1 Problématique

Le langage machine est un langage de bas-niveau c'est à dire qu'il (presque) directement compréhensible par la machine. Cependant le développeur doit gérer des opérations fastidieuses, sources d'erreurs.

Peut-on communiquer avec la machine dans un langage plus compréhensible pour l'Homme ?

Contexte historique

- langage haut-niveau : pas de gestion de la mémoire ; syntaxe qui s'appuie sur langage humain (anglais)
- langage compilé/interprété
- Python : van Rossum le dvp pendant ses vacances (noël 89)

2 Des constructions élémentaires

2.1 Les actions de base

Pour réaliser un programme en langage machine, nous avions besoin d'effectuer plusieurs actions :

- Enregistrer une valeur en mémoire.
- Entrée/sortie :
  - Demander une valeur à l'utilisateur.
  - Afficher une valeur à l'écran.
- Comparer deux valeurs.
- Répéter une action plusieurs fois.

Ces actions sont implémentées dans tous les langages de plus haut-niveau. Nous utiliserons le *Python*.

2.2 Les variables

Python est un *langage interprété* c'est à dire que les instructions sont traduites en langage machine *à la volée*. C'est *l'interpréteur* (figure 1) qui joue ce rôle.

```
Python 3.7.6 (default, Jan  8 2020, 19:59:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

FIGURE 1 – Console Python

Activité 1 :

1. Dans le dossier *Maths* ou *NSI* du bureau, ouvrir une console Python : *Python 3.6.8*
2. Entrer les instructions ci -après (valider après chaque ligne) :

Christophe Viroulaud

```
1 a = 12
2 b = 17
3 c = a + b
4 c
```

3. Expliquer le rôle de chaque ligne.

À retenir : Le signe = n'est pas à comprendre au sens mathématique. C'est un signe d'affectation. Ainsi l'instruction

```
1 12 = a
```

est juste mathématiquement mais ne signifie rien en Python.

4. Que renvoie l'instruction ci-après ?

```
1 c > 30
```

En première approche, nous pouvons considérer une *variable* comme une boîte qui contient une information Cette information peut être de plusieurs natures (nombre, texte, tableau...)

2.3 Entrée/sortie

L'instruction

```
1 input()
```

demande une valeur à l'utilisateur. Cependant la valeur est inaccessible. Il faut donc la stocker dans une variable en mémoire :

```
1 age = input()
```

L'instruction

```
1 print("mon texte")
```

affiche *mon texte* à l'écran. Il est possible d'afficher le contenu d'une variable :

```
1 print(age)
```

Il ne faut alors pas mettre de guillemets.

Activité 2 : Écrire un programme qui demande l'âge de l'utilisateur, calcule l'année de naissance et affiche cette année.

La valeur récupérée par *input* est une chaîne de caractère.

2.4 Utilisation d'un EDI

Il peut rapidement être fastidieux d'écrire un programme dans la console Python. Un *Environnement de Développement Intégré* permettra d'écrire plusieurs lignes de code puis se chargera d'envoyer toutes ces lignes à l'interpréteur. De plus, il sera possible d'enregistrer le programme dans un fichier.

Christophe Viroulaud

Activité 3 :

1. Dans l'espace personnel de l'ordinateur, créer un dossier *NSI* et un sous-dossier *langage*.
2. Ouvrir un EDI au choix : Spyder, Pyzo, EduPython. *NB :* Les démonstrations au tableau seront réalisées avec Spyder.
3. Écrire le programme de l'activité 2 dans la partie gauche de l'EDI.
4. Enregistrer le programme dans le dossier *NSI/langage* sour le nom *naissance.py*
5. Exécuter le programme en cliquant sur la flèche verte (figure 2) ou en appuyant sur la touche *F5*. Le code est exécuté dans la console en bas à droite.

FIGURE 2 – Exécuter un programme

2.5 Structure conditionnelle

En langage machine, il fallait comparer deux registres puis effectuer une action en fonction du résultat de la comparaison. Le comportement est similaire dans un langage de plus haut-niveau. L'instruction

```
a == b
```

renvoie *True* si les variables *a* et *b* sont égales, *False* sinon. Il faut noter l'emploi du *double égal* pour ne pas confondre avec le signe d'affectation.
L'instruction

```
if a == b:
    print(a)
```

compare *a* et *b* et affiche *a* **si a == b**.

Activité 4 :

1. Tester les codes ci-après :

```
a = 5
b = 3
if a == b:
    print("a vaut ",a)
    print("b vaut ",b)
```

```
a = 5
b = 3
if a == b:
    print("a vaut ",a)
print("b vaut ",b)
```

2. Noter les différences d'exécution.
   À retenir : L'indentation a un rôle fondamental en Python. Par convention, on indente avec *quatre espaces*. Il est possible d'utiliser la *tabulation*, l'EDI se chargeant de la convertir en espaces.

Christophe Viroulaud

3. Tester le code ci-après pour plusieurs valeurs de a et b. Bien observer l'indentation.

```
1 a = 5
2 b = 3
3 if a == b:
4     print("a et b sont égaux.")
5 else:
6     print("a et b sont différents.")
```

4. Il est possible de tester plusieurs conditions. Trouver des valeurs de a et b pour lesquelles le message « a est vraiment très grand. » est affiché.

```
1 a = 5
2 b = 3
3 if a == b:
4     print("a et b sont égaux.")
5 elif a > 10*b:
6     print("a est vraiment très grand.")
7 else:
8     print("a et b sont différents.")
```

5. Enregistrer le programme sous le nom *condition.py*

attention aux mélanges espaces / tabulation dans des codes copiés depuis le net.

2.6 Répéter une instruction

En langage machine, pour répéter des instructions il fallait créer un compteur, comparer ce compteur à la valeur limite et renvoyer le programme à une ligne précise.

2.6.1 Boucle non bornée

Une boucle *non bornée* répète une instruction *tant que* la condition est vérifiée.

Activité 5 :

1. Tester le programme ci-après :

```
1 compteur = 10
2 while compteur > 0:
3     print("Boum dans {} secondes.".format(compteur))
4     compteur = compteur - 1
5 print("Boum")
```

   Tips : Il est possible d'insérer des valeurs de variables dans du texte. Remarquer la structure de la ligne 3.

2. En anglais, que signifie *while* ?
3. À quelle ligne compare-t-on le compteur avec la valeur limite ?
4. Quel est le rôle de la ligne 4 ? Que se passera-t-il si cette ligne est retirée ?
5. Enregistrer le programme sous le nom *boucle-non-bornee.py*

Christophe Viroulaud

pour > 3.6 : print(f"Boum dans compteur secondes.")

2.6.2 Boucle bornée

Il existe une autre manière de répéter des instructions avec un mécanisme qui varie le compteur automatiquement.

Activité 6 :

1. Tester le programme ci-après :

```
1 for compteur in range(10):
2   print("Le compteur vaut {}".format(compteur))
```

2. Lire la documentation de la fonction *range* :

   https://docs.python.org/fr/3/tutorial/controlflow.html#the-range-function

3. Adapter le code précédent pour afficher :
   - Le compteur vaut 6.
   - Le compteur vaut 7.
   - Le compteur vaut 8.
   - Le compteur vaut 9.
4. Adapter le code précédent pour afficher :
   - Le compteur vaut 0.
   - Le compteur vaut 3.
   - Le compteur vaut 6.
   - Le compteur vaut 9.
5. Enregistrer le programme sous le nom *boucle-bornee.py*

Christophe Viroulaud