Info113 - TD1

L'objectif de ce premier TD est de se familiariser avec des éléments de base du langage *Python*. Les concepts vus dans ce TD sont :

- les constantes et les opérations usuelles pour les nombres, les *chaînes de caractères* et les *booléens*,
- la définition de variables,
- la définition et l'utilisation de fonctions simples.

Introduction

Python étant un langage interprété, il peut s'utiliser à la manière d'une calculatrice. Au lancement de *Idle*, une fenêtre contenant l'interpréteur s'ouvre :

```
Python 3.*.* ...
...
>>>
```

On peut écrire des expressions après le *prompt* (les trois symboles `>>>`). Le résultat est affiché à l'écran après l'appui sur la touche `Entrée`. Par exemple :

```
>>> 2 + 2
4
```

Si l'expression entrée n'est pas valide, l'interpréteur affiche un message d'erreur :

```
>>> 2 +
SyntaxError: invalid syntax
```

1 Les nombres entiers

Entrez quelques nombres entiers : l'interpréteur les affiche simplement à l'écran : 42, -273, ...

Les opérations habituelles sont disponibles : l'addition (notée `+`), la différence (notée `-`), la multiplication (notée `*`), la division *entière* (notée `//`) et l'exponentiation (notée `**`).

Entrez les expressions suivantes pour obtenir leurs résultats :

```
34 + 39
32 - 305
18 * 5
123 // 10
```

```
2 ** 4
```

Comme en mathématiques, il est possible de les combiner les opération et d'utiliser des parenthèses pour grouper des opérations. Les conventions sont les mêmes, et `1 + 2 * 3` signifie bien `1 + (2*3)` et non pas `(1 + 2) * 3`.

Testez en rentrant les expressions suivantes :

```
2 * 3 + 12 // 2
7 * (5 + 2)
2 * 3 * 4 * 5
(18 - 4) * 12
(2 + 4) ** 2
2 - 3 + 4 - 5 + 6
```

2 Les flottants et les opérations sur les flottants

On peut aussi manipuler des nombres à virgule, qu'on appelle des *nombres flottants* en informatique. On les écrit utilisant un point et non une virgule.

```
3.14159265359
-3.333
12.
-23.0
```

On peut aussi utiliser une notation avec exposant. Ainsi, les deux expressions suivantes sont équivalentes.

```
23456.789
2.3456789e4
```

Les opérations sont similaires à celles sur les nombres entiers, mais la division est noté avec un unique symbole `/`.

```
18.4 + 12.3
12.4 - 34.8
-3.2 * 32
24.5 / 2
2 ** 0.5
```

Quelle différence y a t'il entre les deux expressions suivantes ?

```
13 // 2
13 / 2
```

Entrez quelque expressions utilisant des nombres flottants dans l'interpréteur :

```
2.34 * (1.3 - 0.98)
4.567 * 2.2 + 18.
2.3 + 4.5 + 6.7
(2.3 * 4 - 4 * 2.3) ** 0.2
0.1 + 0.2
```

Avez-vous des remarques sur la précision des calculs ? En particulier quel est la valeur de la dernière expression donnée ci-dessus.

3 chaînes de caractères

Le langage permet aussi de manipuler du texte en utilisant des *chaînes de caractères*, c'est à dire des suites de lettres ou autres symboles (chiffres, ponctuation, etc.). Le texte doit être délimité par des guillemets hauts, simples (`'`) ou doubles (`"`). Il est important de noter qu'il sera impossible d'utiliser le caractère `'` dans une chaîne délimitée par des guillemets simple (ou l'inverse).

```
""
"Python c'est trop bien!"
''
'Blablabla'
```

Les trois opérations de base sur les chaînes de caractères sont : la *concaténation*, la répétition, l'accès au caractère à une position donnée, et la taille.

```
"Py" + "thon"
"bla" * 2
"Blabla"[2]
len("Blabla")
```

Entrez quelque expressions utilisant ces opérations dans l'interpréteur. Quel est la position du premier caractère d'une chaîne ? Que se passe-t-il si on accède à un caractère situé au delà de la chaîne ?

Quelle est la différence entre les deux expressions ci-dessous ?

```
"12"
12
```

4 Valeurs booléennes

On peut comparer des nombres avec les relations habituelles. Les symboles pour « strictement plus petit » et « strictement plus grand » sont `<` et `>`. Pour « plus petit ou égal » et « plus grand ou égal », il suffit de coller le signe `=` : `<=` et `>=`.

Pour tester si deux nombres sont égaux, le symbole est `==` et pour tester s'il sont différent, le symbole est `!=`.

Tester ces relations avec

```
12 < 18
23.4 > 54
1 <= 1
42 >= 0
```

```
23 == 223
12 != 21
```

Attention, le symbole = tout seul ne permet pas de comparer deux nombres.

```
>>> 3 = 5
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

On peut composer ces relation avec : la conjonction (`and`), la disjonction (`or`) et la négation (`not`).

```
not 1 > 2
1 < 10 and 10 < 100
not False
```

Exercice : à quoi correspondent ces comparaisons sur les chaînes de caractères ?

5 Variables et affectation

Il peut être laborieux d'écrire de longues expressions, surtout quand elles contiennent plusieurs occurrences d'une même sous-expression. Par exemple, pour calculer la valeur de $x^2 - x + 1$ pour $x = 12 \times 3 + 18$.

```
(12 * 3 + 18) ** 2 - (12 * 3 + 18) + 1
```

Pour simplifier l'écriture de cette expression, on peut définir une variable x avec la valeur appropriée, puis faire le calcul avec cette variable :

```
>>> x = 12 * 3 + 18
>>> x**2 - x + 1
2863
```

Utilisez des variables pour écrire quelque expressions complexes.

```
x = 2 * 3 + 18
y = 328
z = x - y
z ** 2 + x * y

s1 = "Bla"
s2 = "bla"
s1 + s2 * 9
```

On peut modifier la valeur d'une variable en lui affectant une nouvelle valeur. L'ancienne valeur est alors perdue... Notez que l'on peut quand même utiliser l'ancienne valeur pour définir la nouvelle valeur. Ainsi, pour augmente la valeur d'une variable de 1, on peut écrire

```
x = x+1
```

Testez les instructions suivantes :

```
x = 1
```

```
x
x = x/3
x
x = x/3
x
x = x/3
```

Que se passe-t-il si on utilise une variable pas encore définie dans une expression ?

Exercice 1 : définissez une variable `x` dans l'interpréteur avec une valeur flottante de votre choix, puis écrivez un calcul qui permet de trouver directement sa valeur absolue.

Indice 1 : vous pourriez avoir besoin de calculer la racine carrée d'un nombre.
Indice 2 : on peut utiliser l'exponentiation pour calculer une racine carrée.

Exercice 2 : définissez une variable `msg` contenant une chaîne de caractère, puis une variable `l` contenant un entier dans l'interpréteur. Trouvez maintenant un moyen d'obtenir une chaîne de caractère contenant autant de répétition de `msg` qu'il faut pour avoir au moins `l` caractères.

Indice : la division entière peut être utile.

6 Définition de fonctions simples

Que se passe-t-il maintenant si on veux recommencer les exercices précédents avec des paramètres `x`, `msg` et `l` différents ? On doit donner une nouvelle valeurs aux variables, puis de réécrire exactement le même calcul. Pour éviter de recopier le calcul, on peut définir des *fonctions*. Pac exemple, pour reprendre le calcul du début de la partie précédente :

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

f(12 * 3 + 18)
```

Pour définir une fonction, on utilise le mot-clé `def` suivi du nom de la fonction (`f` dans l'exemple ci-dessus), d'une liste de noms d'arguments entre parenthèses séparés par des virgules, et enfin le symbole `:`. On donne ensuite la valeur de retour de la fonction avec le mot-clé `return` suivi d'une expression qui peut contenir les noms des arguments de la fonction (ils sont en fait des variables).

Remarquez que pour utiliser une fonction, on utilise simplement son nom, suivi de la liste des valeurs pour les arguments. Nous avons en fait déjà utilisé une application de fonction plus haut, laquelle ?

Exercice 3 : reformulez vos réponses aux deux exercices précédents en définissant des fonctions, puis en les utilisant avec différentes valeurs.

Exercice 4 : définissez une fonction `moyenne2` calculant la moyenne entre deux nombres, puis des fonctions `moyenne3` et `moyenne4` calculant la moyenne entre trois et quatre nombres.

Exercice 5 : définissez des fonction `lettre_debut`, `lettre_fin` et `lettre_centre` qui prennent un argument contenant une chaîne non vide, et retrournent sa première lettre, dernière lettre, et lettre centrale respéctivement.