Introduction à Python

M1 - IA / GL

Ilyas Bambrik

2024

Table des matières

Introduction

L'objectif de ce court tutoriel c'est de se familiariser avec la syntaxe de base du langage python.

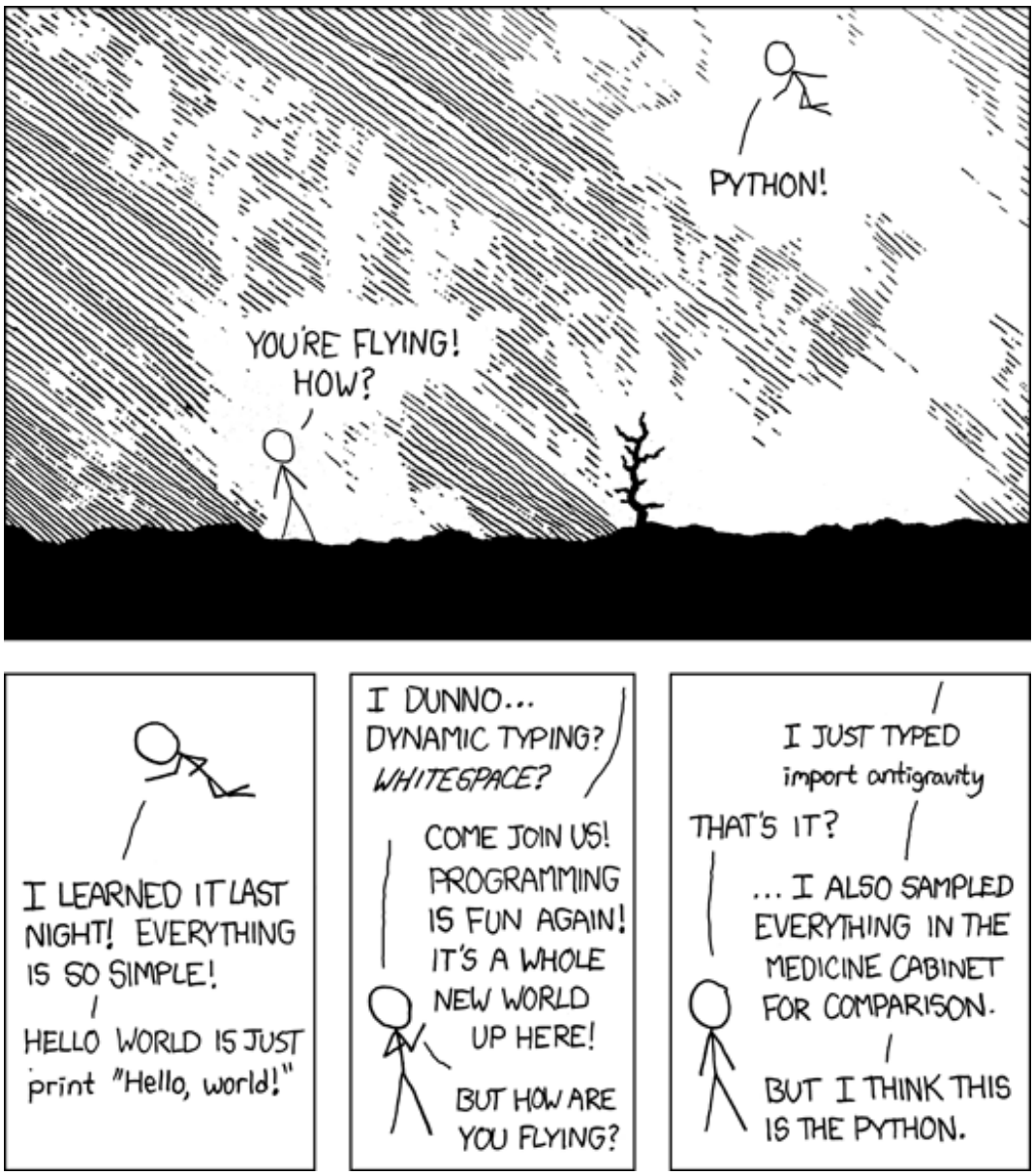
Python est un langage de scripte extrêmement simple syntaxiquement et possédant une bibliothèque riche (regex, manipulation des matrices, combinaison et permutation, programmation web coté serveur,etc). Comme d'autres langages de script (php et OTCL), python supporte la programmation orientée objet.

Pendant les dernières années, python a gagné en popularité due aux nombreux projets développés en python comme (par exemple OpenStack, Fsociety et MechanicalSoup).

Afin de vérifier que python est correctement installer, il suffit de vérifier que les fichiers sources python (*.py*) sont distingués par l'icône python.

Python présente plusieurs avantages :

- Riche bibliothèque ;
- Syntaxe simple à apprendre ;
- Prototypage d'algorithme simplifié ;
- Large communauté active dans le développement Web, Intelligence Artificielle, Data Science, Cyber Sécurité et teste de pénétration;


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Syntaxe Python

Contrairement aux langages *static-typed* comme C et Java, les types des variables sont *dynamiquement* inférés lors de l'affectation. Ceci facilite énormément la programmation ;

Listing 1 Simple programme python

```
1 print('x=')
2 x=int(input())
3 print('p=')
4 p=int(input())
5 print(x,' puissance ',p,' =',x**p)
```

- *print* est la fonction d'affichage sur écran comme *printf* et *System.out.print* (et *println*) ;
- *input* permet la lecture d'une ligne de chaîne de caractères comme *scanf* (voir aussi *gets* en C) et *Scanner.next*;
- Afin de calculer x à la puissance p, l'opérateur ** est utilisé ;

Exemple

Listing 2 Simple calculatrice en python

```
1 #debut de la procedure 'calculatrice'
2
3 def calculatrice(nbr1, nbr2,operation):
4     if (operation == 'A'):
5         return nbr1+nbr2
6     elif (operation == 'S'):
7         return nbr1-nbr2
8     elif (operation == 'M'):
9         return nbr1*nbr2
10    elif (operation == 'D'):
11        if ( nbr2==0):
12            print "Division par zero"
13            return
14        return nbr1/nbr2
15    print("Operation indefini")
16 #fin de la procedure 'calculatrice'
17
18 print("Entrez la valeur de Nbr1=")
19 nbr1=float(raw_input())
20
21 print("Entrez la valeur de Nbr2=")
22 nbr2=float(raw_input())
23
24 print("\nEntrez le type d'operation\nA:\tAddition\nS:\tSoustraction\nM:
   \tMultiplication\nD:\tDivision\n")
25 op=input()
26
```

```
print("Resultat= ", calculatrice(nbr1, nbr2,op))

```

- Il n'existe pas de commentaire multi-lignes en python et le caractère # marque le début d'un commentaire mono-ligne (comme en TCL et Bash/sh);
- La déclaration d'une procédure (ou fonction) commence par le mot clé *"def"*. Comme dans le corps du programme, les types des paramètres ainsi que le type de retour d'une procédure ne sont pas déclarés explicitement (il sont dynamiquement inférés selon le traitement effectué sur les arguments). Par exemple :
  def NomProcedure (argment1, argument2,argument3) *:*
- La clause *"switch"* n'existe pas dans la syntaxe python. Cependant, cette clause est remplacée par *"if-elif-else"*, pour le teste de multiple cas. En outre, une condition est formulée de la manière suivante :
  if (*condition*) *:*

Par exemple, le programme présenté dans *Listing 3* illustre comment les conditions sont formulées

Listing 3 Les conditions et opérateurs logiques

```
def Mention(note):
    if(note <7):
        return "F"
    elif (note >=7 and note<10):
        return "D"
    elif (note >=10 and note<13):
        return "C"
    elif (note>=13 and note<15):
        return "B"

    return "A"


Note=int(input())
print(Mention(Note))
```

- Le type "char" n'existe pas en python. De plus les chênes de caractères peuvent être enfermer entre *doubl quottes* (") ou *single quottes* (') comme en *php* et *Javascript*. Une chaîne de caractères multi-lignes est déclarée avec """ (ou ''') au début et à la fin. Celle-ci peut aussi être utilisée à la place d'un commentaire multi-lignes ;
- Les opérateurs logiques binaires en python sont *and* et *or* à la place *&&* et || dans les langages C-type. L'opérateur *not* remplace l'opérateur de négation (*!*) . En outre, les valeurs logiques en python sont *True* et *False* (sensible à la case) ;

Remarque : Différences entre syntaxe Python 2 et Python 3

Afin de tester les exemples proposés dans ce cours, il faut installer Pytnon 3. En outre, il est important de signaler que la syntaxe de Python 2 est différente de celle de Python 3. Par exemple, la fonction *"raw_input()"* (présente dans python 2.x) est remplacé en python 3.x par la fonction *"input()"*.

1. Structure de bloques par tabulations

La principale innovation syntaxique en python est l'utilisation de tabulations. La tabulation dans un programme python indique le niveau du bloque et l'imbrication des instructions. Toute violation de cette règle syntaxique est notifié par un message d'erreur lors de l'interprétation. *Selon le nombre de tabulations séparant l'instruction du début de la ligne, l'interpréteur python infère le bloque où l'instruction est située*. L'imposition de cette convention syntaxique rend la lecture du code source plus facile. Par exemple, voir *Listing 4* :

Listing 4 Structure de bloque en python

```
1 #Bloque~1
2 instruction1
3 instruction2
4 if (Condition1):
5     #Bloque~2
6     instruction3
7     instruction4
8     if (Condition1):
9         #Bloque~3
10        instruction5
11        instruction6
12    instruction7
13 else :
14    #Bloque~4
15    instruction8
16    instruction9
17 instruction10
18 instruction11
```

2. Casting de type en python

Pour effectuer le casting en python, la variable ou la valeur qu'on souhaite convertir est passée en paramètre au type voulu. Le casting en python se fait comme suite :

Variable1=*TypeVariable1*(Varible2D_UnAutreType)

ou bien

Variable1=*TypeVariable1*(ValeurD_UnAutreType)

Cette écriture est équivalente en C / Java à l'instruction suivante :

Variable1= (*TypeVariable1*) Varible2D_UnAutreType ;

Variable1= (*TypeVariable1*) ValeurD_UnAutreType ;

Dans le programme présenté dans *Listing 4* (ligne 28) la valeur lue du clavier (input()), qui est une chaîne de caractères, est convertie en type *float* avant d'être affectée à la variable *nbr1*.

Les boucles en python

II

L'usage de la clause *for* en python est quelque peut atypique. Au lieu de la déclaration traditionnelle, la boucle en python est déclarée comme une itération d'une variable sur un tableau (comme en Bash). Pour l'exemple présenté dans *Listing 5* (*ligne 4*) la variable "*element*" itère sur les éléments du tableau *tab*. Autrement dit, la variable "*element*" prend la valeur d'un élément de *tab* à chaque itération.

Autres particularités des tableaux en python :

- La déclaration d'un tableau se fait simplement par l'affectation à un tableau vide (*ligne 19*).
- Les éléments sont dynamiquement ajoutés par la méthode *append* (*ligne 22*).

Le programme présenté dans *Listing 5* comporte une procédure qui calcule la somme des valeurs des éléments d'un tableau *(ligne 2)* :

Listing 5 La boucle *for*

```
1 import sys
2 def Somme (tab):
3     valeur_somme=0
4     for element in tab:
5         valeur_somme=valeur_somme+element
6     return valeur_somme
7
8 # Programme principale
9
10 print("Entrez le nombre d'elements du tableau")
11 Nombre_D_Elements=int(input())
12
13 TableauD_Entier=[]
14 for i in range(Nombre_D_Elements):
15     print "TableauD_Entier[" + i + "]="
16     TableauD_Entier.append(int(input()))
17
18 print( "Somme = ",Somme(TableauD_Entier) )
```

La fonction Somme suivante est la même fonction présentée dans *Listing 5*, écrite avec la boucle *while (ligne 4):*

Listing 6 La boucle *while*

```
1 def Somme (tab):
2     valeur_somme=0
3     i=0
4     while(i<len(tab)):
5         valeur_somme=valeur_somme+tab[i]
6         i=i+1
7     return valeur_somme
```

L'invocation de la fonction *range(Nombre_D_Element)* (ligne 13- *Listing 5*) retourne un objet contenant les chiffres allant de *0* jusqu'à *Nombre_D_Element - 1*. Pour voir ce résultat, il suffit d'exécuter le programme suivant (*list* représente le type tableau qui permet de convertir l'objet retourné par *range* ne tableau):

```
1 print(list(range(10)))
```

L'affichage de l'exécution précédente sera comme suite :

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

La fonction range() peut avoir de 1 jusqu'à 3 paramètres :

range(N1,N2) génère un tableau de nombre allant de *N1* jusqu'à *N2-1*

range(N1,N2,step) génère un tableau de nombre allant de *N1* jusqu'à *N2-1* un pas de valeur *step. Step peut être négatif*

Listing 7 La fonction range()

```
1 print("Tableau de 10 a 20",list(range(10,20)))
2
3 print( "Tableau de 50 a 100 avec un pas de 15",list(range(50,100,15)))
```

```
Tableau de 10 a 20 [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Tableau de 50 a 100 avec un pas de 15 [50, 65, 80, 95]
```

Les tableaux

Dans le programme suivant, la taille du tableau (*TailleTableau*, ligne 1) est lue à partir du clavier. Ensuite, les éléments lus à partir du clavier sont ajoutés au tableau (*Tableau.append(Element)*, *ligne 5*). En outre, la fonction *len()* prend en paramètre un tableau et retourne la taille de celui-ci (*ligne 8*).

Listing 8 Manipulation des tableaux

```
1 TailleTableau=int(input())
2 Tableau=[]
3 for i in range(TailleTableau):
4     print "Tableau[",i,"]="
5     Tableau.append(int(input()))
6 print("Tableau=",Tableau)
```

```
Taille Tableau=
4
Tableau[ 0 ]=
12
Tableau[ 1 ]=
6
Tableau[ 2 ]=
75
Tableau[ 3 ]=
-2
Tableau= [12, 6, 75, -2]
Taille tableau= 4
```

- Comme dans la fonction *range()* présentée précédemment, l'indice de l'élément tableau supporte jusqu'à 3 paramètres (testez l'exemple présenté dans le listing suivant) :
- Tableau [i] : désigne le i^{em} élément du tableau *si (i>=0)*. Si *i<0* , *Tableau [i] désigne l'élément à la position N-i^{em} avec N définie comme la taille du tableau*.
- Tableau [N1 : N2] : désigne la séquence d'éléments du tableau allant de l'indice *N1* jusqu'à *N2-1*.
- Tableau [N1 : N2 :step] : désigne la séquence d'éléments du tableau allant de l'indice *N1* jusqu'à *N2* avec un pas = *step* (*l'élément à l'indice N2 est exclu de la séquence*). Par défaut, Tableau[: :] est équivalente à *Tableau[0 :len(Tableau) :1]* (équivalant au tableau entier).

La fonction *split*() (*équivalente à la fonction split implémentée dans la classe String en Java*) découpe une chaîne de caractères en morceaux selon le délimiteur (le caractère espace dans l'exemple suivant, *ligne 1*) et renvoie un tableau des éléments séparés par le séparateur:

Listing 9 Les slicers

```
1 Tableau=input().split(" ")
2 print( "Tableau = ", Tableau)
3 print( "Tableau [-4] = ", Tableau[-4])
```

```
print( "Tableau [3 : 5] = ", Tableau[3:5])
print( "Tableau [5:2:-1]= ", Tableau[5:2:-1])
print( "Tableau [::]= ", Tableau[::])
```

```
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
Tableau =  ['10', '20', '30', '40', '50', '60', '70', '80', '90', '100', '110', '120', '130', '140', '150']
Tableau [-4] =  120
Tableau [3 : 5] =  ['40', '50']
Tableau [5:2:-1]=  ['60', '50', '40']
Tableau [::]=  ['10', '20', '30', '40', '50', '60', '70', '80', '90', '100', '110', '120', '130', '140', '150']
```

Conseil

La concaténation des listes se fait par une simple opération + :

`[1,2,3,4,5] + [6,7,8,9] = [1,2,3,4,5,6,7,8,9]`

1. Les dictionnaires

Un dictionnaire est une collection qui supporte comme indice une chaîne de caractères (*comme les tableaux associatives en php*) ainsi que d'autres types de valeurs immuables et supportant le hachage :

```
UnDictionnaireVide={} # initialisation d'un dictionnaire vide
UnDictionnaire ={ "Nom":"Mohammed","Specialite":"Informatique","Age":25}
print(UnDictionnaire["Specialite"])
print(UnDictionnaire.has_key("Adresse"))
print(UnDictionnaire.keys())
print(UnDictionnaire.values())
```

L'exécution du programme précédant affiche le résultat suivant :

```
Informatique
False
['Nom', 'Age', 'Specialite']
['Mohammed', 25, 'Informatique']
```

Les classes, objets en python

IV

Python supporte de même les concepts de la POO. Le programme présenté dans *Listing III.6* présente une classe TypeJoueur :

Listing 10 Les classes et objets en python

```
1 class TypeJoueur:
2     def __init__(self):
3         self.PointsDeVie=100
4         self.Score=0
5         self.NomJoueur=''
6 
7     def lireNomJoueur(self):
8         self.NomJoueur = input()
9 
10    def mettreAJourScore(self,scoreRecompense):
11        self.Score=self.Score+scoreRecompense
12 
13    def afficherStatisticJoueur(self):
14        print( "NomJoueur=",self.NomJoueur,"\nPoints De Vie=",self.PointsDeVie,
  "\nScore=",self.Score)
15 
16 Med=TypeJoueur()
17 
18 Med.lireNomJoueur()
19 Med.mettreAJourScore(10)
20 Med.afficherStatisticJoueur()
```

- La méthode *def __init__(self):* représente le constructeur de la classe en python. En outre, les attributs de la classe sont déclarés à l'intérieur du constructeur *__init__*.
- Le mot clé *self*, passé en argument dans tous les déclarations de méthode de la classe *TypeJoueur*, représente l'objet qui a invoqué la méthode ou le constructeur. Cependant, lors de l'appel d'une méthode, cet objet *self* est automatiquement inféré. Par exemple :
  Med=TypeJoueur()# est équivalente à Med=TypeJoueur(*Med*). Donc l'objet courant (self) prendra la référence de l'objet qui a invoqué la méthode
  Med.mettreAJourScore(10) # de même cette instruction est équivalente à Med.mettreAJourScore(*Med*, 10)
- L'appel du constructeur (*ligne 16*) se fait sans opérateur *new* contrairement à la syntaxe Java.

Exemple : Constructeur paramétré

Si on souhaite ajouter un constructeur avec des arguments *pointsvie*, *score* et *nom* ,il suffit d'ajouter le constructeur suivant :

Listing 11 Déclaration d'un constructeur paramétré en python

```
1     def __init__(self,pointsvie,score,nom):
2         self.PointsDeVie=pointsvie
3         self.Score=score
4         self.NomJoueur=nom
```

Il est possible aussi de définir l'implémentation des opérateur comme +, -, *, [] , (). Par exemple, la classe *list* (tableau) définie l'opérateur + pour concaténer une liste avec une autre et * pour répéter une liste un certain nombre de fois. Voir l'exemple suivant :

```
1 A=[10,20,30,40]
2 B=[100,200,300,400]
3 C=A+B
4 print(C) # [10, 20, 30, 40, 100, 200, 300, 400]
5 C=A*3
6 print(C) # [10, 20, 30, 40, 10, 20, 30, 40, 10, 20, 30, 40]
```

La surcharge d'opérateur peut être défie dans une classe avec la méthode correspondante à l'opérateur spécifique. Par exemple, + est définie par la méthode *__add__*, * par *__mul__*, / par *__div__*. L'exemple suivant définie l'opérateur d'addition (__add__ *ligne 15*) pour modifier le score du joueur et l'opérateur de multiplication (__mul__ *ligne 18*) pour multiplier le score par un facteur. Les deux méthodes précédentes retourne l'objet après la modification du score (*ligner 17 et 20*). La surcharge des deux opérateurs est testée dans *ligne 26 et 28*.

```
1 class TypeJoueur:
2     def __init__(self):
3         self.PointsDeVie=100
4         self.Score=0
5         self.NomJoueur=''
6 
7     def lireNomJoueur(self):
8         self.NomJoueur = input()
9 
10    def mettreAJourScore(self,scoreRecompense):
11        self.Score=self.Score+scoreRecompense
12 
13    def afficherStatisticJoueur(self):
14        print( "NomJoueur=",self.NomJoueur,"\nPoints De Vie=",self.PointsDeVie,
  "\nScore=",self.Score)
15    def __add__(self,scoreRecompense):
16        self.Score=self.Score+scoreRecompense
17        return self
18    def __mul__(self,multiplicateurScore):
19        self.Score=self.Score*multiplicateurScore
20        return self
21 
22 Med=TypeJoueur()
23 
24 Med.lireNomJoueur()
25 Med.mettreAJourScore(10)
```

```
26 Med+=20
27 Med.afficherStatisticJoueur()
28 Med*=5
29 Med.afficherStatisticJoueur()
```

Tuple et unpacking

V

Python offre un autre type similaire au tableau appelé tuple. Les tuples supportent la plus part des opérations sur les tableaux sauf qu'ils sont immuables (une fois que le tuple est créé, on ne peux pas changer son contenu).

```
>>> Etudiant=("Mohammed",18,"60kg","L1")
>>> print Etudiant
('Mohammed', 18, '60kg', 'L1')
>>> print type(Etudiant)
<type 'tuple'>
>>> print Etudiant[1]
18
>>> for attribut in Etudiant:
        print attribut,

Mohammed 18 60kg L1
>>> Etudiant[1]=20

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    Etudiant[1]=20
TypeError: 'tuple' object does not support item assignment
```

Les tableaux et les tuples sont souvent utilisés pour déballer (unpack) plusieurs valeurs. Supposant que ont possède un tuple T contenant le nom, l'age et le niveau d'un étudiant et on souhaite attribuer chaque valeur à une variable.

```
1 T=("Mohammed",18,"L3")
2 nom,age,niveau=T
3 print(nom)
4 print(age)
5 print(niveau)
```

- Le tuple précédant peut être écrit sans les parenthèses :
  T="Mohammed",18,"L3"
- La même syntaxe s'applique avec les tableaux. Ceci est utile pour retourner plusieurs valeurs au même temps par une fonction.

Syntaxe

L'instruction "nom,age,niveau=T" est équivalente à :

```
nom=T[0]
age=T[1]
niveau=T[2]
```

Cette syntaxe peut être utilisée afin de permuter les valeurs des variables :

```
a=1
```

b=2

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

ou bien

```
a,b=[b,a]
```

Permet de placer la valeur de a dans b et la valeur de b dans a.

Paramètres de fonction par défaut

VI

Dans une fonction il est possible de déclarer un ou plusieurs paramètres par défaut.

```
1 def multiplier(nombre,facteur=2):
2   return nombre*facteur
3 print( multiplier(3,5)) # resultat =15
4 print( multiplier(7)) # resultat = 14
```

- Dans l'exemple précédant, la fonction multiplier possède *un paramètre par défaut facteur avec une valeur 2*. Ceci dit que si la valeur du paramètre n'est pas précisée, la valeur de ce dernier est égale à 2 (voir l'appel de multiplier avec un seul paramètre).
- Il est possible de déclarer plusieurs paramètres par défaut. La seule restriction est que les paramètres par défaut ne sont pas suivies par un paramètre obligatoire :
  L'écriture suivante est correcte :
  def fontion(*argobligatoire1,argobligatoire2,parametrepardefaut1=3,parametrepardefaut2=2, parametrepardefaut3=1*) ;
  Par contre l'écriture suivante est incorrecte car *argobligatoire2* vient après parametrepardefaut3 ;
  def fontion(*argobligatoire1,parametrepardefaut1=3,parametrepardefaut2=2,parametrepardefaut3=1, argobligatoire2*)

Remarque : Variable global

A l'intérieur d'une fonction, pour avoir accès à une variable global, il est nécessaire d'utiliser le mot clé *global* suivit par le nom de la variable :

```
1 variableGlobal=4
2 def impromerlavaleurdeVG():
3   global variableGlobal
4   print( variableGlobal)
5 print( impromerlavaleurdeVG())
```

Fonction lambda

VII

Souvent, il est nécessaire de déclarer des fonctions qui assurent un traitement limités ou calculer une formule prédéfinie.

Par exemple supposant qu'ont souhaite écrire une fonction permettant de convertir la température en degré Celsius vers Fahrenheit.

temp_Fahr=(temp_celsius × 9/5) + 32

```
1 def convertirCelsiusToFahr(temp_celsius):
2   return (temp_celsius * 9./5) + 32
```

A ce niveau, il faudra noter qu'une fonction peut être affectée à une variable. Une fonction déclarée en python n'est qu'un objet de type fonction :

- Dans l'exemple suivant on affecte à la variable *var1* la fonction *convertirCelsiusToFahr*;
- Par la suite, dans *ligne 3* avec la fonction type il est possible de voire que le type de la variable *var1* est "<type 'function'>" ;
- Il est même possible de faire appel à la procédure stockée dans *var1* avec *var1(val)* (*Ligne 5*) ;
- Ainsi les fonctions sont appelées *first class citizens* car il sont traitées comme des variables / objets et peuvent être utilisées comme des arguments pour d'autre fonctions.

```
1 def convertirCelsiusToFahr(temp_celsius):
2   return (temp_celsius * 9./5) + 32
3 var1=convertirCelsiusToFahr
4 print (type(var1)) # imprime <type 'function'>
5 print( var1(3.))
```

Comme d'autres langage évolués, il existe une façon plus concise afin de déclarer une telle opération appelée *expression lambda*:

```
1 fun=lambda temp_celsius:(temp_celsius * 9./5) + 32
2 print( fun(3.))
```

- L'expression lambda précédante déclare une fonction anonyme avec un seul paramètre *temp_celsius*, et retourne la valeur de la formule (temp_celsius * 9./5) + 32.
- La fonction déclarée est attribuée à la variable *fun*. Par la suite, afin d'invoquer cette fonction il suffit d'utiliser la variable *fun* (*Ligne 2*).
- Dans cette déclaration le mot clé *return* n'est pas utilisé alors que la fonction retourne la valeur de la formule. Ceci est appelé *implicite return*.

Fonction d'ordre supérieur (higher order function)

VIII

Une *fonction d'ordre supérieur* est une fonction qui a un paramètre une/plusieurs autre(s) fonction(s) ou retourne une fonction. Il existe plusieurs fonctions d'ordre supérieurs prédéfinies qui sont très utiles.

Exemple : map

Supposant qu'ont souhaite lire une liste d'entier, séparés par un espace, à partir du clavier. Ce travail peut être accompli de la manière suivante :

```
1 def convertirEnListEntier(L):
2         resultat=[]
3         for valeur in L:
4                 resultat.append(int(valeur))
5         return resultat
6 print( convertirEnListEntier(raw_input().split(" ")) )
```

- Le programme précédant prend chaque valeur de la liste, la convertie vers le type *int* en faisant appel à *int (valeur)* et ajoute le résultat au tableau *resultat*.
- Il existe une fonction qui permet d'appliquer une fonction à chaque élément de la liste et de retourner la liste contenant le résultat sans changer la liste originale. Cette fonction est appelée *map* :

```
1 resultat = list(map(int,raw_input().split(" ")))
2 print( resultat)
```

- La fonction *map*, prend une fonction en paramètre (dans cet exemple *int*) et une collection / tableau, applique cette fonction à chaque élément et ajoute le résultat au tableau sortie.
- L'avantage de ceci est que si on souhaite changer le type de conversion ou la fonction appliquée sur les éléments de la liste,il suffit de changer la fonction argument de *map*. Par exemple supposant que on souhaite avoir les valeurs de la liste saisie du clavier comme des doubles (*float*). La seule chose qui a changée est la fonction passée en paramètre (*float au lieu de int*).

```
1 resultat = list(map(float,raw_input().split(" ")))
2 print( resultat)
```

Exemple : filter

Une autre fonction similaire à *map*, est la fonction *filter*. Supposant qu'on possède une liste des notes et on souhaite retenir seulement les notes supérieures à 14. La fonction suivante fait exactement ça :

```
1 def notesSuppA14(L):
```

```
        resultat=[]
        for valeur in L:
                if valeur>14:
                        resultat.append(int(valeur))
        return resultat
Liste=[13, 15, 16, 10, 9, 18, 11, 10]
print( notesSuppA14(Liste))
```

Par contre, avec *filter,* il est plus facile d'accomplir ce traitement :

```
def supp14(note):
        return note>14
print(list(filter(supp14,Liste)))
```

- Il suffit de définir une fonction qui va retourner une valeur booléenne (True ou False) et la mettre comme paramètre pour filter. *Appelons cette fonction comme la fonction d'évaluation.*
- La fonction d'évaluation va être appliquée sur chaque élément de la liste. L'élément est ajouté à la liste résultat si l'évaluation retourne *True.*
- Il est possible d'écrire la fonction précédente d'une manière plus concise avec une expression lambda :

```
Liste=[13, 15, 16, 10, 9, 18, 11, 10]
print filter(lambda valeur:valeur>14,Liste)
```

TP 0 Exercice de programmation Python

IX

1. Exercice : Résumé Statistique

Exercice sur les tableaux et boucles

Complétez votre réponse après chaque question. Ne copiez pas le code à partir du pdf, celui-ci est téléchargeable.

```
Notes=[15, 13, 5, 11, 5, 14, 9, 6, 14, 1,
       8, 8, 11, 3, 12, 11, 2, 17, 14, 1, 6, 12, 13, 14, 1]
# Question 1 convertir les elements du tableau Notes du type en entier (int) au type float.
# et imprimer le resultat sur ecran
""" le resultat attendu = [15.0, 13.0, 5.0, 11.0, 5.0, 14.0, 9.0, 6.0, 14.0, 1.0, 8.0, 8.0, 11.0,
3.0, 12.0, 11.0, 2.0, 17.0, 14.0, 1.0, 6.0, 12.0, 13.0, 14.0, 1.0]"""



# Question 2 complete le programme pour calculer et imprimer la moyenne sur l'ecran
# Indication : utiliser le tableau apres conversion des element en float
# le resultat attendu = 9.04



# Question 3 calculez la variance type de Notes et imprimer le resultat sur ecran
# le resultat attendu = 23.6384



# Question 4 calculez la valeur la plus frequente dans le tableau et imprimez le resultat sur ecran
# le resultat attendu = 14.0 ou 14 (se repete 4 fois)


# Question 5 triez le tableau Notes sans utiliser les methodes predefinies ( .sort() et sorted ) dans l'ordre ascendant
"""le resultat attendu
[1.0, 1.0, 1.0, 2.0, 3.0, 5.0, 5.0, 6.0, 6.0, 8.0, 8.0, 9.0, 11.0,
11.0, 11.0, 12.0, 12.0, 13.0, 13.0, 14.0, 14.0, 14.0, 14.0, 15.0, 17.0]
ou bien
[1, 1, 1, 2, 3, 5, 5, 6, 6, 8, 8, 9, 11,
11, 11, 12, 12, 13, 13, 14, 14, 14, 14, 15, 17]
"""

```

```
# Qustion 6 creez une liste contenant seulement les valeurs uniques de Notes:
# Indication : utilisez le tableau apres le trie ou l'operateur 'in' pour tester
# l'appartenance
# resultat attendu =[1, 2, 3, 5, 6, 8, 9, 11, 12, 13, 14, 15, 17]


```

Question 1

Convertir les éléments du tableau Notes du type en entier (int) au type float et imprimer le résultat sur écran.

Indice :

Le résultat attendu = [15.0, 13.0, 5.0, 11.0, 5.0, 14.0, 9.0, 6.0, 14.0, 1.0, 8.0, 8.0, 11.0,

3.0, 12.0, 11.0, 2.0, 17.0, 14.0, 1.0, 6.0, 12.0, 13.0, 14.0, 1.0]

Question 2

Complété le programme pour calculer et imprimer la moyenne sur l'écran.

Indice :

Indication : utiliser le tableau après conversion des élément en float

Le résultat attendu = 9.04

Question 3

Calculez la variance de Notes et imprimer le résultat sur écran. (E(X) n'est que la moyenne que vous venez de calculer dans la question précédente).

$$\mathrm{Var(X)}=\frac{\sum_{i=1}^{n}(X_i-E(X))^2}{n} = E(\,((X-E(X))^2\,)$$

Indice :

Le résultat attendu = 23.6384

Question 4

Calculer la valeur la plus fréquente dans le tableau et imprimez le résultat sur écran.

Indice :

Le résultat attendu = 14.0 ou 14 (se répète 4 fois)

Question 5

Triez le tableau Notes sans utiliser les méthodes prédéfinies (.sort() et sorted) dans l'ordre ascendant.

Indice :

Le résultat attendu

[1.0, 1.0, 1.0, 2.0, 3.0, 5.0, 5.0, 6.0, 6.0, 8.0, 8.0, 9.0, 11.0,

11.0, 11.0, 12.0, 12.0, 13.0, 13.0, 14.0, 14.0, 14.0, 14.0, 15.0, 17.0]

ou bien

[1, 1, 1, 2, 3, 5, 5, 6, 6, 8, 8, 9, 11,

11, 11, 12, 12, 13, 13, 14, 14, 14, 14, 15, 17]

Question 6

Créez une liste contenant seulement les valeurs uniques de Notes.

Indice :

Indication : utilisez le tableau après le trie ou l'opérateur 'in' pour tester l'appartenance.

Résultat attendu =[1, 2, 3, 5, 6, 8, 9, 11, 12, 13, 14, 15, 17]

2. Exercice : Server Load Balancing

Dans ce défi, on vous donne N serveurs. Chaque serveur dispose d'un courant de charge Li de tâches en cours d'exécution. Ensuite un batch de k tâches arrive et doit être distribué sur les N serveurs. Votre travail consiste à concevoir un programme qui distribuera les travaux k entrants sur les serveurs de sorte que la différence entre le nombre de travaux sur le serveur avec la charge la plus élevée et celui avec la charge la plus faible soit minimale. Appelons cette métrique Déséquilibre minimal.

Par exemple si nous avons N=4 et les charges initiales comme suit [5,0,2,1]. Pour k=3 tâches arrivantes, la distribution des travaux pour obtenir ce qui suit [5,2,2,2] permet d'obtenir le déséquilibre minimal. Le résultat ici est 3 (5-2)

Ce défi était l'un des problèmes d'Amazon Last Mile 22.

Input :

Ligne 1 : N Un entier indiquant le nombre de serveurs

Ligne 2 : k Un entier indiquant le nombre de travaux à planifier

Ligne 3 : Une ligne contenant des entiers Li où chaque entier indique la charge actuelle du serveur

Output :

Un entier indiquant le déséquilibre minimum réalisable après la planification des k tâches.

Contraintes :

1<=N<10000

0<=Li<=10000

0<=k<=100000000

Exemple :

6

5

5 8 7 1 4 3

Sortie :

4

3. Exercice : Surface Maximale

Bob joue à LEGO. Il a N petites briques. Chaque brique est un cube 1x1x1.

En utilisant TOUTES les briques, Bob construit un gros "bloc". Ce bloc doit être un cuboïde simple (c'est-à-dire sans espace vide à l'intérieur). Bob s'intéresse à la surface externe du bloc (somme des surfaces des 6 faces) qu'il a construit.

Quelles sont les surfaces minimales et maximales possibles que Bob peut construire avec les N briques?

Input :

Un entier N, le nombre de petites briques

Output :

Une ligne, avec deux nombres entiers séparés par un espace.

Le premier entier est la surface minimale.

Le deuxième entier est la surface maximale.

Contraintes :

1 <= N <= 1 500 000

Exemple 1:

1

Sortie 1:

6 6

Exemple 2:

9

Sortie 2 :

30 38

Exemple 3 :

144

Sortie 2 :

168 578

4. Exercice : Maximiser l'entier

On vous donne un entier N. Votre but est de créer le plus grand entier possible.

Pour ce faire, vous pouvez échanger deux chiffres adjacents tant qu'ils sont de parité différente. Par exemple, vous pouvez échanger un chiffre impair et un chiffre pair, mais pas deux chiffres pairs. Vous pouvez effectuer autant d'échanges que vous le souhaitez.

Exemple :

Input :

Un entier N.

Output :

L'entier maximisé.

Contraintes :

N est composé d'au plus 900 chiffres.

Exemple :

325

Sortie :

352