(x))$ où

    $c(x) = x + 1$
    $d(x) = x^3$

11. $q(x) = \dfrac{3x^3 + 2}{20x^2 + 1}$

TP 2 – Conditions

Implémentez et testez les fonctions ci-dessous.

1. f(x) = 0 si x < 0
   = 1 sinon.

Correction.

```
def f(x):
    if x < 0:
        return 0
    else:
        return 1

print(f(3))
```

2. g(x) = « toto » si x ≤ 0
   = « tata » sinon.

3. h(c) = 0 si c = ‘a’
   = 1 si c = ‘b’
   = 2 si c = ‘c’
   = 3 sinon.

4. i(x) = 0 si x ≤ 0
   = 1 si 0 ≤ x ≤ 100
   = 2 sinon.

5. j(x) = x^2 si -10 < x < 10
   = x^3 si 50 < x < 60
   = 1 sinon.

6. k(x) = ‘a’ si x = 1
   = ‘b’ si x = 2
   = ‘c’ si x = 3
   = ‘_’ sinon.

7. l(w) = 1 si w = « toto »
= 2 si w = « toutou »
= 0 sinon.

8. m(c) = « haut » si c = ‘h’
= « bas » si c = ‘b’
= « droite » si c = ‘d’
= « gauche » si c = ‘g’
= « autre » sinon.

9. n(x, y) = 1 si x = y
= 0 sinon.

10. p(a, b) = Vrai si a = b
= Faux sinon.

11. q(x, y, z) = Vrai si x = y = z
= Faux sinon.

12. r(x, y, z) = 1 si x > y et x > z
= 2 si y > x et y > z
= 3 si z > x et z > y
= 0 sinon.

13. s(x, y, z) = (1+y+z) / x si $x \neq 0$
= (2+x+z) / y si $y \neq 0$
= (3+x+y) / z si $z \neq 0$
= 0 sinon.

14. t(x, y, z) = $x^4 + (1+y^3 + z^5) / x$ si $x \neq 0$
= $y^4 + (2+x^3 + z^5) / y$ si $y \neq 0$
= $z^4 + (3+x^3 + y^5) / z$ si $z \neq 0$
= 0 sinon.

TP 3 – Procédures

1. Exercice

Écrire une procédure *afficher_arguments* prenant trois arguments, et les affichant sur la console.

Correction.

```
def afficher_arguments(a, b, c):
    print(a, b, c)

afficher_arguments(1, "toto", 3.5)
```

2. Exercice

Écrire une procédure *etes_vous_toto* qui demande à l'utilisateur d'entrer son nom et qui affiche :

- « Vous êtes Toto » si l'utilisateur entre « Toto » ;
- « Vous n'êtes pas Toto » sinon.

3. Exercice

Écrire une procédure *etes_vous_toto2* qui demande à l'utilisateur d'entrer son nom et qui affiche :

- « Vous êtes Toto » si l'utilisateur entre « Toto » ou « toto » ;
- « Vous n'êtes pas Toto » sinon.

4. Exercice

Écrire une procédure *etes_vous_untel* qui prend en argument une chaîne de caractères, par exemple « Tata » ; qui demande à l'utilisateur d'entrer son nom et qui affiche :

- « Vous êtes Tata » si l'utilisateur entre « Tata » ;
- « Vous n'êtes pas Tata » sinon.

5. Exercice

Écrire une procédure *qui_etes_vous* qui demande a l'utilisateur d'entrer son nom, par exemple « Bond » ; puis d'entrer son prénom, par exemple « James » ; et enfin son âge, par exemple « 40 ». Le programme affiche alors :

```
Vous êtes James Bond et vous avez 40 ans.
```

6. Exercice

Écrire une procédure *qui_est_plus_grand* qui se comporte ainsi :

```
>>> qui_est_plus_grand(3, 5)
5 est plus grand que 3
>>> qui_est_plus_grand(5, 3)
5 est plus grand que 3
```

```
>>> qui_est_plus_grand(3,3)
3 et 3 sont égaux !
```

7. Exercice

Écrire une procédure *quelle_heure* qui demande a l'utilisateur d'entrer une heure h (exprimée ici par un entier) et qui affiche :

- « Je vous demande pardon ? » si $h < 0$ ou $h \geq 24$;
- « Il faut dormir ! » si $0 \leq h < 5$;
- « Bon matin ! » si $5 \leq h < 12$;
- « Bon appétit ! » si $12 \leq h < 14$;
- « Bon après-midi ! » si $14 \leq h < 18$;
- « Bon soir ! » si $18 \leq h < 21$;
- « Bonne nuit ! » si $21 \leq h < 24$.

8. Exercice

Écrire une procédure *calculette* qui demande à l'utilisateur d'entrer :

- Un nombre x ;
- Un opérateur op ;
- Puis un nombre y ;

Et qui affiche le résultat du calcul : x op y.

Exemple:

```
>>> calculette()
Nombre : 5
Operateur : *
Nombre : 2
=
10
>>> calculette()
Nombre : 6
Operateur : /
Nombre : 3
=2
>>> calculette()

Nombre : 5
Operateur : +
Nombre : 6
=
11
>>> calculette()
Nombre : 5
Operateur : -
```

```
  Nombre : 6
  =
-1
```

TP 4 – Boucle « Pour »

1. Exercice

Implantez la procédure *dix* qui affiche les 10 premiers entiers.

Correction.

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

dix()
```

2. Exercice

Implantez la procédure *n_premiers_entiers*, qui prend en argument un entier n, et affiche les n premiers entiers.

3. Exercice

Implantez la procédure *n_premiers_carre*, qui prend en argument un entier n, et affiche les n premiers entiers au carré.

4. Exercice

Implantez la procédure *n_premiers_pairs*, qui prend en argument un entier n, et affiche les entiers pairs compris entre 0 et n.

5. Exercice

Implantez la procédure *n_premiers_impairs*, qui prend en argument un entier n, et affiche les entiers impairs compris entre 0 et n.

6. Exercice

Implantez la procédure *_10_fois_coucou* qui affiche 10 fois « coucou ».

7. Exercice

Implantez la procédure *n_fois_coucou*, qui prend en argument un entier n, et affiche n fois « coucou ».

8. Exercice

Implantez la fonction *somme_n_premiers_entiers*, qui prend en argument un entier n, et retourne la somme des n premiers entiers ente 0 et n-1.

9. Exercice

Implantez la fonction *somme_n_premiers_carres*, qui prend en argument un entier n, et retourne la somme des n premiers entiers au carré compris ente 0 et n-1.

10. Exercice

Implantez la fonction *somme_n_premiers_pairs*, qui prend en argument un entier n, et retourne la somme des n premiers entiers pairs compris ente 0 et n-1.

11. Exercice

Implantez la fonction *somme_n_premiers_impairs*, qui prend en argument un entier n, et retourne la somme des n premiers entiers impairs compris ente 0 et n-1.

12. Exercice

Implantez la fonction *somme_n_premiers_pairs_carre*, qui prend en argument un entier n, et retourne la somme des entiers pairs, compris entre 0 et n-1, au carré.

13. Exercice

Implémentez l'algorithme suivant :

```
FONCTION toto(n) :
  res = 0
  POUR i=0,..,n-1 :
    SI i est divisible par 3 :
      res = res + i
    FIN SI
  FIN POUR
  RETOURNER res
```

14. Exercice

Implémentez l'algorithme suivant :

```
FONCTION toto2(n, k) :
  res = 0
  POUR i=0,..,n-1 :
    SI i est divisible par k :
      res = res + i
    FIN SI
  FIN POUR
  RETOURNER res
```

15. Exercice

Donnez une nouvelle implantation de *toto* à partir de *toto2*.

TP 5 – Boucles « Tant que »

Ce TP est en partie une redite du TP précédent ; mais vous utiliserez cette fois-ci les boucles « Tant que ».

1. Exercice

Implantez la procédure *dix* qui affiche les 10 premiers entiers.

Correction.

```
def dix():
    i = 0
    while i < 10:
        print(i)
        i += 1

dix()
```

2. Exercice

Implantez la procédure *n_premiers_entiers*, qui prend en argument un entier *n*, et affiche les *n* premiers entiers.

3. Exercice

Implantez la procédure *n_premiers_carre*, qui prend en argument un entier *n*, et affiche les *n* premiers entiers au carré.

4. Exercice

Implantez la procédure *n_premiers_pairs*, qui prend en argument un entier *n*, et affiche les entiers pairs compris entre 0 et *n*.

5. Exercice

Implantez la procédure *n_premiers_impairs*, qui prend en argument un entier n, et affiche les entiers impairs compris entre 0 et n.

6. Exercice

Implantez la procédure *_10_fois_coucou* qui affiche 10 fois « coucou ».

7. Exercice

Implantez la procédure *n_fois_coucou*, qui prend en argument un entier *n*, et affiche *n* fois « coucou ».

8. Exercice

Implantez la fonction *somme_n_premiers_entiers*, qui prend en argument un entier *n*, et retourne la somme des n premiers entiers ente 0 et *n*-1.

9. Exercice

Implantez la fonction *somme_n_premiers_carres*, qui prend en argument un entier *n*, et retourne la somme des *n* premiers entiers au carré compris ente 0 et *n*-1.

10. Exercice

Implantez la fonction *somme_n_premiers_pairs*, qui prend en argument un entier *n*, et retourne la somme des *n* premiers entiers pairs compris ente 0 et *n*-1.

11. Exercice

Implantez la fonction *somme_n_premiers_impairs*, qui prend en argument un entier *n*, et retourne la somme des *n* premiers entiers impairs compris ente 0 et *n*-1.

12. Exercice

Implantez la fonction *somme_n_premiers_pairs_carre*, qui prend en argument un entier *n*, et retourne la somme des entiers pairs, compris entre 0 et *n*-1, au carré.

13. Exercice

Implémentez l'algorithme suivant :

```
FONCTION toto(n) :
  res = 0
  i = 0
  TANT QUE i<n :
    si i est divisible par 3 alors
      res = res + i
    fin si
    i = i+1
  FIN TANT QUE
  RETOURNER res
```

14. Exercice

Implémentez l'algorithme suivant :

```
FONCTION toto2(n, k) :
  res = 0
  i = 0
  TANT QUE i<n :
    SI i est divisible par k :
      res = res + i
    FIN SI
    i = i+1
  FIN TANT QUE
```

```
  RETOURNER res
```

15. Exercice

Donnez une nouvelle implantation de *toto* à partir de *toto2*.

16. Exercice

Implantez l'algorithme suivant :

```
FONCTION toto3() :
  res = 0
  i = 1
  TANT QUE i<=100 :
    res = res + i
    i = i+1
  FIN TANT QUE
  RETOURNER res
```

17. Exercice

Implantez l'algorithme suivant :

```
FONCTION toto4(a, b) :
  res = 0
  i = a
  TANT QUE i<=b :
    res = res + i
    i = i+1
    FIN TANT QUE
  RETOURNER res
```

18. Exercice

Donnez une nouvelle implantation de *toto3* à partir de *toto4*.

19. Exercice

Écrire un programme qui boucle tant que 'q' n'est pas entré au clavier. A chaque tour de boucle, on affichera ce qui est entré au clavier.

20. Exercice

Écrire un programme qui modélise le jeu ci-dessous.

Ce jeu se joue à deux joueurs.

- *Le premier joueur choisit un entier compris entre 1 et 100.*
- *Le second joueur cherche cet entier.*
- *Si l'entier qu'il propose est trop grand, le joueur 1 dit "trop grand !".*
- *Si l'entier qu'il propose est trop petit, le joueur 1 dit "trop petit !".*
- *Et si le joueur 2 trouve l'entier, alors le joueur 1 dit "trouvé !".*

Note : dans le cas de cet exercice, le joueur 1 correspond à la machine et le joueur 2 est humain.

TP 6 – Listes

1. Exercice

Écrire une procédure *afficher*, qui prend en argument une liste, et qui affiche ses éléments les uns au-dessous des autres.

Correction.

```
def afficher(lst):
    n = len(lst)
    for i in range(n):
        print(lst[i])

afficher(["aaa", "bb", "c"])
```

2. Exercice

Écrire une procédure *afficher_a*, qui prend en argument une liste *lst* ainsi qu'un entier *a* ; et affiche : lst[a] si *a* est correctement défini.

3. Exercice

Écrire une procédure *afficher_a_b*, qui prend en argument une liste *lst* ainsi que deux entiers *a* et *b* ; et qui affiche les éléments de *lst* d'indices compris enter *a* et *b* (si *a* et *b* sont correctement définis).

Exemple :

```
afficher_a_b([1,2,3,4,5,6], 2, 5)
```

affichera :

```
3,4,5 et 6.
```

4. Exercice

Écrire une fonction *concatener*, qui prend en argument une liste de chaînes de caractères *lst*, et affiche ses éléments en les concaténant à la manière de l'exemple ci-dessous :

```
>>> concatener(["Bonjour", "a", "tous", "!"])
Bonjour a tous !
```

5. Exercice

Écrire une fonction *appartient*, qui prend en argument une listes lst ainsi qu'un élément x ; et qui retourne :

- Vrai si x est dans lst ;
- Faux sinon.

6. Exercice

Écrire une fonction *non_appartient*, qui prend en argument une listes *lst* ainsi qu'un élément *x* ; et qui retourne :

- Vrai si *x* n'est pas dans *lst* ;
- Faux sinon.

7. Exercice

Écrire une fonction *appartient2*, qui prend en argument deux listes *lst1* et *lst2* ainsi qu'un élément *x* ; et qui retourne :

- Vrai si *x* est à la fois dans *lst1* et dans *lst2* ;
- Faux sinon.

8. Exercice

Écrire une fonction *appartient3*, qui prend en argument deux listes *lst1* et *lst2* ainsi qu'un élément *x* ; et qui retourne :

- Vrai si *x* est dans *lst1* et n'est pas dans *lst2* ;
- Faux sinon.

9. Exercice

Écrire une fonction *appartient4*, qui prend en argument trois listes *lst1*, *lst2* et *lst3* ainsi qu'un élément *x* ; et qui retourne :

- Vrai si *x* est dans *lst1* et également est dans « *lst2* ou *lst3* » ;
- Faux sinon.

TP 7 – Liste d'entiers

1. Exercice

Implantez la fonction *est_vide* qui retourne Vrai si la liste passée en argument est vide ; et Faux sinon.

Correction.

```
def est_vide(lst):
    return len(lst) == 0
```

ou

```
def est_vide(lst):
    return lst == []
```

2. Exercice

Implantez la fonction *maximum* qui retourne le maximum d'une liste d'entier passée en argument.

3. Exercice

Implantez la fonction *somme* qui retourne la somme des éléments de la liste d'entier passée en argument.

4. Exercice

Implantez la fonction *moyenne* qui retourne la moyenne des éléments de la liste d'entier passée en argument.

5. Exercice

Implantez la fonction *tous_egaux* qui retourne Vrai si tous les éléments de la liste passée en argument sont égaux ; et Faux sinon.

6. Exercice

Implantez la fonction *produit_scalaire* qui retourne le produit scalaire des deux listes d'entiers (de même taille) passées en argument.

Formellement, le produit scalaire est défini par :

$$(x_1,\dots,x_n).(y_1,\dots,y_n)=x_1y_1+\dots+x_ny_n.$$

7. Exercice

Implantez la fonction *egaux* qui retourne Vrai si les deux listes d'entiers (de même taille) passées en argument contiennent les mêmes éléments (dans le même ordre) ; et Faux sinon.

Si :

- lst1 = [x_1, ..., x_n] et
- lst2 = [y_1, ..., y_n] ;

Alors :

egaux(lst1, lst2) retournera Vrai si : x_i = y_i pour chaque i.

8. Exercice

Implantez la fonction *est_triee*, qui prend en argument une liste d'entier ; et qui retourne Vrai si la liste est triée ; et Faux sinon.

TP 8 – Mots

1. Exercice

Testez, dans l'IDLE Python, la suite de commandes suivantes :

```
a = "toto"
a
a == "toto"
b = "Toto"
b
b == "Toto"
a == b
a != b
a[0] = "T" # produit une erreur
for x in a:
print(x)
len(a)
a[0]
```

```
a[1]
a[2]
a[3]
a[4] # produit une erreur
list(a) == list(a)
list(a) == list(b)
list(a) != list(b)
"t" in "toto"
"T" in "toto"
"to" in "toto"
d = a + " et " + b
a in d
b in d
```

2. Exercice

Implantez la procédure *concatener*, qui prend en argument une liste de chaîne de caractères ; et qui retourne la concaténation des éléments de cette liste comme dans l'exemple suivant :

```
>>> concatener(["a", "aa", "aaa", "aaaa"])
a aa aaa aaaa
```

3. Exercice

Implantez une fonction, qui prend en argument un mot *m* ainsi qu'un caractère *c*, et qui retourne Vrai si *c* est présent dans *m*.

4. Exercice

Implantez une procédure qui affiche chaque lettre d'un mot passé en argument. On affichera chaque lettre sur des lignes différentes.

5. Exercice

Implantez une procédure qui prend en argument un mot *m* ainsi qu'un caractère *c*, et qui affiche le nombre d'occurrences de *c* dans *m* (i.e. le nombre de fois où apparaît la lettre *c* dans le mot *m*, par exemple le caractère 't' apparaît deux fois dans le mot "toto").

6. Exercice

Implantez une fonction *mot_present*, qui prend en argument un mot *m1* et un mot *m2* ; et qui retourne Vrai si *m1* est présent dans *m2*, et Faux sinon.

Exemple :

```
  >>> mot_present("toto", "Bonjour toto")
True
```

Dr. M. GUEDJ

TP 9 – Matrices

1. Exercice

Testez les commandes suivantes dans l'IDLE Python :

```
>>> from numpy import *
>>> zeros((5,5))
>>> zeros((2,5))
>>> ones((5,5))
>>> ones((2,5))
```

Dans ce qui suit, vous testerez vos fonctions et procédures en utilisant les fonctions NumPy : *zeros* et *ones*.

2. Exercice

Implémentez une procédure d'affichage de matrice. La procédure prendra en argument la matrice ainsi que les nombres de ses lignes et de ses colonnes.

3. Exercice

Implémentez la fonction qui retourne Vrai si la matrice passée en argument ne contient que des 1 ; et qui retourne Faux sinon.

4. Exercice

Implémentez la multiplication d'une matrice par un scalaire.

Exemple de multiplication par un scalaire :

$$2 \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

5. Exercice

Implémentez l'addition de deux matrices (les deux matrices seront considérées comme ayant les mêmes dimensions).

Exemple d’addition de deux matrices :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix} = \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

TP10 – Mini Projets

1. Exercice [Pierre-Feuille-Ciseau]

Implantez le jeu Pierre-Feuille-Ciseau. L'utilisateur joue contre la machine. Il utilise les touches ‘p’ pour « pierre », ‘f’ pour « feuille », ‘c’ pour « ciseau » et ‘q’ pour quitter le jeu (sinon le jeu boucle).

Exemple de partie :

```
>>>
p
pierre contre feuille
joueur 2 gagne !
p
pierre contre pierre
Ex aequo !
p
pierre contre ciseau
joueur 1 gagne !
f
feuille contre ciseau
joueur 2 gagne !
f
feuille contre feuille
Ex aequo !
f
feuille contre ciseau
joueur 2 gagne !
q
>>>
```

2. Exercice[Dessin de triangle]

Écrire un programme qui dessine, sur la console, un triangle comme suit :

```
#
##
###
####
#####
####
###
##
#
```

En demandant préalablement à l'utilisateur d'entrer la hauteur du triangle (ici la hauteur est de 5).

3. Exercice [Jeu des allumettes]

Réalisez (en mode console) « le jeu des allumettes », dont la description est donnée ci-dessous.

Ce jeu se joue a deux. Les joueurs sont devant un certain nombre d'allumettes. A chaque tour, il faut en enlever 1, 2 ou 3. Celui qui prend la dernière perd.

4. Exercice [Jeu de Toto]

Réalisez un programme, qui affiche sur la console un plateau, dans lequel Toto, symbolisé par la lettre 'T', se déplace. C'est un bon exercice de maîtrise des éléments de base de l'algorithmique (structures de donnés et de contrôle).

Le plateau sera exprimé par une matrice de caractères :

```
plateau = [[" " for j in range(colonne)] for i in range(ligne)]
```

1. Dans un premier temps, Toto se déplacera aléatoirement dans le plateau (dans les limites du plateau). On utilisera `from random import randint` et `randint(1,n)` qui renvoi un entier compris entre 0 et n.
2. Dans un second temps, recherchez un déplacement plus fluide de Toto.
3. Enfin, réalisez un déplacement « manuel » de Toto, i.e. via entrées de l'utilisateur au clavier.

```
---- Exercices facultatifs ----
```

5. Exercice [Bulls and Cows]

Réalisez (en mode console) le jeu « Bulls and Cows », dont la description (extraite de Wikipedia) est

donnée ci-dessous.

The numerical version of the game is usually played with 4 digits, but can also be played with 3 or any other number of digits.

On a sheet of paper, the players each write a 4-digit secret number. The digits must be all different. Then, in turn, the players try to guess their opponent's number who gives the number of matches. If the matching digits are in their right positions, they are « bulls », if in different positions, they are « cows ». Example:

- *Secret number: 4271*
- *Opponent's try: 1234*
- *Answer: 1 bull and 2 cows. (The bull is "2", the cows are "4" and "1").*

The first one to reveal the other's secret number wins the game.

Dans notre cas, on s’intéresse a un match « humain versus CPU ». L'humain seul droit trouver le code

du CPU.

6. Exercice [Tic-tac-toe]

Réalisez (en mode console) le jeu « Tic-tac-toe » (aussi appelé « morpion »), dont la description est donnée ci-dessous.

Le Tic-tac-toe se joue sur une grille carrée de 3x3 cases.

Deux joueurs s'affrontent. Ils doivent remplir chacun à leur tour une case de la grille avec le symbole qui leur est attribué : O ou X. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale.

1. Dans un premier temps, en mode deux joueurs « humains ».
2. Ensuite en mode joueur « humain » versus joueur CPU (jeu aléatoire pour le CPU).
3. Enfin, intéressez-vous a l'IA du CPU.