TP2 : de l'algorithme au programme Python

Objectif : maitriser les notions de variable, affectation et les tests conditionnels

N.B Vous allez réaliser ce TP sous Pycharm, donc prenez le temps de lire le guide présent sur moodle

Exercice 1 : variables/affectation

Ecrire en Python un programme vous permettant de permuter les valeurs de deux variables contenant des entiers.

Exemple d'exécution :

```
Entrez x : 1
Entrez y : 2

Avant permutation:
x : 1
y : 2

Après permutation:
x : 2
y : 1
```

- **Il faut bien respecter l'affichage donné dans l'exemple !**

Exercice 2 : Année de naissance

Ecrivez un programme `age.py` qui :

- Demande l'âge de l'utilisateur ;
- Lit la réponse de l'utilisateur et l'enregistre dans une variable `age` de type entier ;
- Calcule l'année de naissance (à un an près) de l'utilisateur et l'enregistre dans la variable `annee` de type entier ;
- Affiche l'année de naissance ainsi calculée.

Exemple d'exécution du programme :

```
Donnez votre age :
20
Votre annee de naissance est : 2005
```

Exercice 3 : Echange de trois valeurs

Récupérer le programme `TP2_swap.py` fourni sur moodle. Ce programme a pour but de demander à l'utilisateur d'entrer trois nombres et de les afficher. Il doit ensuite les permuter et les afficher à nouveau. Le code pour effectuer la permutation est manquant.
Il vous est demandé de compléter ce programme (entre les commentaires comme indiqué dans le code) par le code approprié pour réaliser la permutation suivante : le contenu de '*a*' doit aller dans '*b*', celui de '*b*' dans '*c*' et celui de '*c*' dans '*a*'.

Voici un exemple de déroulement :

```
Entrez la premiere valeur : 51
Entrez la deuxieme valeur : 876
Entrez la troisieme valeur : 235
Les valeurs entrees sont : a = 51, b = 876 et c = 235
Permutation: a ==> b, b ==> c, c ==> a
Les valeurs permutees sont : a = 235, b = 51 et c = 876
```

Exercice 4 : Fondue

Le but de cet exercice est d'écrire un programme qui permet d'adapter automatiquement, en fonction du nombre de convives, les quantités d'ingrédients nécessaires à la confection d'une fondue fribourgeoise.

Ecrivez un programme `Fondue.py` qui :

1. Déclare une constante `BASE`, avec la valeur 4, et qui indique le nombre de personnes pour laquelle est conçue la recette de base ;
2. Déclare une variable `fromage`, initialisée à 800.0, qui donne la quantité de fromage en grammes nécessaire pour `BASE` personnes ;
3. Déclare une variable `eau`, initialisée à 2, qui donne la quantité d'eau en décilitres nécessaire pour `BASE` personnes ;
4. Déclare une variable `ail`, initialisée à 2, qui donne le nombre de gousses d'ail nécessaires pour `BASE` personnes;
5. Déclare une variable `pain`, initialisée à 400, qui donne la quantité de pain en grammes nécessaire pour `BASE` personnes ;
6. Demande à l'utilisateur d'introduire le nombre de convives pour lequel on veut préparer la recette ;
7. Lit la réponse de l'utilisateur et l'enregistre dans une variable `nbConvives` de type entier ;
8. Adapte les quantités de chaque ingrédient en faisant une règle de trois
   `(nouvelleQuantite = quantiteDeBase * nbConvives / BASE ) ;`
9. Affiche la recette pour le nombre de convives voulus selon l'exemple ci-dessous :

Exemple d'exécution du programme :

```
Entrez le nombre de personne(s) conviées à la fondue : 3
Pour faire une fondue fribourgeoise pour 3 personnes, il vous
faut :
- 600.0 gr de fromage
- 1.5 dl d'eau
- 1.5 gousse(s) d'ail
- 300.0 gr de pain
```

N.B. En Python, il n'y a pas de moyen de déclarer des constantes qui ne sont pas modifiables après. Pour cela et par convention, les constantes seront tout simplement déclarées avec des noms en majuscule pour les différencier des variables.

Exercice 5 : expressions conditionnelles

Rappel

Le langage python permet comme tout autre langage de définir des blocs d'instructions liées à une même action – on parle également d'une instruction composée. On les retrouve partout dans les programmes, définition d'une fonction, structures de contrôle, etc.

Les blocs sont toujours définis de la même manière :

```
Ligne d'en-tête:
        première instruction du bloc
        deuxième instruction du bloc
        ... ..
        dernière instruction du bloc
```

La ligne d'entête se termine par « **:** » pour indiquer à python qu'il débute un bloc d'instructions
Les instructions qui le composent sont alors repérées par l'**indentation** – pas de bloc d'accolades comme dans d'autre langage. L'indentation doit être homogène et rigoureuse, soit des espaces, soit des tabulations mais pas les deux. **On choisira dans la suite des TP d'utiliser les tabulations.**

Dans un programme, il est souvent nécessaire d'orienter son déroulement vers l'exécution d'un bloc ou d'un autre. Pour cela, des instructions testent des conditions et modifient le comportement du programme en conséquence. L'instruction « **if** » est l'instruction conditionnelle de base.

```
if expression_logique :
        instruction 1
        instruction 2
        .....
        instruction n
else :
        autre bloc d'instruction
```

Si « **expression_logique** » est vérifiée, le bloc contenant « **instruction 1**, **instruction 2**, …, **instruction n** » est exécuté, sinon, il est ignoré et le bloc « **else** », qui est facultatif est exécuté.

Il faut parfois effectuer des tests avec plusieurs alternatives. On peut alors utiliser l'instruction « **elif expression_logique :** » pour définir un nouveau test propre à chaque alternative

```
if expression_logique1 :
        instruction bloc1
elif expression_logique2 :
        instruction bloc2
else :
        instruction bloc3
```

Si « **expression_logique1** » est vérifiée, on exécute les instructions du **bloc 1**, sinon si «**expression_logique2** » est vérifié, on exécute les instructions du **bloc 2**, sinon on exécute les instructions du **bloc 3**.

Travail à faire :

Ecrivez un programme Python qui lit un nombre et indique s'il est positif, négatif ou s'il vaut zéro et s'il est pair ou impair.

Exemple d'exécution :

```
Entrez un nombre entier: 5
Le nombre est positif et impair
Entrez un nombre entier: -4
Le nombre est négatif et pair
Entrez un nombre entier: 0
Le nombre est zéro (et il est pair)
```

Exercice 6 :

Ecrire un script nommé « **tp2exo6.py** » qui simule le lancer d'une pièce avec un nombre aléatoire entre 0 et 100, si le nombre est plus petit que 50, le résultat à afficher est « Pile !» sinon c'est « Face !».

Exercice 7 :

Ecrire un script nommé « **tp2exo7.py** » qui simule une pièce truquée qui donnera « Pile » en moyenne deux fois sur trois.

Exercice 8 : intervalle

Soit l'intervalle I composé de l'union des intervalles [2,3[et]0,1] et [-10,-2] :
I = [2,3[U]0,1] U [-10,-2] avec I représentant des valeurs réels.

Écrivez le programme Intervalle.py qui :

1. Demande à l'utilisateur d'entrer un réel ;
2. Enregistre la réponse de l'utilisateur dans une variable x de type réel ;
3. Teste l'appartenance de x à l'ensemble I et affiche le message «x appartient à I» si c'est le cas, et «x n'appartient pas à I» dans le cas contraire. **Votre test doit utiliser uniquement les opérateurs relationnels < et ==.** Tous les opérateurs logiques sont, par contre, autorisés.

Notez que, en logique élémentaire, «non(A et B)» peut aussi s'écrire «(non A) ou (non B)».

Vous pouvez tester votre programme avec quelques valeurs, par exemple -30, -11, -2, -1 ….

Voici à quoi devrait ressembler l'exécution de votre programme :

```
Entrez un nombre décimal : -20
x n'appartient pas à I
...
Entrez un nombre décimal : -10
x appartient à I
...
```

Pour rappel voici quelques exemples d'utilisation d'une structure « if » (le « else » est optionnel) avec les opérateurs booléens classiques « and », « or » :

```
if variable > 2 and variable < 10 :
        print('valeur entre 2 et 10 (exclus)')
if variable > 2 or variable < 10 :
        print('valeur supérieure à 2 ou inférieure à 10 (exclus)')
```