Cours Introduction à la Programmation Python I

(IP1 Python)

Arnaud Sangnier
sangnier@irif.fr

Jeudi 7 Septembre 2017
MIASHS et MATHS

But du cours

- Apprendre les bases de la programmation
- Être capable de comprendre des programmes
- Être capable d'écrire des programmes simples
- Langage utilisé : **Python 3**
- **Points positifs**
  - Cours sans difficulté théorique
  - Savoir programmer est un atout important
- **Points 'négatifs'**
  - Travail régulier nécessaire
  - Fort taux d'échec à l'examen
  - Besoin de beaucoup de rigueur dans l'écriture des programmes

Organisation des enseignements

- **Cours**
  - 6 cours
  - Un mercredi sur deux de 15h45 à 17h45, amphi 1A
  - **20/09** – **4/10** – **18/10** – **8/11** – **22/11**
- **Cours/td**
  - 2h par semaine
  - Début : semaine du 18 septembre
- **Tp**
  - 2 fois 2h par semaine
  - Début : semaine du 18 septembre
- **Tutorat**
  - Accès libre
  - Tous les jours entre 12h et 14h (horaires et salles à préciser)
  - Début : semaine du 25 septembre

Organisation des enseignements

- **Cours**
  - 6 cours
  - Un mercredi sur deux de 15h45 à 17h45, amphi 1A
  - **20/09** – **4/10** – **18/10** – **8/11** – **15/11**
- **Cours/td**
  - 2h par semaine
  - Début : semaine du 18
- **Tp**
  - 2 fois 2h par s
  - Début : se eptembre
- **Tutorat**
  - Accès libre
  - Tous les jours entre 12h et 14h (horaires et salles à préciser)
  - Début : semaine du 25 septembre

Important : Respecter votre groupe

Évaluation

- **Contrôle continu cours/td**
  - 2 épreuves
- **Contrôle continu Tp**
  - 2 épreuves
- **Partiel**
  - Un partiel de 2h le **samedi 4 novembre de 12h30 à 14h30**
- **Examen**
  - Un examen de 3h en décembre

Note

Td : Résultats des cours/td

Tp : Résultat des tp

P : Résutat du partie

E : Résultat de l'examen

- Note finale Cc : (Td + Tp) / 2
- Note écrit Ne : Max (E, (E + P) / 2)
- **Note session 1 : (3*Ne + Cc) / 4**
- **Remarques :**
  - Absence aux CC : 0
  - Absence au partiel : 0
  - Absence à l'examen : pas de note
  - **Une mauvaise note au partiel est rattrapable**

Points sur le contenu

- Les bases de la programmation seront présentés en cours/td
  - Des supports vous seront distribués
- Les Tp servent à mettre en pratique ces bases
  - Les énoncés seront sur Moodle
- En amphi :
  - Présentation de concepts généraux
  - Aide à la compréhension de ce qui est vu en td/tp
  - Correction partielle des examens de l'année précédente

Communication

- N'hésitez pas à communiquer avec vos chargés de cours/td et tp
- Vous pouvez aussi m'écrire : **sangnier@irif.fr**
- Nous lisons tous nos mails régulièrement
  - Respecter cela dit les règles de courtoisie dans vos mails
    - N'oubliez pas de **signer votre mail**, d'**écrire sans faute d'orthographe**, de préciser votre groupe etc
- N'hésitez pas à refaire les exercices chez vous et à vous adresser à vos encadrants en cas de doute
- **Évitez d'envoyer un programme tapé dans un mail ou dans un document Word !!!!**
- **Page Moodle du cours** (toutes les infos y sont données) :
  - https://moodlesupd.script.univ-paris-diderot.fr/course/view.php?id=2825
- **Page internet du cours** (pour les supports)
  - https://www.irif.fr/~sangnier/enseignement/ip1-python.html

Programmer

- Pour les TPs, il vous faut un **login** et un **mot de passe** pour pouvoir vous connecter (donnés au moment de votre inscription)
- Comment travailler vos cours :
  - Écrire les programmes sur feuille sans les tester n'est pas suffisant
  - Il faut écrire des programmes chez vous ou en salle de TP et tester qu'ils fonctionnent bien
  - La voie vers le succès pour ce cours : **programmer encore et encore**

Qu'est ce qu'un programme ?

- Un programme est une **suite d'instructions** qui pourra être 'exécutée' par la machine
- Quelles sont les instructions disponibles
  - Faire un calcul arithmétique (par ex. 12 * 5)
  - Afficher une chaîne de caractères
  - Déplacer la souris
  - Lancer un autre programme
  - Manipuler des données
  - Jouer un son
  - etc

Où trouve-t-on les programmes ?

- Tout ce que vous utilisez sur une machine telle qu'un ordinateur, une tablette ou un smartphone est un programme
  - Les applications
  - Les logiciels
  - Mais aussi le système qui fait fonctionner votre appareil (androis, IOS, Windows, Linux,…)

Comment écrit-on un programme ?

- Un programmeur écrit un programme dans un langage de programmation
- Il existe plusieurs langages de programmation et plusieurs familles de langage de programmation
  - **Langages Orientée Objets :**
    - Ex : Java, **Python**
  - **Langage Impératif :**
    - Ex : C
  - **Langage Fonctionnelle :**
    - Ex : CAML, OCAML,

Comment la machine comprend tous les langages ?

- Le langage de programmation est un langage 'compréhensible' par les humains
- Les instructions sont un mélange d'anglais et d'opérations mathématiques, plus certaines instructions spécifiques à chacun des langages
- Le programme écrit par le programmeur est contenu dans un fichier, on parle de **code source**
- Le code source est ensuite soit **traduit** vers un langage compréhensible par la machine (langage binaire), on parle de **compilation**, soit il est interprété par un **interpréteur** qui exécute ces instructions (langage interprété)
- Pour pouvoir exécuter un programme, il faut donc soit avoir le compilateur (Java) ou l'interpréteur (Python, OCaml)
- **Remarques : L'interpréteur et le compilateur sont eux-mêmes des programmes**

Schéma d'exécution d'un code source avec compilation

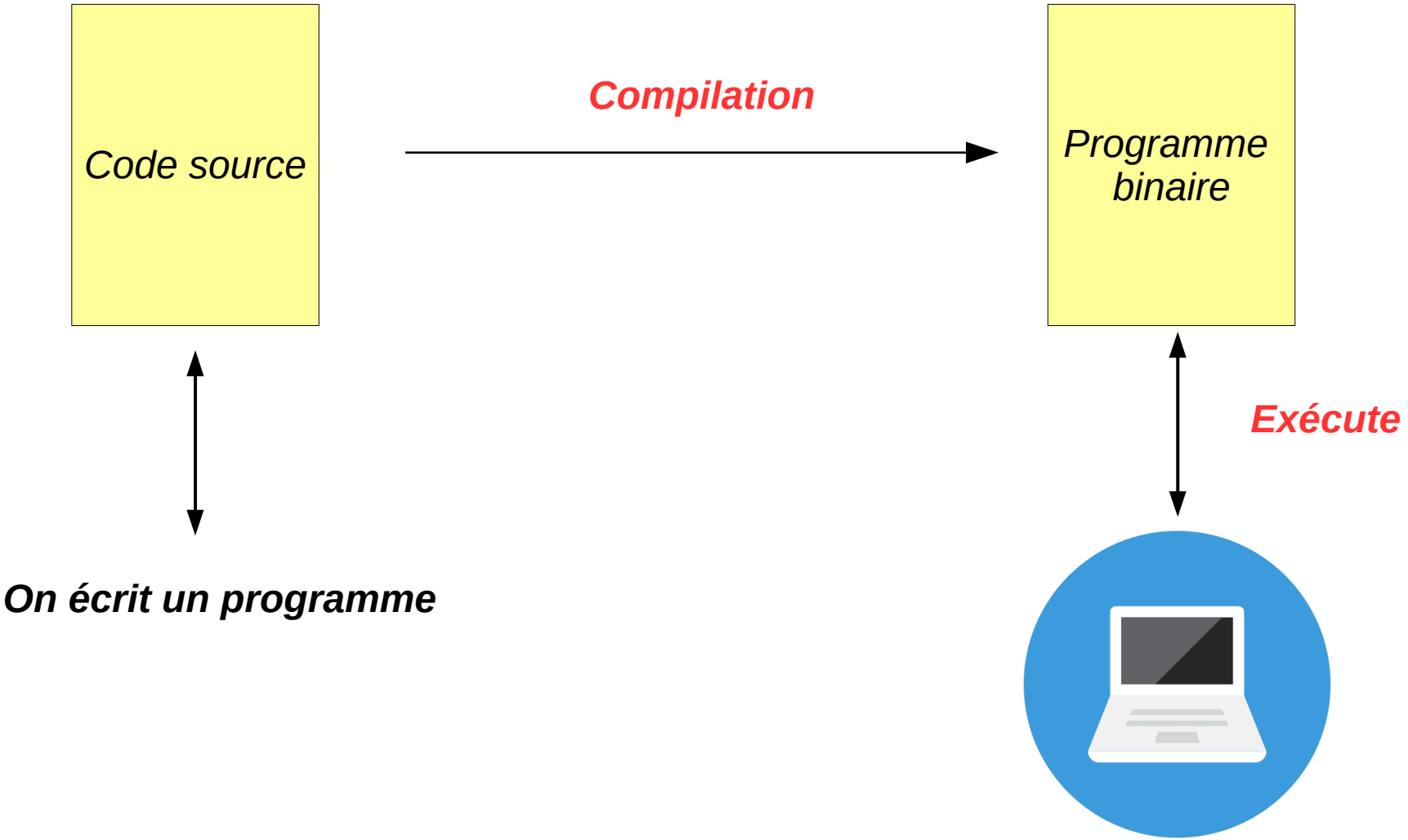
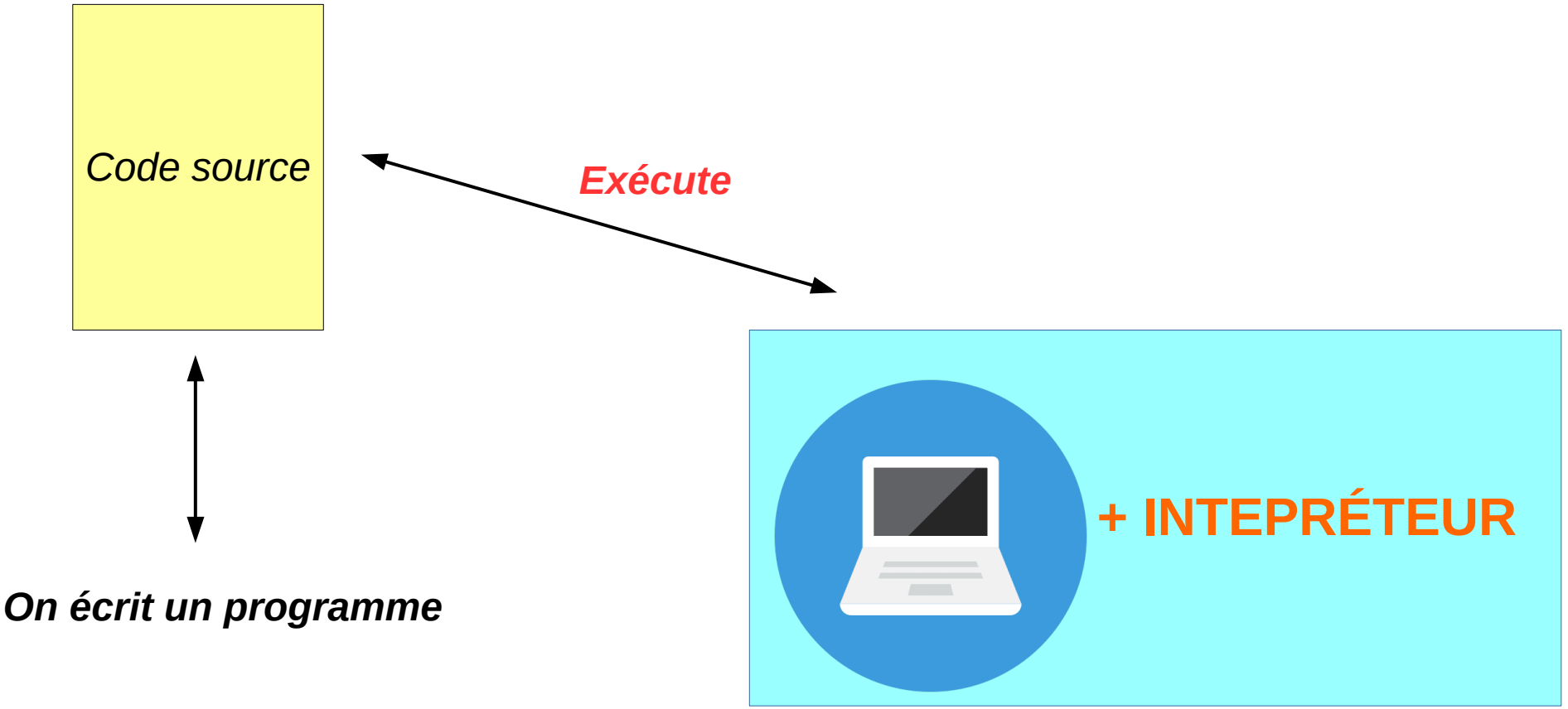


Schéma d'exécution d'un code interprété



Langage étudié

- **PYTHON 3**
- En fait un **sous langage**
  - Réalisation de calculs arithmétiques
  - Manipulation de chaînes de caractères
  - Lire/écrire/modifier des variables
  - Boucler sur des instructions
  - Tester des valeurs
  - Écrire et appeler des fonctions

Données manipulées

- Un programme manipule des données
- Ces données peuvent être de différentes sortes
- On parle en fait de **type**
  - **int** : il s'agit des données entières (par exemple : 3, 4, 5000, etc.)
  - **float** : il s'agit des nombres réels avec virgules (par exemple : 3.5, 2.4,etc)
  - **str** : il s'agit de chaînes de caractères (par exemple "Hello World", "345", "Un message !", etc)
  - On verra qu'il existe d'autre types

Quelques remarques sur les données

- Il faut éviter de mélanger les données de type différent
- Il faut toujours avoir en tête quel type de données on manipule
  - **La division entière 2 // 4 donne 0 alors que la division réelle 2 / 4 donne 0.5**
- À quoi sert le type **str** ?
  - Typiquement à stocker des données correspondant à des chaînes de caractères, mais aussi des messages que l'on souhaite afficher
- **ATTENTION :**
  - Une machine n'a pas une précision infinie
  - Ainsi, on ne peut pas compter jusqu'à l'infini
  - Pour les nombres réels, on ne dispose pas d'une précision infinie
    - Par exemple : 1 // 3 est interprété en python comme 0.3333333333333333
    - Il n'y a pas un nombre infinie de chiffres après la virgule !

Opération sur les données

- Un programme peut faire des opérations sur les données
- Sur les données entières, comme un calculatrice :
  - addition (2 + 5), soustraction, division entière (3 // 4), multipplication, etc
- Sur les chaînes de caractères : concaténation
  - "Hello" + "World !" donne la chaîne "Hello World !"
- **ATTENTION :**
  - ne mélanger pas les types dans les opérations en faisant
  - Par exemple : "Hello" + 3 ne marche pas, ni 3 * "Bob"

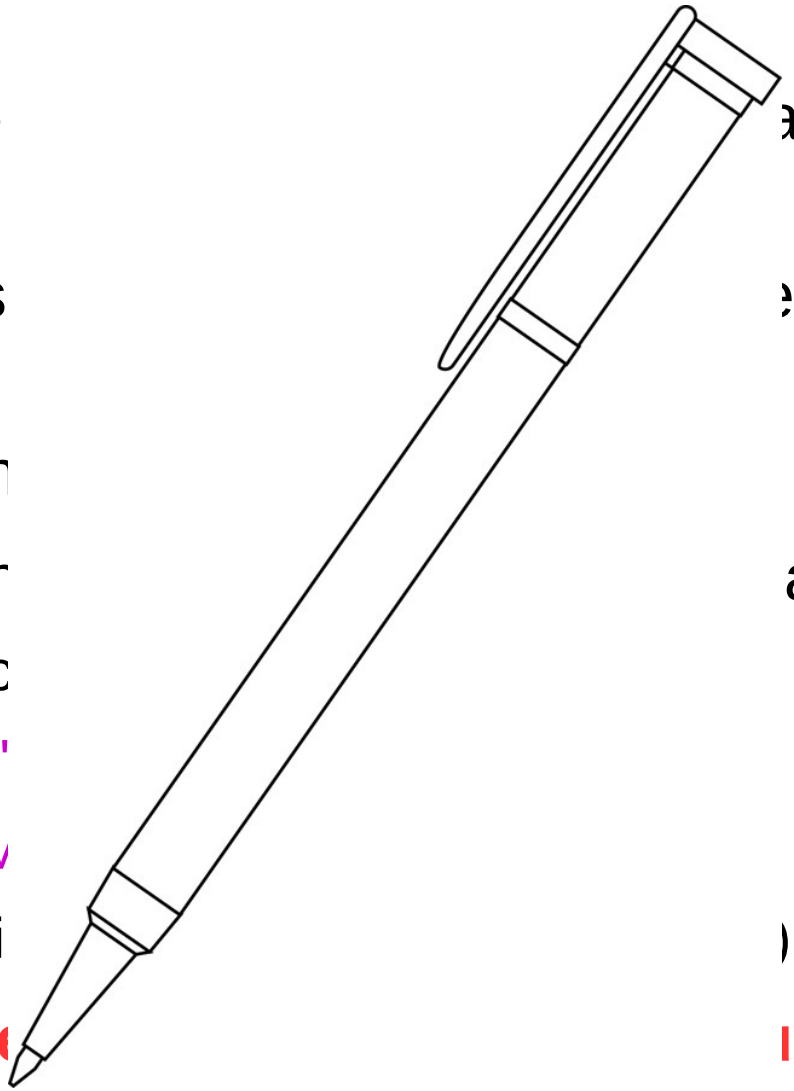
Opération sur les données

- Un programme peut faire des opérations sur les données
- Sur les données entières, comme un calculatrice :
  - addition (2 + 5), soustraction, division entière (3 // 4), multipplication, etc
- Sur les chaînes de caractères : concaténation
  - "Hello" + "World !" donne la chaîne "Hello World !"
- **ATTENTION :**
  - ne mélanger pas les types dans les opérations en faisant
  - Par exemple : "Hello" + 3 ne marche pas, ni 3 * "Bob"

Voir le résultat d'une opération

- Un programme qui fait des opérations le fait **silencieusement** (on ne voit pas l'effet)
- Pour voir le résultat d'une opération, on peut demander au programme de l'afficher
- On utilise la fonction **print**
- L'argument donné est affiché sur le terminal, par exemple
  - print (6*7) affiche 42
  - print ("Hello ! ") affiche Hello !
  - print("Un" + "Message") affiche UnMessage
  - **Attention** : print("6*7") affiche 6*7 (et pas 42)
- **Un programme n'affiche rien si on ne lui demande pas**

Voir le résultat d'une opération



- Un programme ait **silencieusement** (on ne voit pas
- Pour voir le rés eut demander au programme de
- On utilise la for
- L'argument dor al, par exemple
  - print (6*7) affic
  - print ("Hello ! "
  - print("Un" + "M
  - **Attention** : pri)
- **Un programm ui demande pas**

Les variables

- Un programme peut stocker les données
  - pour faciliter leur manipulation
  - pour abstraire leur valeur
  - pour les réutiliser plus tard
  - pour faire des calculs complexes
- Il dispose de sa mémoire (pensez à un ensemble de cases)
- Une variable indique un endroit de la mémoire où est stocké une donnée
- Une variable a un nom, par exemple x, y, z, var, z3
- Pour utiliser la variable, on utilise son nom
- Opérations sur les variables : **Affectation**, **Lecture** et **Modification**

Opération sur les variables

- **Affectation** (mettre une donnée dans une variable)
  - Attention on utilise le symbole = , mais qui ne veut pas dire égalité
  - Par exemple : x = 5 (on met 5 dans la variable x)
- **Lecture** (lire la donnée d'une variable)
  - On utilise le nom de la variable à la place de la donnée
  - Par exemple : print (x + 2) affiche 7
- **Modification** (modifier la valeur d'une variable)
  - Comme l'affectation : x = 8

Opération sur les variables

- Le programme suivant

```
x = 3
y = 2
z = x + 1
x = 6
y = 2 * x
print(x)
print(y)
print(z)
```

- Affiche

```
6
12
4
```

Quelques règles de bonne conduite

- Toujours initialiser une variable, par exemple au début du programme
- Ne pas utiliser une variable pour stocker des valeurs de type différents
  - Ex : x = 2 puis x = "Hello"
  - Beaucoup de langages n'autorise pas cela
- **Interdit** de mettre à gauche de = une valeur et à droite une variable
  - Par exemple : 2 = x

Que fait la machine ?

- Si par exemple on a une ligne z = (x * x) + 2
  1) Va chercher la valeur de la variable x
     - Si x n'a pas de valeur → **Erreur**
  2) Calcule (x * x) + 2
  3) Stocke la valeur obtenue dans la variable z
- On calcule d'abord ce qui se trouve à droite du symbole =

Exemple - I

- Le programme suivant

```
x = 3
y = 2
z = x * x
y = 3 * z
print(x)
print(y)
print(z)
```

- Affiche

```
3
27
9
```

Exemple - II

- Le programme suivant

```
x = 3
x = x + x
x = x - 1
print(x)
```

- Affiche

```
5
```

Plusieurs valeurs possibles mais un seul programme

- Une clef essentielle de la programmation est d'écrire des programmes qui vont marcher pour différentes valeurs possibles
- Par exemple :
  - Un programme qui calcule n! (factorielle de n)
  - Un programme qui calcule la somme de deux entiers a et b
  - Un programme qui calcule le pgcd de deux entiers a et b
- Ici les valeurs de n, a et b ne sont pas connus à l'avance, on sait juste qu'il s'agit d'entier

Les fonctions

- Une fonction d'un programme est une liste d'instructions
- Elle peut être appelée plusieurs fois
- Elle peut prendre des valeurs en entrée
  - Il s'agit des arguments
- Elle peut calculer une valeur et la renvoyer
  - Il s'agit de la valeur de retour
- On lui donne un nom (le nom de fonctions)
- Exemple : la fonction print( ) qui ne retourne pas de valeur mais qui affiche à l'écran

Exemple de fonctions

```
def f (x) :
        return (2 * x)

a = f(3)
b = f(5)

print (a)
print(b)
```

Définition de la fonction f

Appel de la fonction f

- Affiche

```
6
10
```

- **Si on enlève les print, le programme n'affiche rien**

Que fait la machine ?

```
def f (x) :
        return (2 * x)
```

- Si par exemple on a une ligne z = f (4)

  1) Elle remplace la valeur x de f par 4

  2) Elle calcule 2*4

  3) Elle renvoie la valeur 8

  4)Elle stocke cette valeur dans z

Exemple

```
def f (x) :
      return (2 * x)

z = 10
z = f(z)

print (z)
```

- Affiche

```
20
```

Exemple

```
def f (x) :
      x = x + 1
      return (2 * x)

x = 10
z = f(x)

print (x)
print (z)
```

ATTENTION :
Les deux x ne sont pas la même variable

- Affiche

```
10
22
```

Exemple

```
def f (x) :
        x = x + 1
        return (2 * x)

x = 10
z = f(x)

print (x)
print (z)
```

- Plus sûr en écrivant :

```
def f (y) :
        y = y + 1
        return (2 * y)

x = 10
z = f(x)

print (x)
print (z)
```

On évite ainsi les confusions possibles

Instructions conditionnelles et boucles

- Vous verrez aussi des instructions pour tester la valeur de variable

```
if (x == 5) :
        print ("AH")
else :
        print("OH")
```

Affiche AH si x vaut 5 et OH sinon

- Ou pour répéter un certain nombre de fois une instruction

```
for i in range (0,100,1) :
        print("Hello")
```

Affiche 100 fois Hello