Cours d'algorithmique et de programmation pour la classe de seconde

F.Gaudon

11 juillet 2019

1 Distinction entre algorithmique et programmation

Définition :

> Un *algorithme* est une succession *d'instructions* (aussi appelés *commandes*) permettant la résolution d'un problème donné.

Remarque :
Le terme « algorithme » vient du nom du mathématicien arabe du IX^{e} siècle *Al Khwarizmi* qui écrivit une des premières méthodes systématiques de résolution de certaines équations.

Exemple :

```
Stocker dans b le carré de a ;
afficher b.
```

L'algorithme précédent stocke dans « une boîte » nommée b le carré d'un nombre nommé a et affiche la valeur contenue dans b.

Définition :

> Un *Langage de programmation* est une ensemble d'instructions et de règles syntaxiques compréhensibles par un ordinateur et permettant de créer des algorithmes. Un *programme* est la traduction d'un algorithme dans le langage de programmation utilisé.

Exemples :
BASIC, PASCAL, C++, assembleur sont des langages de programmation.
En particulier, les calculatrices Texas Instrument et Casio mettent à disposition un langage de programmation provenant du langage BASIC mais avec des instructions plus limités.
Dans ce cours, nous utiliserons le langage de programmation *python*.
C'est un langage qui présentent les intérêts suivants

- C'est un langage *libre*, c'est à dire que n'importe qui ayant les compétences techniques nécessaires peut l'adapter à ses besoins ou contribuer à le perfectionner ;
- il est *multi-plateformes*, c'est à dire qu'un programme écrit en python fonctionnera aussi de la même manière sur des ordinateurs sous Windows, Linux, MAC OS, Android ou tout autre système d'exploitation ;
- il fait partie des langages les plus utilisés dans le monde universitaire et dans le monde professionnel ;
- il est réputé pour sa simplicité de programmation.

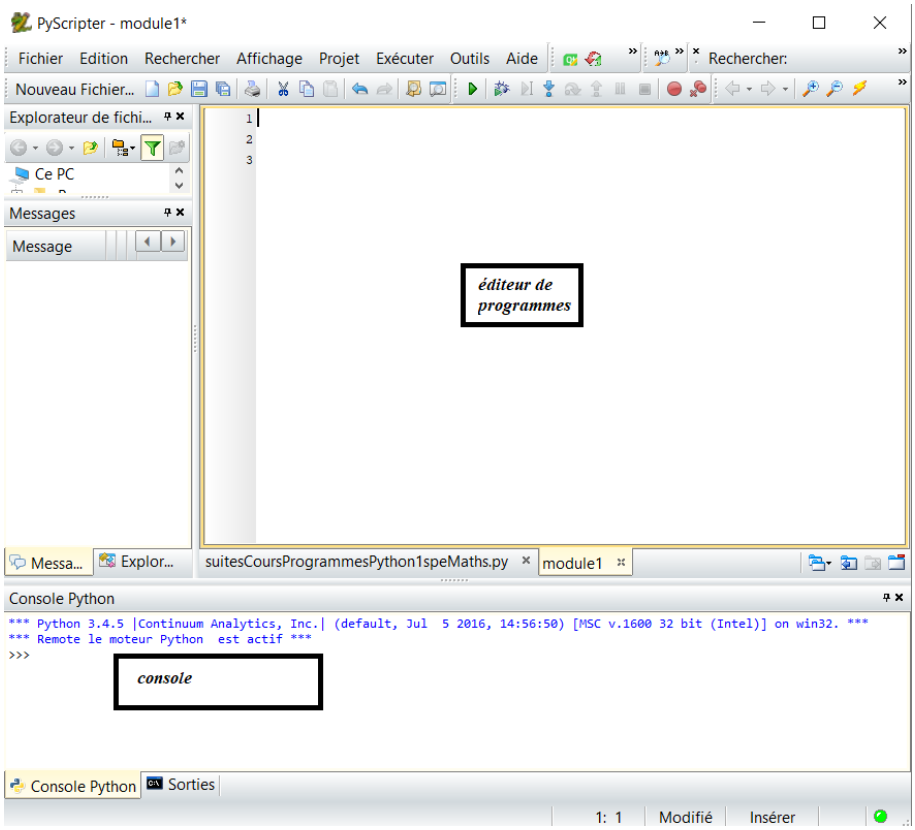
- Les calculatrices récentes des marques les plus courantes (Texas Instrument, Casio et Numworks) permettent de programmer en python.

Exemple :
Le programme suivant, que nous apprendrons à comprendre plus tard, est une écriture ou *implémentation* en python de l'algorithme de calcul de carré précédent :

```
b=a*a
print(b)
```

Programmation en python en pratique :

- L'édition d'un programme se fait dans un logiciel d'édition de texte (notepad, wordpad, gedit,...), ou dans un logiciel comprenant à la fois un *éditeur de texte* et un *interpréteur python*, ce que l'on appelle une distribution python (Edupython par exemple dont la fenêtre apparaît ci-dessous) :



On peut écrire le programme ligne par ligne dans la *console*, ou en entier dans *l'éditeur de programme*.

- Les instructions doivent être séparées par un retour à la ligne. Certains blocs d'instructions seront *indentés*, c'est à dire qu'ils devront être décalés par rapport à la marge.

2 Les variables, affectations

Définition :

On appelle variable tout emplacement de la mémoire de l'ordinateur ou de la calculatrice dans lequel on stocke une information qui peut être changée. On dit alors qu'on *affecte* une valeur à la variable. Une variable est donc constituée :

- d'*un nom* qui permet de reconnaître où elle se situe dans la mémoire de l'ordinateur ou de la calculatrice ;
- d'*une valeur* : le nombre ou plus généralement l'information stockée.

Afin que le logiciel du langage de programmation puisse calculer l'espace en mémoire nécessaire pour stocker les valeurs de la variables, on attribue aux variables un *type* : entier, flottant (≈ décimal), caractère, chaîne de caractère,... .

Remarques :

Quelque soit le langage de programmation, un nom de variable doit respecter certaines contraintes :

- Les majuscules et les minuscules ne sont pas interchangeables : par exemple « a » et « A » ne désignent pas les mêmes variables ;
- Un certain nombre de caractères spéciaux sont interdits, il est donc recommandé d'éviter d'utiliser des caractères spéciaux dans les noms de variables : par exemple, « * » ne doit pas être utilisé dans un nom de variable (confusion possible avec l'opérateur de multiplication). Les espaces sont en outre interdits dans les noms de variables. Il est recommandé d'éviter les lettres accentuées. Les chiffres sont autorisés mais ils ne doivent pas être commencer le nom d'une variable ;
- Les mots-clefs du langage de programmation ne peuvent pas être utilisés comme nom de variables : « if » ou « for » sont interdits par exemple.

Exemple de syntaxe en algorithmique :

$a \leftarrow 3$ signifie que l'on affecte la valeur 3 à la variable a.

Déclaration des variables :

Dans un algorithme ou un programme, on commence généralement par énoncer les variables qui seront employées et la nature des données que l'on affectera à la variable : nombres entiers, nombres décimaux, chaînes de caractères. On dit que l'on « déclare » les variables.

Premiers types de variables en python :

- `int` : entier relatif (« integer » en anglais) ;
- `float` : nombre à virgule flottante, ≈ décimal ou nombre réel ;
- `bool` : booléen, variable ne pouvant prendre que les valeurs 0 ou 1, `True` ou `False` ;

La déclaration des types de variables n'est pas indispensable en langage python : l'interpréteur python attribue lui-même autant que possible, le type le plus adapté à la variable lors de la première utilisation. On peut connaître le type d'une variable en python en tapant dans la console `type(nomDeLaVariable)`.

Exemples [Programmation d'affectation en langage python] :
Le programme suivant donne des exemples d'affectations de variables :

```
a=3      # On affecte la valeur 3 à la variable a
b=a+2 # On affecte la valeur 3+2 donc 5 à la variable b
b=b*b # On affecte la valeur 5*5 donc 25 à la variable b
c=7*b # On affecte la valeur 7*25 donc 150 à la variable c
```

C'est une implémentation en python de l'algorithme suivant :

Variables *a, b, c : nombres réels*
Début traitement
$a \leftarrow 3$;
$b \leftarrow a + 2$;
$b \leftarrow b * b$;
$c \leftarrow 7 * b$;
Fin

Remarque :
Les textes qui suivent les symboles # sont appelés des *commentaires* : ils sont ignorés par le langage python mais permettent d'expliquer le code.

3 Fonctions

Définition :

> Une *fonction* en programmation est un bloc d'instructions qui a reçu un nom, dont le fonctionnement dépend d'un certain nombre de *paramètres* aussi appelés *arguments* et qui renvoie, éventuellement, un résultat.

Syntaxe en langage python :

```
def nomDeLaFonction(ArgumentsSéparésParDesVirgules):
        instruction
        ...
        instruction
        return résultatEventuel
```

Exemples [Programmation de fonctions en langage python] :

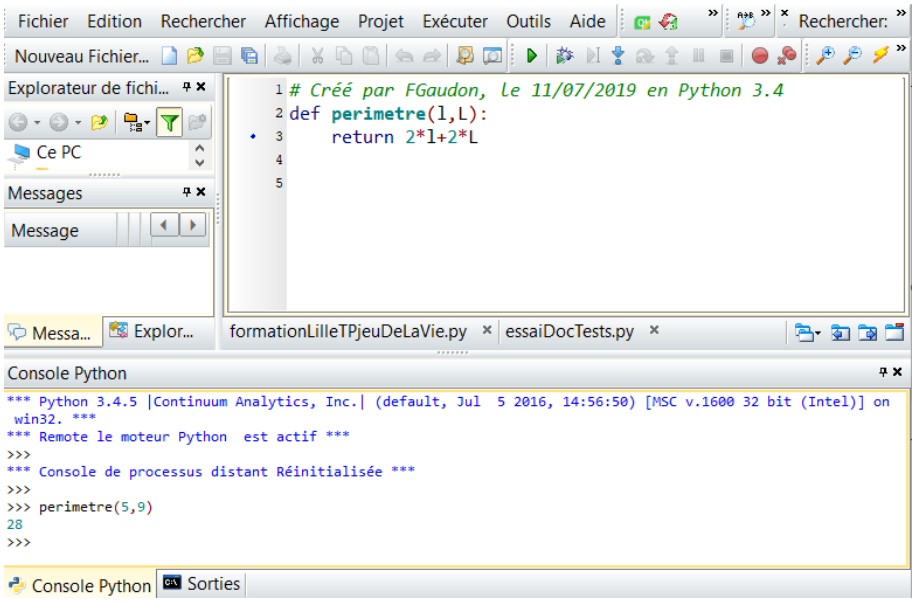
- La fonction suivante renvoie le carré d'un nombre réel a donné :

```
def calculCarre(a):
        return a*a
```

- La fonction suivante renvoie le périmètre d'un rectangle de dimensions L et l données :

```
def perimetre(l,L):
        p=2*L+2*l
        return p
```

Note : Pour pouvoir utiliser ces fonctions dans une distribution python (par exemple Edupython), les taper dans l'éditeur de programmes, les éxécuter, puis taper le nom de la fonction dans la console avec les paramètres : `calculCarre(5)` ou `perimetre(5,9)`.



4 Structures conditionnelles

4.1 Si..alors..sinon

Définitions :

> Ces instructions permettent de tester si une condition est vraie ou fausse et de poursuivre le programme d'une manière différente selon que la condition est vraie ou fausse.

Syntaxe en algorithmique :

Si
`condition`
Alors
`instructions si condition vraie`
Sinon
`instructions si condition fausse`
FinSi

Syntaxe en langage python :

```
if condition :
        instruction
        ...
        instruction
else:
        instruction
        ...
        instruction
```

Exemple [Programmation de conditions en langage python] :
La fonction suivante permet de tester si une température dépasse le seuil d'ébullition :

```
def testEbullition(T):
        if T>=100:
                return True
        else:
                return False
```

Elle renvoie comme résultat un booléen c'est à dire une variable ne pouvant prendre que les valeurs `True` « Vrai » ou `False` « Faux ».

4.2 Opérateurs relationnels et logiques

Définition :

Pour tester une condition on utilise les *opérateurs relationnels* suivants :
- $a = b$ teste l'égalité de a et de b ;
- $a < b$ teste si a est strictement inférieur à b ;
- $a \leq b$ teste si a est inférieur ou égal à b ;
- $a > b$ teste si a est strictement supérieur à b ;
- $a \geq b$ teste si a est supérieur ou égal à b ;
- $a \neq b$ teste si a est différent de b.

On utiliser ausi pour les conditions plus complexes les opérateurs logiques "et" ("AND"), "ou"("OR") et "non" ("not").

Syntaxe en langage python :
- `a==b` teste l'égalité de a et de b ;
- `a<b` teste si a est strictement inférieur à b ;
- `a<=b` teste si a est inférieur ou égal à b ;
- `a>b` teste si a est strictement supérieur à b ;
- `a>=b` teste si a est supérieur ou égal à b ;
- `a!=b` teste si a est différent de b ;
- `condition1 && condition2` qui peut aussi s'écrire `condition1 and condition2` teste si les deux conditions sont vraies simultanément ;
- `condition1 || condition2` qui s'écrit aussi `condition1 or condition2` teste si l'une au moins des deux conditions est vraie ;
- `!condition` teste si la négation de la condition est vraie.

Exemple [Programmation de conditions en langage python] :
Le programme suivant teste si une personne vérifie les conditions pour se présenter à une élection municipale : la personne doit être majeure et éligible.

```
def candidatMunicipales(age,eligibilite):
        if (age>=18) and eligibilite==True:
                return 'oui'
        else:
                return 'non'
```

On remarquera qu'ici, la fonction renvoie comme résultat une chaîne de caractère : « oui » ou « non ». On aurait pu renvoyer un booléen `True` ou `False` à la place.

5 Boucles

5.1 Boucles bornées « for »

Définition :

Ces instructions sont utilisées pour contrôler les boucles en incrémentant (augmentant) une variable lorsque l'on connaît le nombre de répétitions. La variable est augmentée d'une valeur de départ jusqu'à une valeur d'arrivée d'un pas donné (l'incrément).

Syntaxe algorithmique :

Pour
`variable`
allant de
`valeur de départ`
à
`valeur d'arrivée`
faire
`instructions`
fin

Syntaxe en langage python :

```
for nomDeLaVariableCompteur in range(valeurDeDépart,valeurArrivée+1):
        instruction
        ...
        instruction
```

Exemple [Programmation d'une boucle bornée en langage python] :
Le programme suivant permet d'obtenir la somme des n premiers entiers $1+2+...+n$ où n est donné.

```
def sommeEntiers(n):
        somme=0
        for k in range(0,n+1):
                somme=somme+k
        return somme
```

5.2 Boucles non bornées « while »

Définition :

Éxécute un groupe de commandes tant qu'une condition est vraie. On l'utilise donc lorsque l'on ne connaît pas à l'avance le nombre de répétitions. La condition est testée en début de boucle donc si la condition est vraie lorsqu'elle est testée en début de boucles, *toutes* les instructions contenues à l'intérieur de la boucle sont effectuées.

Syntaxe :
Tant que `condition`
`instructions`
faire
`instructions`
fin tant que

Syntaxe en langage python :

```
while condition:
        instruction
        ...
        instruction
```

Exemple [Programmation d'une boucle non bornée en python] :
Le programme suivant permet d'obtenir le nombre de multiplications par 2 nécessaires pour atteindre le nombre n donné à partir du nombre 1 :

```
def seuil(n):
        k=1
        i=0
        while (k<n):
                k=k*2
                i=i+1
        return i
```