Présentation de l'interface et du langage Python

Si on s'intéresse à l'unité centrale d'un ordinateur, on constate rapidement qu'il s'agit d'un assemblage de composants électroniques. On peut aisément distinguer l'alimentation, le disque dur, la mémoire, le processeur... tous reliés par des câbles ou autres nappes. D'ailleurs, les informations échangées se font au travers d'un **système binaire** de signaux électriques ne pouvant prendre que deux états :

$$\begin{cases} \text{un potentiel électrique maximum correspondant à la convention 1} \\ \text{un potentiel électrique minimum correspondant à la convention 0} \end{cases}$$

Cette succession de 0 et de 1 désigne en fait un **langage de bas niveau** appelé aussi **langage machine**.

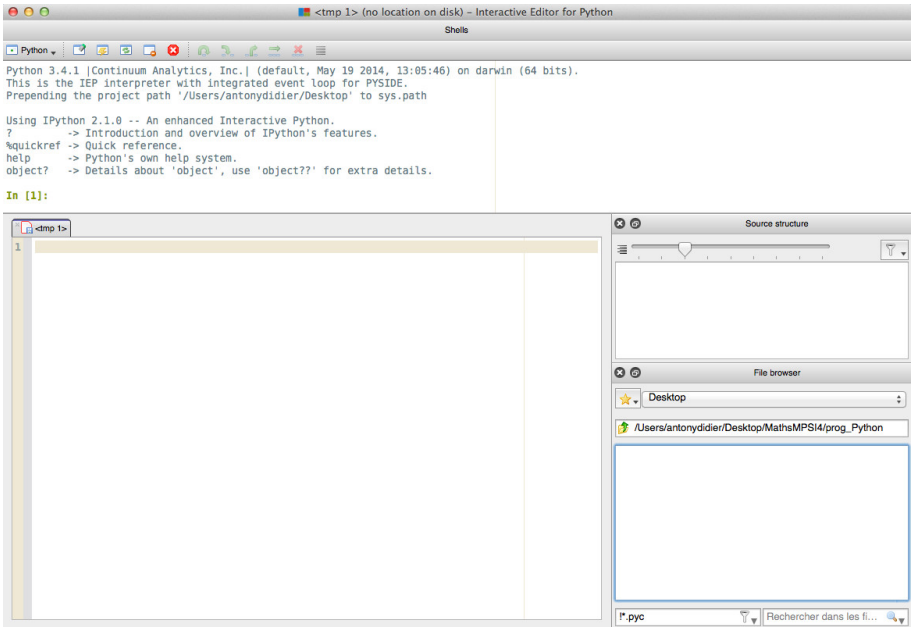
Au cours de cette année, nous n'apprendrons pas à programmer des séquences d'instructions sous ce format binaire, mais nous privilégierons le langage **Python**, un **langage de haut niveau** parmi d'autres : ce type de langage représente en fait un ensemble de mots-clefs qui définissent un code compréhensible par chacun d'entre nous et qui sera interprété par la machine avant d'être exécuté.

Si la programmation semble alors plus aisée, cela nécessite d'abord :

1. d'installer le langage Python sur la machine (par exemple sur www.anaconda.com)
2. puis d'utiliser un logiciel **interpréteur** (par exemple sur www.pyzo.org) qui nous facilitera l'édition et l'interprétation de nos instructions, et ceci afin que les tâches soient effectuées par l'ordinateur.

Présentation et premières instructions dans le langage Python

Concrètement, l'environnement de travail se présentera toujours de la façon suivante :



dans lequel on pourra distinguer :

- un module d'édition qui nous permet de construire nos différents scripts d'instructions dans le langage Python ;
- une console interactive qui permet d'exécuter certaines commandes et surtout d'appeler nos programmes afin qu'ils soient **interprétés** par le logiciel puis exécutés.

Bien entendu, les opérations usuelles sont naturellement reconnues par le langage Python, à la manière d'une calculatrice et il sera très facile d'exécuter des calculs au travers de la console :

```
In : -5+11*6
```

```
61
```

python

```
In : (7+11)*4
```

```
72
```

python

On notera quand même la différence entre la division sur les nombres réels et la division sur les nombres entiers pour laquelle le symbole // renvoie le quotient dans la division euclidienne, alors que le reste est obtenu avec le symbole %.

```
In : 10/3; 10//3; 10%3
                                        3.3333333333333335
                                        3
                                        1
```
python

De la même façon, on fera attention au calcul de la puissance d'un nombre réel :

```
In : 9**2
                                        81
```
python

De plus, la gestion des variables dans le langage Python ressemble à ce que vous avez déjà vu sur d'autres langages. On peut par exemple définir des **variables locales**, que ce soit pour définir des objets ou bien mémoriser des valeurs :

nom de la variable = `valeur`

D'ailleurs, on distinguera ce dernier symbole d'affectation de l'**évaluation booléenne** == qui teste une condition logique :

```
In : resultat = 11
```
python

```
In : resultat == 10
                                        False
```
python

On peut alors utiliser les variables définies dans la feuille de travail, mais on fera très attention à l'ordre dans lequel on les a définies :

```
In : somme(a,resultat)
                                        NameError
                                        (...)
                                        NameError : name 'a' is not defined
```
python

```
In : a = 128; somme(a,resultat)
                                        139
```
python

D'ailleurs, Python autorise la **réaffectation**, c'est à dire qu'on peut modifier la valeur donnée à l'**identifiant** :

```
In : a = a+1; a
                                        129
```
python

Pour finir, on pourra libérer le nom d'une variable grâce à la commande `del` :

```
In : del(a); a
                                        NameError
                                        (...)
                                        NameError: name 'a' is not defined
```
python

Remarque Dans le langage Python, le nom d'une variable est souvent appelé **identifiant** ou **étiquette**, en effet lorsque l'on pose `a = 128`, le logiciel stocke l'objet 128 en mémoire et c'est l'identifiant `a` qui permettra d'extraire la valeur en mémoire. Cela pourra poser quelques problèmes dans la copie de **variables structurées**.

Un langage léger avec une structure physique précise

Contrairement à d'autres langages de programmation, le langage Python est un langage léger qui ne nécessite aucune déclaration de varaibles : on définit des variables et Python identifiera lui-même leur nature. Attention néanmoins, il faudra veiller à la **structure physique** de nos programmes et les sauts de ligne ou indentation seront indispensables.

La plupart du temps, on cherchera à construire nos programmes sous la forme de **fonctions**, c'est à dire qu'à partir de données formelles appelées aussi **arguments**, le programme renverra un ou plusieurs **résultats** :

fonction : un ou plusieurs arguments $\longmapsto$ un ou plusieurs résultats

Par exemple, on souhaite construire le programme *somme* qui pour deux nombres donnés, renvoie la somme de ces nombres :

```
def somme(a,b):
    return a+b
```
python

Pour utiliser notre programme, on commence alors par interpréter celui-ci, avant de l'appeler dans la console en prenant soin de remplacer les arguments par des valeurs concrètes :

```
In : somme(2,5)
                                                    7
```

Remarque La commande `return` est très pratique pour renvoyer un ou plusieurs résultats, mais malheureusement, cette dernière commande affiche le résultat obtenu tout en interrompant le programme. Ainsi, si on souhaite afficher la valeur d'une variable sans arrêter la procédure, on sera donc amené à forcer l'affichage des valeurs à l'aide de la commande intégrée `print` :

```python
def produit(a,b):
    print(a,'x',b,'=')
    return a*b
```

Exercice 1 - Résolution d'une équation du second degré

On considère l'équation (E) $ax^2+bx+c=0$, avec $a,b,c\in\mathbb{R}$.

1. Construire la fonction *sol* qui pour tout triplet (a,b,c) donné, renvoie les solutions réelles de l'équation (E). On veillera à afficher la valeur du discriminant avant les solutions éventuelles.
2. Modifier votre programme *sol* afin que celui teste d'abord si $a=0$ avant de renvoyer les solutions éventuelles. Le programme affichera un message d'erreur si $a=0$.

Exercice 2 - Des programmes simples

Les questions sont indépendantes.

1. Construire la fonction *divise* qui pour tout couple (a,n) donné, vérifie si a divise n.
2. Construire la fonction *pair* qui teste si un entier n donné est pair.
3. Construire la fonction *prodscal* qui pour tout couple $((a,b),(c,d))$ de coordonnées, calcule le produit scalaire usuel entre deux vecteurs de coordonnées (a,b) et (c,d). On veillera à afficher l'orthogonalité des vecteurs si le produit scalaire est nul.
4. Construire la fonction *racine* qui pour tout quadruplet (a,b,c,x) donné, calcule l'image de x par le polynôme $p(x)=ax^2+bx+c$ puis précise si c'est une racine de p.

Remarque On a construit ici quelques fonctions "tests" qui renvoient `True` ou `False` : ces fonctions sont appelées des **fonctions booléennes** et elles pourront être utilisées dans d'autres programmes pour tester la condition d'un objet donné.

Exercice 3 - Définition d'une fonction mathématique

1. Comment pouvez-vous expliquer le script suivant ?

```python
def f(x):
    if x != 0:
        return 1/x
    else:
        print('problème de définition')
```

2. En vous inspirant du programme précédent, définir les fonctions f,g,h donnée par :

$$f(x)=|x^2-3x+2|\ ,\ g(x)=\frac{1}{(x-1)(x-2)}\ ,\ h(x)=\ln(1+x)$$