TP1

Partie 1 — Expressions

Ex.1 — Interpréteur et arithmétique

Dans ce cours vous apprendrez le langage Python : un langage moderne, facile d'accès et très utilisé. Vous travaillerez dans pycharm : une IDE (*Integrated Development Environment*) pour Python, c'est-à-dire un éditeur spécialisé dans la rédaction de programmes python, leur exécution, débogage, etc.

Question 1 : Parcourir rapidement Mettre la Documentation Python 3 dans vos signets. Faire le Quick tour.

Note : si vous ne comprenez pas l'anglais… il faut s'y mettre, il est omniprésent dans le monde de la programmation, de l'ingénierie et des Sciences en général.

Python est un langage interprété : pas besoin, donc, d'écrire le programme complet et de le compiler pour pouvoir enfin l'exécuter; on peut écrire et faire exécuter du python ligne par ligne.

Question 2 : Ouvrir la console python en bas à gauche de pycharm. Ecrire

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

puis entrée.

Noter que `2+2` fait partie du langage python. En appuyant sur entrée vous transmettez ce fragment de code python à l'interpréteur, qui l'évalue et renvoie 4.

Il existe probablement plusieurs versions de python sur votre machine, il faut s'assurez que vous utilisez la bonne.

Question 3 : Lorsque vous avez ouvert la console, à la question précédente, la première chaîne de caractère qui s'est affiché indiquait le chemin d'accès de l'interpréteur python que vous utilisez. Assurez vous qu'il s'agisse bien du Python 3 fourni par Anaconda3. Sinon revoir a fin de *AMETICE > L1MEO > Pour lancer vos TPs* pour arranger ça, et relancer la console.

Question 4 Exécuter dans la console :

```
>>> 7+3*5
>>> (7+3)*5
>>> 20/7
>>> 20//7
>>> 20 % 7
>>> 2,5/5
>>> 2.5/5
```

Noter les priorités entre opérateurs: `*` a précédence sur `+`.

Noter que `/` est la division, `//` est la division entière, et `%` est le reste de la division entière.

Noter que le séparateur décimal est le point, et non pas la virgule, car celle-ci est déjà utilisée pour séparer les éléments d'une liste.

Correction

```
# TODO : Utiliser le shell et la console, en cliquant en bas à gauche, pour cet exercice.
```

Ex. 2 — Variables and types

Variables

En python les variables sont comme des étiquettes que l'on pose sur les objets, pour leur donner un nom.

Un nom de variable est : * une séquence de lettres (a → z , A → Z) et de chiffres (0 → 9), qui commence par une lettre. * Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux etc. sont interdits, à l'exception de l'underscore _. * Les lettres majuscules et minuscules sont distingués: `Joseph`, `joseph`, `JOSEPH` sont donc des variables différentes. * Choisissez des noms de variables clairs, significatifs.

L'instruction python

```
code=4287
```

pose l'étiquette `code` sur l'objet `4287`. Désormais, `code` se verra affectée la valeur `4287`. Cette instruction est une *affectation*. On décide que `code` vaut `4287`, c'est une affirmation, pas une question.

A l'inverse, l'instruction python

```
code==4287
```

interroge l'interpréteur sur le fait que `code` vaille `4287`, ou pas. La réponse peut être `True` (vrai) ou `False` (faux). Cette instruction de modifie pas la valeur de `code`. C'est une question, pas une affirmation.

Question 1 : Affecter les valeurs `7.692` et `21.8` aux variables `a` et `b`. Calculer leur somme dans une variable nommée `resultat` et affichez le contenu de cette dernière.

Question 2 : On cherche à calculer la valeur d'un téléphone valant 999 USD en euros. Affecter cette valeur à une variable `telephone_usd`. En supposant que 1 USD = 0.9 EUR, calculer la conversion dans une variable

nommée `telephone_eur`. En utilisant les flèches haut-bas, changer la valeur de `telephone_usd` à 888 USD et recalculer `telephone_eur`.

Types

Combien valent 3 chaussettes plus 4 caleçons?

La question est déconcertante, peut-être est-elle mal posée, il est parfois difficile de mélanger ainsi des objets de différents types.

En python, les objets ont des types : * **int** : désigne un entier * **float** : désigne un nombre décimal * **str** : désigne une chaîne de caractères On peut consulter le type d'un objet à l'aide de la fonction `type(.)`.

Question 3 Exécuter, dans la console:

```
>>> type(2)
>>> a=2
>>> type(a)
>>> type(10 // 3)
>>> type(10 / 3)
>>> type(2.5)
>>> a=2.5
>>> type(a)
>>> type(2 + 2.5)
>>> type(3 * 1.0)
>>> type(3.14 * 1)
>>> type("Bonjour")
>>> type("""Aurevoir""")
>>> type('!!!')
>>> type("2")
>>> a="2"
>>> type(a)
```

Noter que les variables ne sont pas d'un type figé en python, car ce sont simplement des étiquettes que l'on pose sur des objets, qui eux ont un type figé.

Noter que les trois types de guillemets qui permettent de former des chaînes de caractère en python.

Noter que python accepte de faire des opérations mélangeant `int` et `float`, et que cela donne un objet de type `float`. Cela a du sens.

Par contre, additionner des nombres à des chaînes de caractère n'a pas de sens, et python ne vous signale alors une erreur.

Question 4 Exécuter, dans la console:

```
>>> 2+"hello" #Erreur de type
>>> 2+"2" #Erreur de type
```

```
>>> nombre=2
>>> mot="hello"
>>> nombre+mot #Erreur de type
>>> nombre+nombre
>>> mot+mot
```

Remarquer que `+` est, tantôt une opération sur les nombres, tantôt une opération sur les chaînes de caractères, mais n'accepte pas de mélange des deux.

Remarquer en effet que `2` n'a rien à voir avec `"2"` car l'un est un entier, tandis que l'autre est une chaîne de caractères.

Toutefois, l'un peut être converti en l'autre si on le souhaite vraiment:

Question 5 Exécuter, dans la console:

```
>>> 2+int("2")
>>> str(2)+"2"
```

Question 5 Exécuter, dans la console:

```
>>> x=2
>>> x+x
>>> x="hello"
>>> x+x
```

Remarquer que python ne peut pas savoir, en voyant juste la ligne `x+x`, si l'on est en train de lui demander de faire le `+` sur les entiers ou bien le `+` sur les chaînes de caractères. Il a besoin pour ça de connaître le type de `x`, mais celui-ci peut varier au cours du programme. Le comportement de `x+x` peut donc varier aussi au cours du programme; le terme technique pour désigner ceci est le "typage dynamique".

Correction

```
# Cet exercice se fait dans la console python.
```

Ex. 3 — Manipulations & affichages

Question 1 : Exécuter dans la console :

```
>>> print("hello")
>>> mot="hello"
>>> print(mot)
>>> print(mot+mot)
>>> print(mot+" "+mot)
```

```
>>> print(mot,mot)
>>> print(2)
>>> nombre=2
>>> print(nombre)
>>> print(nombre+nombre)
>>> print(nombre,nombre)
>>> print(mot+nombre) #Erreur de type
>>> print(mot,nombre)
```

Question 2 : * Définir deux variables : `a` et `b` ayant pour valeurs respectivement `5` et `15`. * En une ligne de code, afficher la phrase : "`a vaut 5 et b  vaut 15, leur somme fait 20`"—sans tricher: en utilisant les valeurs des variables à ce moment là. * Changer `b` en `10` et réexécuter la ligne, pour voir si l'affirmation de la phrase reste correcte.

Question 3 : * En utilisant l'opérateur `%`, et en une ligne de code, afficher la phrase : "`That b is a multiple of a is True`"—sans tricher: en utilisant les valeurs des variables à ce moment là. * Changer `b` en `13` et réexécuter la ligne, pour voir si l'affirmation de la phrase reste correcte.

Question 4 : Écrivez les lignes de code permettant d'échanger les valeurs de `a` et de `b`. Sans tricher: vos lignes de codes ne doivent pas dépendre des valeurs spécifique que contiennent `a` et `b` initialement. Réexécuter la ligne de la question 2 pour voir si ça a marché.

Question 5 : A la question 4, est-ce que vous avez eu besoin d'une troisième variable? Ca n'était pas indispensable. Utilisez, soit votre esprit mathématique, soit votre esprit geek, pour vous en dispenser.

Correction

```
# Cet exercice se fait dans la console python.
```

Partie 2 — Programmes

Ex. 4

Un programme est une combinaison d'instructions, tout comme dans une recette de cuisine. Les blocs instructions qui se suivent forment une séquence; l'interpreteur prendra ce bloc et les exécutera l'une après l'autre.

Vous écrirez vos programme dans le fichier `task.py`, dans le pannel du milieu.

"L'état normal d'un programme, c'est de bugger." —Gérard Berry, Professeur d'informatique au Collège de France.

Le remède, c'est de tester votre programme à chaque fois que vous le modifiez. Pour ça, clique droit sur `task.py`, et `Run`.

Question 1 : Écrire un programme qui affecte les variables `note1`, `note2`, `note3` avec les valeurs `15.5`, `12.75` et `14.25` respectivement. Ensuite, calculer la moyenne de ces notes et stocker le résultat dans une variable nommée `moyenne`. Affichez la valeur de la variable `moyenne` précédée du message : "La moyenne des trois notes est".

Question 2 : Si 15 étudiants utilisent 5 ordinateurs, combien utiliseront d'ordinateurs 90 étudiants ? Écrire un programme contenant des variables bien nommées avec les données du problème, puis calculant le résultat dans une variable et l'affichant.

Quand vous aurez terminé, appuyer sur le bouton `Check` du pannel de droite. Cela vous indiquera si votre réponse est exactement celle que nous attendions. Si votre réponse diffère mais qu'elle marche et reste élégante, vous pouvez la conserver.

Une fois le bouton `Check` appuyé, si vous bloquez, vous pouvez consulter la solution en appuyant sur Peek…

Correction

```
#Cet exercice se fait en modifiant ce fichier.
#1. Ne modifiez que l'intérieur les zones marquées
#   (les "placeholders").
#2. Exécutez votre programme à chaque fois que vous y ajoutez une ligne de code !!!
# Pour ça :
#   - Clique droit sur task.py
#   - Run
#   - La fenêtre d'exécution remplace la console en bas.
#3. Quand vous aurez terminé, appuyer sur le bouton Check du pannel de droite.
# Cela vous indiquera si votre réponse est exactement celle que nous attendions.
#4. Une fois le bouton Check appuyé, si vous bloquez, vous pouvez consulter la solution en a

#Q1
note1 = 15.5
note2 = 12.75
note3 = 14.25
moyenne = (note1 + note2 + note3) / 3
print("La moyenne des trois notes est : ", moyenne)

#Q2
etudiants = 15
ordinateurs = 5
```

```
ratio = ordinateurs / etudiants
resultat = 90 * ratio
print(resultat)
```