Structures de base d'un programme

Un programme est un texte qui décrit un algorithme que l'on souhaite faire exécuter par une machine. Ce texte est écrit dans un langage particulier, appelé langage de programmation. Il existe plusieurs milliers de langages de programmation, parmi lesquels Python, Java, C, Caml, Fortran, Cobol, etc. Il n'est cependant pas nécessaire d'apprendre ces langages les uns après les autres, car ils sont tous plus ou moins organisés autour des mêmes notions : affectation, séquence, test, boucle, fonction, etc.
Le langage principal qui sera utilisé durant ces deux années de spécialité NSI est le langage Python.
Apprendre la programmation, ce n'est pas seulement apprendre à écrire un programme, c'est aussi comprendre de quoi il est fait, comment il est fait et ce qu'il fait. Un programme est essentiellement constitué d'**expressions** et d'**instructions**. Nous introduisons dans ce cours trois instructions fondamentales que sont l'**affectation** de variables, le **test** et la **boucle**.

1. L'affectation de variables

L'essentiel du travail effectué par un programme d'ordinateur consiste à manipuler des données organisées comme un ensemble de variable.
Une variable apparaît dans un langage de programmation sous un nom de variable qui doit être explicite. Pour l'ordinateur ce nom est une référence désignant une adresse mémoire, c'est-à-dire un emplacement précis dans la mémoire vive.
Pour stocker une valeur dans une variable, on utilise une instruction appelée affectation, notée par le signe `=`. Par exemple, l'affectation `x = y + 3` est composée d'une variable `x` et d'une expression `y + 3` .

Les opérations

Opération sur des réels	
+	Addition
-	Soustraction
*	Multiplication
/	Division
//	Quotient de la division euclidienne
%	Reste de la division euclidienne
**	Puissance

Opérations booléennes	
==	Egalité
!=	Différence
<=	Infériorité ou égalité
<	Infériorité stricte
>=	Supériorité ou égalité
>	Supériorité stricte
and	Et logique
or	Ou logique

→ Le résultat d'une expression booléenne en peut prendre que deux états, vrai (True en python) ou faux (False)

Priorité des opérations

Lorsqu'il y a plus d'un opérateur dans une expression, l'ordre dans lequel les opérations doivent être effectuées dépend de règles de priorité. Sous Python, les règles de priorité sont les mêmes que celles utilisées en mathématique. On peut les mémoriser aisément à l'aide d'un "truc" mnémotechnique, l'acronyme **PEMDAS** :

- **P** pour parenthèses. Ce sont elles qui ont la plus haute priorité.
- **E** pour exposants. Les exposants sont évalués ensuite, avant les autres opérations. Ainsi **2******1+1 = 3** (et non 4), et **3*1******10 = 3** (et non 59049 !).
- **M** et **D** pour multiplication et division, qui ont la même priorité. Elles sont évaluées avant l'addition **A** et la soustraction **S**, lesquelles sont donc effectuées en dernier lieu. Ainsi **2*3-1 = 5** (plutôt que 4), et **2/3-1 = -0.3333...** (plutôt que 1.0).
- Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite.

Affectations multiples

On peut assigner une valeur à plusieurs variables simultanément. Exemple :

```
>>> x = y = 7
```

On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur :

```
>>> a, b = 4, 8.33
```

Les bonne pratique de nommage

- ✗ Un nom de variable est une séquence de lettres et de chiffres qui doit toujours commencer par une lettre.
- ✗ Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l'exception du caractère _ (souligné).
- ✗ La casse est significative (les caractères majuscules et minuscules sont distingués).
- ✗ Le nommage des variables doit être le plus explicite possible afin de simplifier la compréhension du programme
- ✗ Le nom d'une variable commence toujours par une minuscule
- ✗ On ne peut pas utiliser comme nom de variables les 33 « mots réservés » suivants :

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

2.Le test

Si nous voulons pouvoir écrire des applications véritablement utiles, il nous faut des technique permettant d'aiguiller le déroulement du programme dans différentes directions, en fonction des circonstances rencontrées.
En Python, la syntaxe d'une instruction conditionnelle est montrée dans le script à gauche. Remarquez les points suivants :

- ✗ la ligne de test commence par if, et la condition est suivie d'un deux points (indispensable, sinon Python arrêtera le script avec une erreur de syntaxe).

- ✗ les instructions 1, 2 et 3 sont celles qui seront exécutées si la condition est vraie (True)
- ✗ Ces instructions doivent être indentées (décalées vers la droite) du même nombre de tabulation pour indiquer qu'elles font partie du même if.
- ✗ La partie « else instruction4 instruction5 » est facultative. Ces instructions seront exécutées seulement si la condition qui suit le if est fausse (False)
- ✗ Le mot else doit être suivi d'un deux-points ; il doit être aligné avec le if auquel il correspond.

```
if expression :
        Instruction1
        Instruction2
        Instruction3
else :
        Instruction4
        Instruction5
```

- ✗ Lorsqu'on veut enchaîner plusieurs tests, on peut utiliser l'instruction elif, qui est la contraction de else if. Cette instruction doit être alignée avec le if correspondant, et elle doit être suivie d'une condition, puis d'un deux-points et après d'instructions indentées, comme pour un if. Les instructions seront exécutées si la condition qui suit le if est fausse, mais que la condition qui suit le elif est vraie.

3.La boucle

En programmation, on appelle boucle un système d'instructions qui permet de répéter un certain nombre de fois (voire indéfiniment) toute une série d'opérations. Python propose deux instructions particulières pour construire des boucles : l'instruction for et l'instruction while

La boucle non bornée while

Le mot while signifie « tant que » en anglais. Cette instruction répète continuellement le bloc d'instructions qui suit, tant que l'expression est vraie

```
while expression :
        Instruction1
        Instruction2
        Instruction3
Instruction4
Instruction5
```

la boucle bornée for

La syntaxe d'une boucle for est montrée à droite. i est une variable, e et e' sont des expressions et instruction1, instructions2 sont des instructions, appelée corps de cette boucle. C'est lui qui va s'exécuter autant de fois que précisé dans la boucle.
Comme dans le cas des tests, le corps d'une boucle doit être indenté.

```
for i in range(e,e') :
        Instruction1
        Instruction2
 Instruction3
Instruction4
```

Cette boucle a pour effet d'exécuter le corps de boucle (n - m) fois, où m est la valeur de l'expression e et n celle de l'expression e' . Pour chaque tour de boucle la valeur de la variable i est successivement m, m + 1, ..., n – 1.

Par exemple, exécuter la boucle :

```
for i in range(1,11):
    print("allô ",end="")
print("t'es où ?")
```

a pour effet d'afficher :

```
allô allô allô allô allô allô allô allô allô allô t'es où ?
```

4.Choisir entre une boucle for et la boucle while

Si on connaît à l'avance le nombre de répétitions à effectuer, la boucle for est toute indiquée. À l'inverse, si la décision d'arrêter la boucle ne peut s'exprimer que par un test, c'est alors la boucle while qu'il faut choisir.