TP1 - Bases de Python

1 Instructions élémentaires

Un programme Python est une suite d'**instructions**. Ce sont des ordres élémentaires que l'ordinateur est capable d'exécuter. Par défaut, les instructions sont exécutées les unes après les autres dans l'ordre.
Pour nous, il y a trois instructions de bases à connaître et à maitriser :

- **L'affectation** :

```
variable = expression
```

L'affectation s'écrit avec le signe =. À gauche du signe, on écrit un nom de **variable** (nouvelle ou déjà existante) dans laquelle on désire stocker le résultat d'un calcul ou d'une opération. À droite, on met une **expression** c'est-à-dire une formule qui a une valeur (qui peut être un nombre, une chaine de caractère ou encore un booléen). Par exemple :

```
x = 2
v = x**2 - 2*x + 5
```

Une affectation peut utiliser une variable déjà existante dans l'expression à droite et modifier ensuite cette même variable. Par exemple :

```
u = 1
u = 2*u - 5
# u vaut alors -3
```

- **La fonction print :**

```
print("Hello, World!")
```

La fonction **print** est intégrée au langage Python. Elle permet d'afficher du texte. Dans son utilisation la plus simple, on place entre parenthèse le texte à afficher :

```
print("Bonjour les ECG2 !")
```

Ce texte est une chaine de caractère et doit donc être écrit entre guillemets. On peut cependant utiliser la conversion automatique de Python pour afficher des nombres et des résultats de calculs :

```
x = 2
print(x) # Affiche "2"
```

On peut également enchainer l'affiche de plusieurs éléments en les séparant par des virgules :

```
print("La réponse est ", 42) # Affiche "La réponse est 42"
```

Exercice 1

Écrire une suite d'instructions qui calculent la valeur de $x^3 - 2x^2 + 5x - 12$ pour $x = 2$ puis affichent la valeur obtenue à l'écran.

- La fonction **input** :

```
1 nom = input("Quel est votre nom ?")
```

La fonction **input** est également intégrée au langage Python. Elle permet d'afficher du texte et d'attendre ensuite une entrée de l'utilisateur du programme. Quand l'utilisateur appuie sur la touche entrée, ce qu'il a saisie est alors renvoyé et peut être stocké dans une variable. Dans le code précédent, la variable **nom** contiendra le ce qu'entre l'utilisateur.

Exercice 2

Écrire une suite d'instructions qui demandent le nom de l'utilisateur puis affiche "Bonjour NOM !" où NOM est remplacé par le nom rentré par l'utilisateur.

2 Les types de variables

Les variables ont déjà été évoquées dans la section précédente. Les variables sont des zones dans la mémoire de l'ordinateur qui permettent de stocker des données. On peut alors les réutiliser, par exemple dans un calcul ultérieur.
Mais pour pouvoir faire ces calculs, les données dans ces variables ont des types. Considérez les deux morceaux de code suivants :

```
1 a = 3
b = 5
c = a + b
print(c)
```

et :

```
1 a = "3"
b = "5"
c = a + b
print(c)
```

Ces deux codes semblent faire essentiellement la même chose : on met du contenu dans deux variables nommées `a` et `b`. On calcule ensuite le résultat de l'opération `+` et on affiche le résultat.
Mais leurs comportements sont pourtant différents :

```
1 a = 3
b = 5
c = a + b
print(c) # affiche "8"
```

et :

```
1 a = "3"
b = "5"
c = a + b
print(c) # affiche "35"
```

Que se passe-t-il ?
Dans le premier code, `3` et `5` sont des *nombres* (plus préciséments des entiers) et donc l'opération `+` fait ce qui semble naturel : l'addition des deux nombres.
Dans le second code, `"3"` et `"5"` sont des *chaines de caractères*. L'opération `+` ne peut donc pas faire l'addition [1] mais fait une autre opération : la concaténation de chaines (cela veut dire que l'on met les deux chaines bout à bout).

1. Que voudrait dire par exemple l'*addition* "Bonjour" + "Au revoir" ?

Cette année vous allez manipuler principalement quatre types de variables différents. Dans ce sujet, nous en abordons trois, le dernier sera vu plus tard :

- **Les entiers.** Comme le nom l'indique, il s'agit de nombres entiers. Voici les opérations à connaître à leur sujet :

```
a = 3 + 5 # a contient alors 8, le résultat de l'addition
b = 3 - 5 # b contient alors -2, le résultat de la soustraction
c = 3 * 5 # c contient alors 15, le résultat de la multiplication
d = 5 // 3 # d contient alors 1, le quotient de la division euclidienne
e = 5 \% 3 # e contient alors 2, le reste de la division euclidienne
f = 5**3 # f contient alors 125, l'élévation de 5 à la puissance 3
```

- **Les flottants.** Il s'agit encore de nombres mais à virgules. On ne rentrera pas dans le détails, mais il faut se les représenter comme des nombres réels. Pour écrire un flottant, il suffit d'écrire explicitement la virgule (avec un point comme les anglo-saxons). Par exemple, dans :

```
a = 3.0 * 2.5
```

`3.0` et `2.5` sont des nombres flottants.

Les opérations sur les flottants sont similaires aux opérations sur les entiers, avec quelques nuances :

```
a = 3.0 + 2.5 # a contient alors 5.5, le résultat de l'addition
b = 3.0 - 2.5 # b contient alors 0.5, le résultat de la soustraction
c = 3.0 * 2.5 # c contient alors 7.5, le résultat de la multiplication
d = 3.0 / 2.5 # d contient alors 1.2, le résultat de la division (réelle)
e = 2.5**3.0 # e contient alors 15.625, l'élévation de 2.5 à la puissance 3.0
```

- **Les chaines de caractères.** Un caractère est un symbole qui peut être affiché à l'écran. Une chaine de caractère est donc une suite de tels symboles. On les notes en utilisant des guillements :

```
phrase = "Quelle belle journée aujourd'hui!"
```

Remarquez que les chiffres (de 0 à 9) ont dans la vie de tous les jours un double rôle : il sont à la fois des nombres et des caractères. Mais *Python* fait la distinction. Ainsi `8` représente *le nombre* 8 et `"8"` représente la chaîne de caractère qui contient l'unique caractère 8.

Il y a une opération à retenir sur les chaînes de caractères : *la concaténation* en utilisant le symbole `+`. Par exemple :

```
chaine = "Tralala" + "lalère" # chaine contient alors "Tralalalalère"
```

Il est utile [2] de convertir d'un type à l'autre. Pour cela, il suffit d'utiliser le nom du type comme dans les exemples suivants :

```
a = int("39") # a contient alors le nombre entier 39
b = float(3) # b contient alors le flottant 3.0
c = str(54) # c contient alors la chaine "54"
d = float("2.7") # d contient alors la chaine "2.7"
```

Exercice 3

Essayez de faire la conversion `int(-2.3)` *et la conversion* `int(3.5)`. *Que constatez-vous ? Pourquoi à votre avis ?*

Exercice 4

Essayez de faire la conversion `int("bonjour")`. *Que se passe-t-il ? Pourquoi ?*

2. et c'est même pourquoi nous faisons toute cette digression sur les types

Exercice 5

La fonction **input** *renvoie une chaine de caractère. En utilisant les conversions de types, écrire un code qui demande à l'utilisateur une valeur de x puis calcule la valeur de $x^3 - 2x^2 + 5x - 12$ et l'affiche à l'écran.*

3 Les fonctions

La langage Python permet de définir des fonctions. En informatique et plus particulièrement en Python, les fonctions sont des suites d'instructions que l'on rassemble et auxquelles on donne un nom afin de pouvoir les réutiliser facilement.

Une fonction commence par un en-tête de la forme suivante :

```
1 def nom_de_la_fonction(parametres):
```

Le mot **def** est un **mot-clef** du langage. Il signale à l'interpréteur Python que l'on commence la déclaration d'une fonction.

Après le mot-clef **def**, on place le nom que l'on désire donner à la fonction. Ce nom doit un être un identifiant valable (comme les variables) et peut contenir des lettres majuscules ou miniscules (non accentuées), des chiffres ou encore le symbole underscore _. Le nom doit commencer par une lettre ou par _. Éviter cependant de commencer par _, cela est souvent utilisé à des fins particulières.

Un identifiant ne peut pas être un mot-clef, puisque ces mots sont déjà utilisés par le langage. Il peut cependant être aussi long qu'on le souhaite. Inutile d'en mettre des tartines, mais cela vous permet de mettre un nom clair qui explicite le rôle de la fonction.

Enfin, Python fait la différence entre les majuscules et les miniscules [3]. Les fonctions **ma_fonction** et **Ma_Fonction** ne sont donc pas les mêmes.

Après le nom de la fonction, on met entre parenthèses une liste de paramètres de la fonction, séparés par des virgules. Ce sont des noms de variables qui vont être disponibles au sein de la fonction pour exécuter ses instructions.

Finalement, la ligne se finit par deux points **:**. Ce symbole indique le début d'un bloc de code qui **doit** être indenté. Cela signifie que toutes les instructions de la fonction doivent être décalées vers la droite avec un nombre constant d'espaces ou de tabulations. C'est comme cela que Python sait quelles instructions font partie de la fonction ou non. Lorsque vous arrêtez d'indenter votre code, Python comprendra cela comme la fin de la fonction.

Il y a une instruction spéciale pour les fonctions, l'instruction **return** qui s'utilise ainsi :

```
1 return expression
```

Lorsque Python rencontre l'instruction **return**, l'exécution de la fonction s'**arrête** et le résultat de l'expression est renvoyé à l'appelant. Cela permet donc d'utiliser les fonctions comme dans des expressions. Par exemple :

```
1 def delta_degre2(a,b,c):
      delta = b**2 - 4*a*c
      return delta

5 r = delta_degre2(1,0,1)
  # r vaut -4
```

Exercice 6

Écrire une fonction qui prend en paramètres a, b, c et x et qui calcule la valeur de $P(x)$ avec $P(X) = aX^2 + bX + c$.

3. On dit que Python est sensible à la casse.