Cours : séquence N°1 :

Les bases de la programmation informatique

avec les langages C et Python

Première partie

(C) Achille Braquelaire

SOMMAIRE

Vous pouvez tester le code en utilisant un compilateur en ligne :
http://www.tutorialspoint.com/compile_c_online.php
https://www.codechef.com/ide

https://replit.com/languages/python3

Ou alors un éditeur sur PC :
- codeblocks pour le C
- mu editor pour le python

I. PRÉSENTATION

1. But

Le but de ce document est d'écrire rapidement un programme **simple** dans les langages C et Python.

Dans cette optique, le choix a été fait de ne parler que du strict minimum et ne pas présenter les spécificités des deux langages. Il est évident qu'il ne peut remplacer un cours complet.

2. Langage C et Python

Le langage C est un « vieux » langage de programmation datant des années 1970. Il est considéré comme généraliste et proche du matériel.
C'est un langage compilé : le code source est transformé en code binaire directement exécutable par la machine.
Sa connaissance est incontournable en informatique.

Python lui date des années 90, également généraliste, il est bien plus facile à apprendre que le C.
Langage interprété son code est exécuté pas à pas.
Sa connaissance est également incontournable.

Visuellement, la première différence, c'est **l'absence de *main*** en début de code, des **points-virgules** à la fin de chaque ligne et des tabulations à la place d'accolades.

II. LES TYPES DE VARIABLES

1. Les types de données :

a. Langage C

Les principaux types de variables sont :

Type	Nombre d'octets	Plage de valeurs	Utilisation
char	1	-128 (2^7) à 127 (2^7-1)	Représente un seul caractère (code ASCII) et peut être utilisé pour un « petit » entier compris entre -128 et 127.
int	2 (µp 16 bits) 4 (32 bits)	-32768 (-2^{15}) à 32767 (2^{15}-1)	Entier positif ou négatif.
float	4	3.4028235E+38 à -3.4028235E+38	Nombre réel

Remarques :

- Il existe également des chaînes de caractères, mais ce sont des tableaux de caractères (voir *chaînes de caractères*)
- Le terme *unsigned* mis devant le type de la variable signifie *non signé* et représente uniquement les nombres positifs : la plage de valeurs est ainsi doublé.

Par exemple :

unsigned char	**Taille = 1 octet**	**De 0 à 255 (2^8-1)**	**Représente un seul caractère (code ASCII) ou un entier compris entre 0 et 255.**

b. Python

Les types de données de base sont :

Type	Utilisation
int	Entier positif ou négatif
float	Nombre réel à virgule flottante
str	Chaîne de caractère
complex	Nombre complex
bool	Booléen : True ou False uniquement

2. Déclaration et affectation de variables.

Afin que votre code soit le plus compréhensible possible, il faut veiller à utiliser des noms de variables pertinents : le nom de votre variable doit indiquer sa fonction.

`aire, perimetre, volume` à la place de `a, b, c`

Exception faite pour les variables d'incrémentation dont les noms sont traditionnellement `i, j ou k`

a. En langage C : obligatoire

Déclaration : une variable doit être obligatoire déclarée avant d'être utilisée :

```
int i ;
```

On peut déclarer plusieurs variables de même type sur la même ligne :

```
float aire, perimetre, volume ;
```

Pour des raisons de lisibilité, il est fortement conseillé de déclarer les variables au début du code.

Affectation : par la suite dans le code, on peut affecter une valeur à la variable.

```
aire = -4.2;
```

Déclaration et affectation : On peut donner à la variable, une valeur initiale.

```
int vit= -4;
char i, j, toto ;
```

b. En Python : lors de la déclaration

Il n'y a pas de déclaration de variable, la définition du type de la variable se fait automatiquement lors de l'affectation :

```
x = 3
pi = 3.14159
message = 'Coucou'
```

Il est possible de connaître le type d'une variable par la fonction *type :*

```
type(x)
type(pi)
type(message)
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

3. Exemples

Langage C	Python

`#include <stdio.h>`
`#include <string.h> // pour strcpy`
`int main()`
`{`
`  /* Déclaration */`
`  int i ;`
`  float rayon ;`
`  char nom[10]; // 10 caractères maxi`
`  char ma_lettre; // 1 caractère`
 	
`  /* Affectation */`
`  i=0;`
`  rayon=8.2;`
`  strcpy(nom,"Nestor");`
`  ma_lettre = 'd';`
 	
`  return 0;`
`}`
	`i = 0`
	`rayon = 8.2`
	`nom = "Nestor" # ou nom='Nestor'`
	`OuiNon = True`

III. INTÉRAGIR AVEC L'UTILISATEUR

1. Afficher

a. Langage C : printf

La fonction *printf* se trouve dans la bibiothèque standard : `#include <stdio.h>`

Elle permet de réaliser des sorties formatées de messages et/ou de valeurs des variables sous différents formats.Les formats disponibles sont :

- %d : *Entier Décimal*
- %x : *Entier Hexadécimal*
- %u : *Entier Non Signé*
- %c : *Caractère*
- %s : *Chaîne de caractères*
- %f : *Flottant*

b. Python : print sans le 'f'

L'affichage peut se faire simplement sans avoir recours à l'écriture formatée.

➢ Pour un texte :

```
print ('Hello World')
print ('Hello' + 'World') #Affiche HelloWorld
print (3 * '*') #Affiche ***
```

➢ Une variable :

```
rayon = 8.2
print (rayon)
```

➢ Texte + variable :

```
print ('rayon = ', rayon)
```

Ou alors convertir la variable en string pour la concaténer avec la chaîne 'rayon' comme ceci :

```
print('rayon = ' + str(rayon) )
```

2. Lire

a. C : Lire avec scanf

Elle permet de saisir des valeurs de variables formatées à partir du clavier. Comme printf elle est composée d'un format et des identificateurs de variables à saisir.
A la **différence** de printf, le format ne peut contenir de texte, il est juste composé du format des valeurs à saisir.

scanf (format, d'adresses) : permet de saisir les données au clavier

Exemple 1	Exemple 2

`scanf("%d",&a);`
`// Attend la saisie d'un entier`
	`scanf("%d%d%f",&a,&b,&c);`
	`/* Attend la saisie de deux entier puis d'un float`
	`Tous seront séparés par des espaces */`

Remarques:

- ➢ Le symbole & est obligatoire sauf pour les tableaux
- ➢ Les formats disponibles sont les mêmes que pour *printf* :
  - %d : *Entier Décimal*
  - %x : *Entier Hexadécimal*
  - %u : *Entier Non Signé*
  - %c : *Caractère*
  - %s : *Chaîne de caractères*
  - %f : *Flottant*

3. Python : input

La fonction essentielle est *input* qui lit une chaîne de caractère :

```
print('Entrez un message')
message = input()
```

```
print(message)
```

Une écriture équivalente quasi (hormis le saut de ligne) équivalente est :

```
message = input('Entrez un message : ')
print(message)
```

Pour les autres types de variables que *string*, il faut convertir la chaîne :

- Pour lire un entier : `i = int(input())`
- Un réel : `rayon = float(input())`

4. Exemples

Langage C	Python

`#include <stdio.h>`
 	
`int main ()`
`{`
`  /* Déclarations et affectation */`
`  float rayon = 8.2;`
`  int i;`
`  // Chaine de 10 caractères maxi :`
`  char chaine [10];`
 	
`  // Avec un saut de ligne \n :`
`  printf("Hello World \n");`
 	
`  //Affiche HelloWorld`
`  printf("Hello");`
`  printf("World\n");`
`  // SNEC 3 fois`
`  printf("SNECSNECSNEC\n");`
 	
`  printf("%f\n", rayon);`
`  printf("rayon = %f \n", rayon);`
 	
`  printf("Entrez un message :");`
`  // Attention pas de & devant message :`
`  scanf("%s", chaine);`
`  printf("Le message est %s \n", chaine);`
 	
`  printf ("i = ");`
`  scanf ("%d",&i);`
`  printf ("rayon = ");`
`  scanf ("%f",&rayon);`
`  printf ("i = %d \n",i);`
`  printf ("rayon = %f\n",rayon);`
 	
`  return 0;`
`}`
	`print ('Hello World')`
	
	`#Affiche HelloWorld :`
	`#Ces deux lignes affiche HelloWorld0000`
	`print ('Hello' + 'World')`
	`# end = "" -> sans saut de ligne`
	`print ('Hello', end = "");`
	`print ('World');`
	
	`#Affiche SNECSNECSNEC`
	`print (3 * 'SNEC')`
	
	`rayon = 8.2`
	`print (rayon)`
	`print ('rayon = ', rayon)`
	
	`message = input ('Entrez un message : ')`
	`#Avec + pas de saut de ligne`
	`print ('Le message est ' + chaine)`
	
	`i = int (input ('i =  '))`
	`rayon = float (input ('rayon = '))`
	`print ('i = ' + str (i))`
	`print ('rayon = ' + str (rayon))`

IV. LES BRANCHEMENTS CONDITIONNELS

1. Si ... alors…(sinon)

a. Syntaxe Langage C

Pour faire des choix : l'instruction if

Condition du test : ==, <,>,<=,>=, !=,&&,|| …

Algo	Langage C

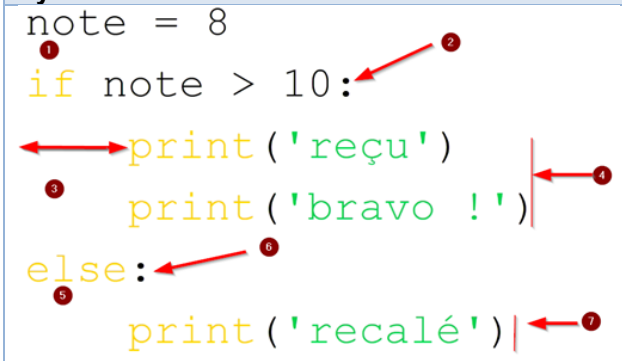
Si a > b
Alors
Afficher "A>B"
Sinon
Afficher "A>B"
	`if (a > b)`
	`  {`
	`  printf("A>B \n");`
	`  }`
	`else`

FinSi	`{`
	`  printf("A<B \n");`
	`}`

Remarques

- Le code est plus lisible avec des tabulations
- il ne faut pas confondre en C, l'opérateur d'égalité **==** et celui de l'affection **=**
- *else* (*sinon)* est optionnel,
- il est possible d'imbriquer des if avec else if

b. Syntaxe Python

Python	Explications
	1. L'instruction if 2. Le test conditionnel se termine par « : » le signe **deux points.** 3. A la place des parenthèses du langage C, python utilise **l'indentation** : quatre espaces ou une tabulation, nous préférons le second format 4. Un bloc d'instruction (une ou plusieurs lignes) dans le cas où la condition est vérifiée 5. Else pour indique le cas où la condition n'est pas vérifiée 6. A nouveau les deux points 7. Et un bloc d'instructions

Remarques

- il ne faut pas confondre l'opérateur d'égalité **==** et celui de l'affection **=**
- *sinon* est optionnel

Les condition du test sont les mêmes que pour la langage C : ==,<,>,<=,>=, !=, sauf **and**, **or.**

Il est possible Avec else if contracté en elif
Remarquer les **indentations** (elif comme else est au même niveau que le if)

```
x = 'Roger'
if x == 'roger':
    print("manque une majuscule")
elif x == 'Roger ':
    print("un espace en trop")
else:
    print('Pas trouvé!')
```

c. Exemple

Langage C	Python

`if (note>10)`
`  {`
`  printf("reçu");`
`  printf("bravo");`
`  }`
`  else`
`  {`
`  printf("recalé");`
`  }`
	`if note>10:`
	`  print("reçu")#une tabulation avant`
	`  print("bravo")`
	`else:`
	`  print("recalé")`

2. Selon … cas …

Cette structure remplace une série de if consécutifs avec l'avantage d'une meilleure lisibilité.

a. Syntaxe langage C

Algo	Langage C

Saisir jour
Selon jour :
Cas 1 : afficher "lundi"
Cas 2 : afficher "mardi"
…
Cas 7 : afficher "dimanche"
	`int jour ;`
	`scanf("%d", &jour);`
	
	`switch (jour) {`
	`case 1 : printf("lundi\n"); break;`
	`case 2 : printf("mardi\n"); break;`
	`…`
	`case 7 : printf("dimanche\n");`
	`break;`
	`default : printf ("Erreur : le jour doit être compris entre 1 et 7\n");`
	`}`

Remarques sur la syntaxe :

- Tous les cas sont dans une même accolade `{…}`
- Chaque ligne commence par `case x :` (les deux points sans le point-virgule)
- `x` doit être de type entier
- Mais comme 'A' est le code ascii de la lettre A, il est possible d'écrire `case 'A' :`
- Faire attention de bien mettre l'instruction `break`; qui fait sortir du `switch – case`, sinon on continue jusque `default` !!
- Si aucune valeur n'existe, les instructions sous `default` sont exécutées

b. Python : n'existe pas

A la place, il faut utiliser des elif :

```c
int jour;
printf("Entrez un jour : ");
scanf("%d", &jour);
printf("\n");
switch (jour) {
case 1 : printf("lundi\n"); break;
case 2 : printf("mardi\n"); break;
  // et ainsi de suite
case 7 : printf("dimanche\n"); break;
default : printf("Erreur :\n");
```

```python
jour =int(input("Entrez un jour : "))
if (jour==1):
    print("Lundi")
elif (jour==2)
    print("Mardi")
    #et ainsi de suite
else:
    print("Erreur : ")
```

V. COMMENTAIRES ET BIBLIOTHÈQUES

1. Les commentaires

Un code bien écrit doit être facilement compréhensif : en le lisant on doit comprendre ce que vous avez fait. Pour cela, il faut au moins utiliser des noms de variables qui indiquent leurs fonctions.Si cela n'est pas suffisant, les commentaires servent à préciser votre code.

a. Langage C

Si le commentaire tient sur une seule ligne, on peut précéder l'explication par « // » :

```
// Sur une ligne
```

Pour plusieurs lignes, le commentaire doit être encadré par « /* » et « */ » à la fin :

```
/* Sur plusieurs
lignes   */
```

b. Python

L'explication sur un ligne débute par « # » :

```
# Sur une ligne
```

Et sur plusieurs lignes, encadré par « """ »

```
""" Sur plusieurs
lignes   """
```

2. Les bibliothèques (ou Library en anglais)

a. Langage C : include

Pour pouvoir utiliser `printf(…)` il faut mettre au début du code `#include <stdio.h>` pour que le compilateur sache que cette fonction est définie « ailleurs ».

```
#include <stdio.h>

int main ()
{
  printf("Hello World \n");
  return 0;
}
```

Toutes les fonctions nécessitent d'inclure des fichiers d'entête

b. Python : import

L'équivalent à `#include` est **import** en langage Python.

Par exemple pour importer la fonction sqrt (square root = racine carrée) :

```
from math import sqrt
print(sqrt(16))
```

Il est possible d'en importer plusieurs à la fois :

```
from math import sqrt, pi, cos
print(cos(pi/2))
```

Et même toutes les fonctions du module math :

```
import math
print(math.sqrt(16)) # Affiche 4
```

Un autre exemple où en plus d'importer, on a défini un synonyme (PI en majuscule au lieu de pi)

```
from math import pi as PI
print(PI)
```

Et même renommer un module :

```
import numpy as np
import matplotlib.pyplot as plt
```

VI. LES STRUCTURES DE CONTRÔLE

1. La boucle for pour répéter un certains nombre de fois

Cette boucle est utilisée lorsqu'on connait le nombre d'itérations avant même de l'appeler c'est-à-dire :
La boucle pour est utilisée lorsque l'on connait le nombre de fois que la boucle va être parcourue

Par exemple au lieu d'écrire :

```
print('bonjour') # ceci est un commentaire 1
print('bonjour') # 2
print('bonjour') # 3
print('bonjour') # 4
print('bonjour') # 5
print('bonjour') # 6
print('bonjour') # 7
print('bonjour') # 8
print('bonjour') # 9
print('bonjour') # 10
```

Il est possible d'utiliser la boucle *for* :

Langage C	Python

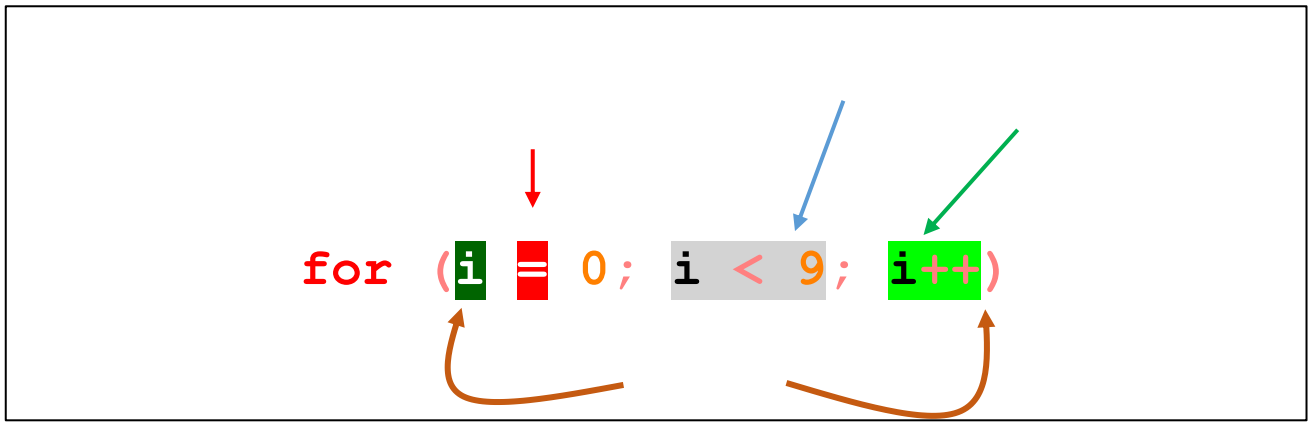
`int i ;`
`for (i = 0; i <= 9; i++)`
`{`
`printf("coucou\n") ;`
`}`
	`for i in range(0,10):`
	`print('bonjour') #tabulation au début`

a. Langage C : for

Algo	Langage C

Pour i variant de 0 à 9
afficher("coucou)
FinPour
	`int i ;`
	`for (i = 0; i <= 9; i++)`
	`{`
	`printf("coucou\n") ;`
	`}`

Explications :



Autres exemples

Exemple 1	Exemple 2

`char car;`
`for(car='A';car!='Z'+1;car++)`
`printf("%c",car);`
	`char i;`
	`for (i = 0 ; i <= 100 ; i++)`
	`{`
	`printf("i = %d\n", i); /*affiche les i de 0 à 100*/`
	`}`

Remarques

- Souvent on voit dans les codes `i++` c'est équivalent à `i=i+1` ou même à `i+=1`
- Généralement, en informatique, on commence à compter à partir de zéro :

`for (int i = 0; i < 9; i++)` est équivalent à `for (int i = 1; i <=10 ; i++)`

- `for ( ;;)` est une boucle infinie

b. Python : toujours for

Comme pour le if, à la place des parenthèses du langage C, python utilise **l'indentation** qui est soit une tabulation, soit quatre espaces.

La fonction range()

La fonction *range* permet de créer une liste de nombres compris entre un nombre de départ (inclus) et un nombre de fin (exclus).

Range() a trois paramètres : début, fin et pas, exemple :

```
for i in range(10): # de 0 à 9 (10 non inclus)
for i in range(1,10): # de 1 à 9
for i in range(0,100,5): # de 0 à 95 par pas de 5
for i in range(0,-10,-3): # de 0 à -9 par pas de -3
```

Par exemple pour afficher 10 fois bonjour :

```
for  i in range(0,10):
    print('bonjour') # indentation au début de ligne
```

c. Exemple simple

```
for( i = 1; i <=10; i=i+1) {
    printf("i=%d\n",i);
    }
```

```
for i in range(1,11):
    print("i=",i)
```

Boucles imbriquées :

```
#include <stdio.h>

int main()
{
  int i, j;
  for (i = 0; i <= 100; i++)
  {
    for (j = 0; j <= 10; j++)
    {
      if (j % 2 == 0)
      {
        printf("Pair\n");
      }
      printf("%d\n", i * j);
    }
    printf("%d\n", i);

  }
  printf("Fin du programme\n");
  return 0;
}
```

```
                }
        printf("Fin du programme\n");
    return 0;
}
```

```
for i in range(101):
  for j in range (10):
    print(j)
    if (j%2 == 0) :
      print('Pair') # dans le if
    print(i*j) # dans le 2 for
  print(i) # dans le 1er
print('Fin du programme') # à la fin
```

2. While pour répéter en respectant une condition

Cette boucle est utilisée quand on ne connait pas le nombre de fois que la boucle doit être itérée.

a. Exemple en Langage C

Algo	Langage C

Somme<-0
Tant que somme<100
faire somme=2*somme+1
	`Somme=0 ;`
	`while (somme < 100)`
	`{`
	`somme = 2 * somme + 1;`
	`}`

Remarques

- Attention : La condition se trouvant au début de la boucle, il faut s'assurer qu'elle soit bien vérifiée. Dans l'exemple ci-dessus, Somme est mis à zéro
- Il n'y a pas de point-virgule à la fin de la ligne contenant le **while**
- **while (1) {…}** est une boucle infinie.

b. En python

C'est exactement la même syntaxe qu'en langage C. Il ne faut pas oublier les deux points « : » à la fin de la ligne et l'indentation (tabulation ou espace).

```
x=45
y=55
while x<50 and y <70:
    x=x+1
    y=y+1
    print(x,y)
```

c. Exemples

Tant que la somme <100

```
#include <stdio.h>
int main() {
  int somme = 0;
  while (somme < 100)
  {
    somme = 2 * somme + 1;
  }

  printf("somme=%d\n", somme);
// Affiche somme=127
  return 0;
}
```

```
somme = 0
while somme<100:
    somme = 2*somme + 1
print('somme=',somme)
```

A la place d'une boucle for

Langage C	Python

`#include <stdio.h>`
`int main() {`
`for( i = 1; i <=10; i=i+1) {`
`printf("i =%d \n",i);`
`}`
`return 0;`
`}`
	`for i in range(1,11):`
	`print("i =",i)`

`#include <stdio.h>`
`int main() {`
`int i = 1;`
`while (i <= 10)`
`{`
`printf("i=%d\n", i);`
`i ++;`
`}`
`return 0;}`
	`i = 1`
	`while ( i <10 ):`
	`print("i =",i)`
	`i = i + 1`

3. La boucle faire … tant que : uniquement en C

La condition n'est vérifiée qu'à la fin de la boucle : celle-ci sera exécutée au moins une fois.

a. Exemple

Algo	Langage C

Reponse : caractère
répéter
Afficher ("Voulez-vous sortir (O/N) ?")
lire reponse
tant que reponse ≠ 'O'
	`char caractere ;`
	`do`
	`{`
	`printf("Voulez-vous sortir (O/N) ?");`
	`scanf("%c", &caractere);`
	`}`
	`while (caractere != 'O');`

b. Remarque

- Attention aux accolades et au point-virgule à la fin de while.

VII. LES FONCTIONS

1. Le concept de fonction

Les fonctions déjà présentes jusqu'alors

Vous avez déjà utilisé en C/Python des fonctions : `printf(…), scanf(…), main()` ou `print(…), int(…), range(…)` Elles sont reconnaissables par des parenthèses à la fin de leur nom. Elles ont toutes été conçues et mises au point par d'autres personnnes, puis réunies dans des librairies (C) ou modules (Python).

Besoin :

Dès que le programmme commence à prendre de l'ampleur, il est nécessaire d'écrire vos propres fonctions : un programme de plus de cent lignes commencent à être illisible, difficile à corriger et des parties sont répétitives. De plus, il est nécessaire de décomposer votre travail, en sous-tâches, qui vont être codées en fonctions.

Une fonction c'est :

Une suite instructions isolées du reste du programme, qui possède un nom, et qui peut être appelée par ce nom à n'importe quel endroit du programme et autant de fois que l'on veut.

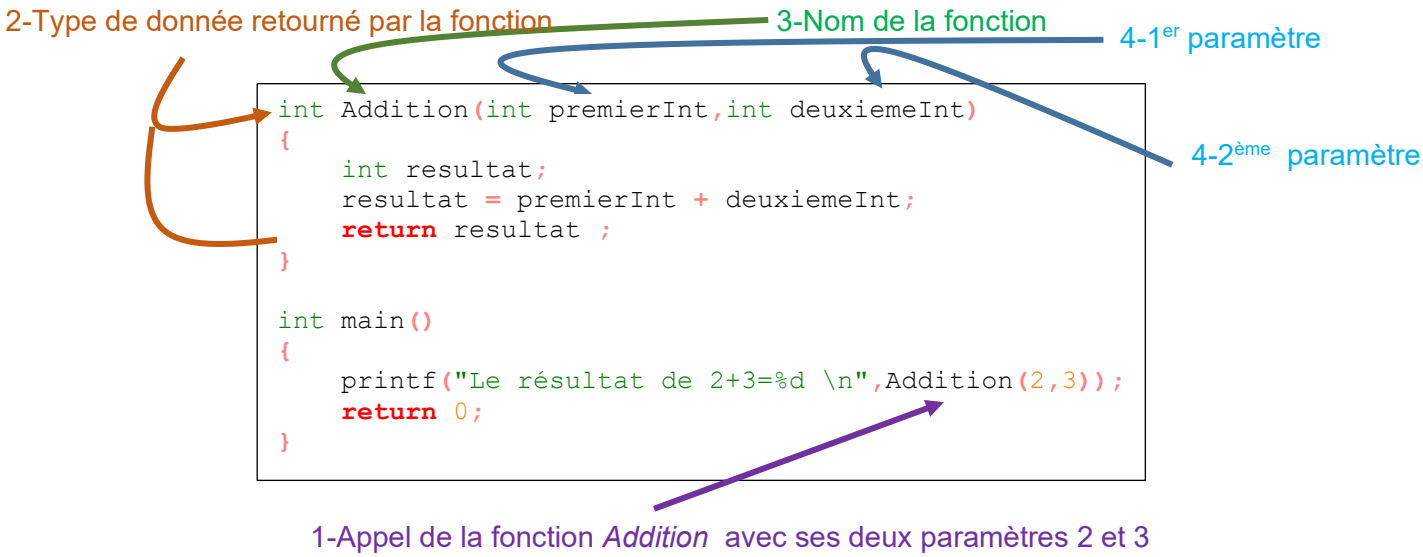
Points essentiels :

- Un programme écrit sans fonction devient difficile à comprendre dès qu'il dépasse un centaine de lignes
- Les fonctions permettent de scinder le programme principal en plusieurs parties
- Le programme principal regroupe les fonctions en décrivant les enchaînements
- Une fonction peut elle-même, être découpée en plusieurs autres fonctions.
- Et pour terminer, et peut-être le plus essentiel, les fonctions permettent le partage de tâches : dans un projet important où travaille plusieurs développeurs, chacun est responsable d'une ou de plusieurs fonctions, qui sont ensuite utilisées par les autres programmeurs et mise en commun dans le programme principal.

Dans cette partie, nous ne parlerons pas de modularité (découper le fichier principal en plusieurs fichiers), mais cette notion est indispensable pour les plus longs programme.

2. Les fonctions en Langage C

a. Exemple commenté




```
int Addition(int premierInt,int deuxiemeInt)
{
    int resultat;
    resultat = premierInt + deuxiemeInt;
    return resultat ;
}

int main()
{
    printf("Le résultat de 2+3=%d \n",Addition(2,3));
    return 0;
}
```

1. Cette fonction *Addition* est appelée dans le *main* avec deux paramètres déclarés comme étant des entiers au tout début de la définition de la fonction. *Addition* renvoie toujours dans le *main* une valeur de retour.
2. Dans la déclaration de la fonction, le premier terme indique le type de donnée qui va être retourné (ici un entier int)
3. Le second terme est le nom de la fonction
4. Ensuite les définitions des paramètres

Remarques :

- **Ne jamais mettre de printf dans les fonctions, sauf nécessité, utilisez *return* pour renvoyer une valeur puis l'afficher.**
- L'instruction *return* renvoie une valeur et met fin à l'exécution de la fonction et redonne le contrôle à la fonction appelante.
- Par (mauvaises) habitudes de nombreux programmeurs utilisent des parenthèses pour encadrer la valeur de retour. *return* est une **instruction** pas une fonction

b. Fonction sans paramètre ne retournant pas de valeur Exemples

```c
#include <stdio.h>

void Affiche10Fois() //Fonction sans paramètre ne retournant pas de valeur
{
    int i;
    for( i = 0; i < 9; i++) printf("coucou\n") ;
}
```

c. Fonction sans paramètre, retournant une valeur

```c
int Aleatoire() // Fonction sans paramètre, retournant une valeur
{
  srand(time(NULL)); // indispensable pour obtenir un nombre différent
                     //à chaque fois
  return (rand() % 20 );  // % 20 représente le reste de la division entière par 20
}
```

d. Deux paramètres sans retour de valeur

```c
/* Fonction utilisant un ou plusieurs paramètres
    ne retournant pas de valeur */
void AfficherAddition(int i, int j)
{
    printf("L'addition entre %d et %d est %d \n",i,j,i+j);
}
```

e. Plusieurs paramètres et retournant une valeur

```c
//Fonction utilisant un ou plusieurs paramètres, retournant une valeur
float Discriminant(float b, float a, float c)
{
    float delta = b*b - 4*a*c;
    return(delta);
}
```

f. Les fonctions sont regroupées dans le *main*

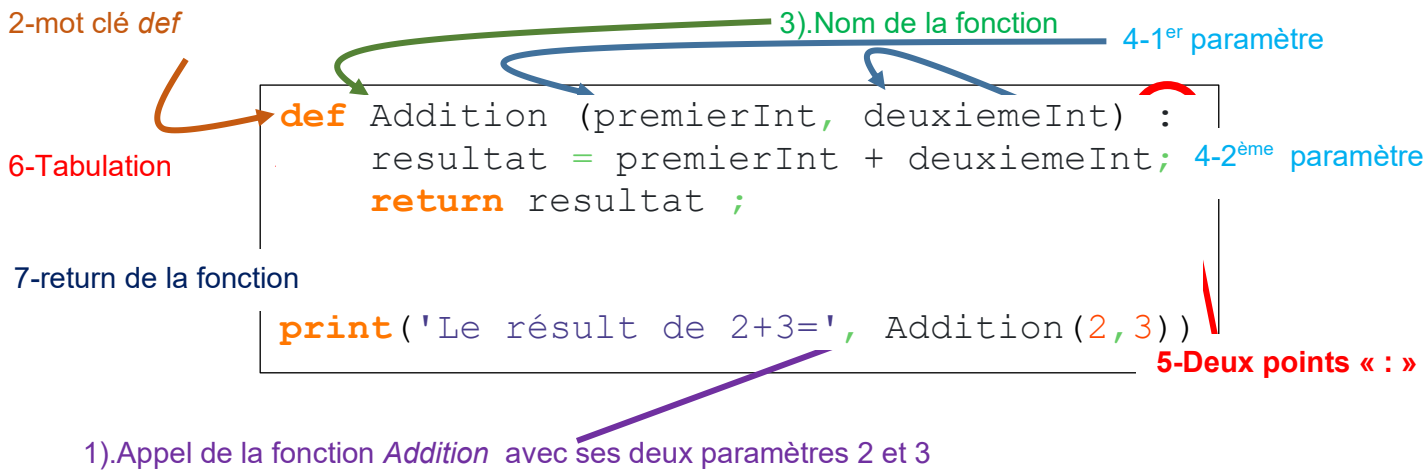
```c
int main()
{
   int nombre;
   Affiche10Fois();
   nombre=Aleatoire();
   printf("un nombre aléatoire entre 0 et 20 : %d\n",nombre);
   AfficherAddition(2,3); // Affiche L'addition entre 2 et 3 est 5
   nombre = Discriminant(6,3,2);
   printf("Le Discriminant est %d \n",nombre); // Affiche L'addition entre 2 et 3 est 5
return 0;
}
```

Remarques

- Quatre fonctions ont été écrites puis utilisées dans *main*
- le programme principal devient alors court et lisible

3. Les fonctions en Python

a. Exemple commenté



1. Cette fonction *Addition* est appelée avec deux paramètres. *Addition* renvoie une valeur de retour.
2. Dans la déclaration de la fonction, le premier terme est **def**
3. Le second terme est le nom de la fonction
4. Ensuite les définitions des paramètres
5. **Deux points** « **:** »
6. **Une tabulation** (ou indentation)
7. Un renvoie de la valeur par *return*

Remarques :

- On retrouve les deux points et la tabulation des boucle *for* et *while*
- **Ne jamais mettre de print dans les fonctions, sauf nécessité, utilisez *return* pour renvoyer une valeur puis l'afficher.**
- L'instruction *return* renvoie une valeur et met fin à l'exécution de la fonction et redonne le contrôle à la fonction appelante.
- Par (mauvaises) habitudes de nombreux programmeurs utilisent des parenthèses pour encadrer la valeur de retour. *return* est une **instruction** pas une fonction

b. Fonction sans paramètre ne retournant pas de valeur

```
import random


def Affiche10Fois():
    '''
    Fonction sans paramètre
    ne retournant pas de valeur
    '''
    for i in range(10):
        print('coucou')
```

c. Fonction sans paramètre, retournant une valeur

```
def Aleatoire():
    '''
    Fonction sans paramètre,
    retournant une valeur
    '''
    return random.randint(0, 20)
```

d. Deux paramètres sans retour de valeur

```
def AfficherAddition(i, j):
```

```
    '''
    Fonction utilisant un ou plusieurs paramètres
    ne retournant pas de valeur
    '''
    print('addition entre',i,'et','j','est',i+j)
```

e. Plusieurs paramètres et retournant une valeur

```
def Discriminant(b, a, c):
    '''
    Fonction utilisant plusieurs paramètres
    retournant une valeur
    '''
    delta = b * b - 4 * a * c;
    return delta
```

f. Les fonctions sont appelées

```
Affiche10Fois()
nombre = Aleatoire()
print('un nombre aléatoire entre 0 et 20 : ', nombre);
AfficherAddition(2, 3)  # Affiche L'addition entre 2 et 3 est 5
nombre = Discriminant(6, 3, 2);
print('Le Discriminant est ', nombre); # Affiche le delta
```

Remarques

- Quatre fonctions ont été écrites puis utilisées plus bas
- le programme principal devient alors court et lisible
- il est possible de « mimer » une fonction *main* comme étant le programme principal à l'aide de l'instruction

```
if __name__ == "__main__":
```

4. Comparaison

Langage C

```c
#include <stdio.h>
#include <stdlib.h> // Pour rand
#include <time.h> // Pour time

void Affiche10Fois() //Fonction sans paramètre ne retournant pas de
valeur
{
    int i;
    for( i = 0; i < 9; i++) printf("coucou\n") ;
}

int Aleatoire() // Fonction sans paramètre, retournant une valeur
{
  srand(time(NULL)); // indispensable pour obtenir un nombre différent
                     //à chaque fois
  return (rand() % 20 );  // % 20 représente le reste de la division
entière par 20
}

/* Fonction utilisant un ou plusieurs paramètres
    ne retournant pas de valeur */
void AfficherAddition(int i, int j)
{
    printf("L'addition entre %d et %d est %d \n",i,j,i+j);
}

//Fonction utilisant un ou plusieurs paramètres, retournant une valeur
float Discriminant(float b, float a, float c)
{
    float delta = b*b - 4*a*c;
    return(delta);
}


int main()
{
   int nombre;
   Affiche10Fois();
   nombre=Aleatoire();
   printf("un nombre aléatoire entre 0 et 20 : %d\n",nombre);
   AfficherAddition(2,3); // Affiche L'addition entre 2 et 3 est 5
   nombre = Discriminant(6,3,2);
   printf("Le Discriminant est %d \n",nombre); // Affiche le delta

    return 0;
}
```

Python

```python
import random


def Affiche10Fois():
    '''
    Fonction sans paramètre
    ne retournant pas de valeur
    '''
    for i in range(10):
        print('coucou')


def Aleatoire():
    '''
    Fonction sans paramètre,
    retournant une valeur
    '''
    return random.randint(0, 20)


def AfficherAddition(i, j):
    '''
    Fonction utilisant un ou plusieurs paramètres
    ne retournant pas de valeur
    '''
    print('addition entre',i,'et','j','est',i+j)



def Discriminant(b, a, c):
    '''
    Fonction utilisant plusieurs paramètres
    retournant une valeur
    '''
    delta = b * b - 4 * a * c;
    return delta


Affiche10Fois()
nombre = Aleatoire()
print('un nombre aléatoire entre 0 et 20 : ', nombre);
AfficherAddition(2, 3)  # Affiche L'addition entre 2 et 3 est 5
nombre = Discriminant(6, 3, 2);
print('Le Discriminant est ', nombre); # Affiche le delta
```