TP Python 3

1- Types, variables, tests

Le plus long des voyages commence par le premier pas.

Lao-Tseu

Préparation :

- les types de base : entiers, flottants, booléens, chaîne de caractères ;
- les variables ;
- les saisies et affichage ;
- les séquences d'instruction et l'alternative if.

Objectifs :

- utilisation d'un « Environnement de Développement Intégré PyCharm» ;
- utilisation des types de base, des variables et de l'affectation ;
- création des premiers scripts de séquences d'instructions.

Fichiers à produire : `TP1_1.py` `TP1_2.py` `TP1_3.py` `TP1_4.py` `TP1_5.py`

1.1.1 - Interpréteur " Calculette "

Python est un *interpréteur* et on peut l'utiliser facilement pour expérimenter le langage en utilisant ce que l'on appelle le **shell Python**. Dans ce mode, Python fonctionne comme une calculatrice avec un ruban déroulant les opérations et les résultats ; vous serez amenés à l'utiliser régulièrement lors des travaux pratiques afin de tester rapidement certaines opérations.

Dans ce mode, le comportement de Python est le suivant :

- Il affiche un message puis une invite (en anglais un « prompt ») : `>>>`.
- Il attend qu'on lui soumette des commandes (par exemple : `2+3`).
- Lorsqu'une commande est validée par l'appui sur alors elle est évaluée et le résultat de cette évaluation est affiché (sauf s'il n'y a pas de résultat - valeur `None`).
- Si la commande est constituée d'une instruction sur plusieurs lignes, l'invite se transforme en `...` sur les lignes suivantes, jusqu'à ce que l'on saisisse une ligne vide pour indiquer que la commande est terminée et que l'on veut que l'exécution se fasse.
- Une fois une commande évaluée et son résultat affiché, l'interpréteur ré-affiche l'invite et attend la saisie d'une nouvelle commande.

Si vous n'avez aucune interface graphique installée, vous pouvez, dans un terminal ligne de commande, taper : `python3 (ubuntu)` puis appuyer sur la touche. Ceci démarre l'interpréteur Python en mode calculette, comme dans Illustration 2 ci-après. Il est alors possible de saisir des expressions Python pour voir quel est le résultat de leur évaluation.

Pour sortir de l'interpréteur, il suffit de taper quit().

```
[andrei@litchi ~]$ python3
Python3.1.2 (r312:79147, Apr 22 2019, 16:11:29)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 4/3*3.141459*2**3 # Volume d'une sphère de rayon 2.
33.50889599999999
>>>
```

Illustration 2: Interpréteur en ligne de commande

Remarque

Dans les lignes de code Python, les **commentaires** sont introduits par le caractère **#** et s'étendent jusqu'à la fin de la ligne. Ils sont destinés à donner des indications aux personnes qui lisent le code, et sont ignorés par Python.

Lorsque l'on fait une erreur dans la ligne de code Python, l'exécution de celle-ci provoque ce que l'on appelle une « levée d'exception », qui par défaut fait afficher diverses informations décrivant l'erreur (cause, ligne où elle a été détectée…). Voici quelques exemples (il y a d'autres catégories d'erreur) :

```
>>> 3*4*(7-2))          # insertion d'une parenthèse fermante de trop
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
Syntax Error: 3*4*(7-2)): <string>, line 110
>>> 3.2/0               # division par zéro
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.ZeroDivisionError: float division
>>> "256"+4             # opération + entre des types incompatibles
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.TypeError: Can't convert 'int' object to str implicitly
>>> 4/3*pi*r**2         # utilisation de variables non définies
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.NameError: name 'pi' is not defined
```

Levées d'exceptions suite à des erreurs

Python dispose d'Environnements de Développements Intégrés (« EDI » ou « IDE » en anglais) qui lui sont adaptés, et qui facilitent autant l'édition du code Python que les tests et l'exécution. Il est d'ailleurs généralement installé avec un EDI simple et écrit lui-même en Python : IDLE.

Dans ces TP, nous utiliserons un EDI libre pour le monde de l'éducation : PyCharm IDE , aussi bien pour la partie« Manip » des tests en mode calculette que pour écrire des scripts et les exécuter.

1.1.2 Types numériques

Le **type entier** `int` et **type flottant** `float`, essayez dans le Python Shell :

```
>>> type(4)
>>> type(4.56)
```

Calculs

Utilisez les quatre **opérations de base** ('**+**', '**-**', '*****', '**/**') sur des entiers relatifs et des flottants. Utilisez la **variable prédéfinie '_'** pour rappeler le dernier résultat.

Essayez et commentez :

```
>>>        20 / 3
```

```
>>> 20 // 3
>>> 20 % 3
>>> 15 / 0
```

Hiérarchie des opérateurs

Essayez et commentez :

```
>>> 7 + 3 * 4
>>> (7 + 3) * 4
```

Cas particulier de la **division entre entiers**, essayez et commentez :

```
>>> 8 / 4
```

Autres opérateurs

Expérimentez : que font ces opérateurs, commentez ?

```
>>> 20.0 // 3
>>> 20.0 % 3
>>> 2.0 ** 10
>>> 5 ** 2 % 3# Ici, quel est l'opérateur prioritaire ?
>>> 5 % 2 ** 3# Et là ?
>>> 18 @ 3
```

Remarque

Python dispose d'autres types numériques, entre autres un type standard `complex`, qui peut s'exprimer directement en spécifiant une valeur de partie imaginaire suffixée par

un j : `4+3.1j`

Certains types numériques sont disponibles dans des modules spécifiques (fractions, nombres décimaux, etc).

1.1.3 - Type chaîne de caractères

Pour les textes, on parle de *chaînes de caractères*[4] ou tout simplement de *chaînes*.

Le **type chaîne** `str`, essayez :

```
>>> type("toto")
```

Les opérateurs de base des chaînes

Les chaînes peuvent être mises bout à bout (opération de **concaténation**), recopiées un certain nombre de fois (opération de **répétition**) et il est possible de connaître leur nombre de caractère (**longueur**).

Essayez et commentez (attention aux espaces) :

```
>>> "Bonjour " + 'Joe' + " " + """Student"""
>>> '~'*60
>>> '*' * 20 + " Bienvenue " + '*' * 20
>>> len('')
>>> len("Bonjour Joe")
```

1.1.4 Type logique booléen

Deux valeurs possibles : vrai et faux. On obtient des valeurs booléennes soit en saisissant directement les constantes Python `True` et `False`, soit en calculant des expressions logiques (par exemple des comparaisons).

Le type booléen `bool`, essayez :

```
>>> type(True)
```

Expressions logiques

Essayez et commentez :

```
>>> 4 < 9
>>> (3*8)!=(2*12)
>>> 3.14159 == 3.141592653589793
>>> 4 < 8 < 7
>>> "Bab" > "Bob"
>>> 3 < "trois"
```

Opérateurs logiques

Essayez et commentez :

```
>>> 3 < 4 and 9 == 3*3
>>> 3-1>4 or 3-1>2
>>> not len("Jean")<=3
>>> (3-1>4 or len("Jean")>3) and (9!=3*3)
```

Mémo : tables de vérité

a	b	not a	a or b	a and b
False	False	True	False	False
False	True	True	True	False
True	False	False	True	False
True	True	False	True	True

1.1.5 - Transtypage

Chaînes et nombres flottants ou entiers ne sont pas équivalents, même s'ils peuvent apparaître identiques lors de l'affichage. Le sens des opérateurs n'est pas toujours le même et des types différents ne peuvent pas toujours être combinés. Essayez et commentez :

```
>>> "3" * 10
>>> 3 * 10
>>> "3" * 10.0
>>> "3"+"3"
>>> 3+3
>>> "3"+3
```

Il est possible d'effectuer une conversion d'une valeur d'un type dans un autre, à condition qu'elle soit compatible. Cette opération s'appelle le transtypage. Essayez et commentez :

```
>>> int("12")
>>> int("12.4")
>>> float("12.4")
>>> str(12)
>>> str(12.4)
>>> bool(12)
>>> bool(0)
```

1.1.6 -Notions de donnée, de variable et d'affectation

Choix des noms de variable

Outre le respect des règles sur la syntaxe des noms (`a…z_` suivi de `a…z0…9_`, en excluant les mots clés du langage), le choix des noms de variables doit se faire de façon à ce que leur lecture fournisse directement une information explicite sur leur utilité. Par exemple : `age`, `maxi`, `moyenne`, `somme_x`…

L'affectation

On *affecte* (ou *assigne*) une valeur à une variable (un nom) en utilisant le signe égal **=**.

⚠ **Attention**
Cela n'a *rien à voir* avec l'opérateur mathématique exprimant l'égalité.

Pour afficher la valeur associée à une variable, on entre simplement son nom comme expression à calculer, on peut même mettre plusieurs variables en les séparant par des virgules.

Essayez :

```
>>> nom = "Student"
>>> prenom = "Joe"
>>> age = 20
>>> etudiant = True
>>> moyenne_notes = 11.5
>>> nom
>>> nom,prenom,age
```

Note : on aurait pû utiliser une *affectation multiple* pour éviter trop de lignes :

```
>>> nom,prenom,age,etudiant="Student","Joe",20,True
```

Les *mots clés réservé du langage* ne peuvent pas être utilisés comme nom de variable, ce sont :

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

On vérifie que c'est le cas en essayant :

```
>>> in = 34
```

Lorsque l'on relie un *contenu* (la valeur `20`) à un *contenant* (la variable `age`), on crée une association au moment de l'affectation. Association qui durera jusqu'à ce que l'on ré-affecte la variable avec une autre valeur. Essayez :

```
>>> age = 72
>>> age
```

Lors d'un anniversaire de Joe, on peut utiliser une **incrémentation** pour ajuster son age. Essayez :

```
>>> age = age + 1
>>> age
```

1.1.7 -Fonctions d'entrées/sorties

Ces fonctions sont très rarement utilisées dans le mode calculette. Par contre, elles seront incontournables dans les scripts de la partie programmation.

input

La fonction d'entrée, **input**, permet d'afficher une directive à l'utilisateur et d'attendre qu'il saisisse des informations au clavier en les validant par un appui sur CR. La fonction retourne le texte tapé par l'utilisateur sous la forme d'une valeur de type chaîne, que l'on stocke généralement dans une variable afin de pouvoir l'utiliser ultérieurement.

Essayez :

```
>>> formation = input("Quelle est votre formation ? ")
>>> moybac = float(input("Moyenne au BAC ? "))
>>> formation,moybac
```

print

La fonction de sortie, **print**, permet d'afficher des valeurs et des variables. Les caractères spéciaux contenus dans les chaînes sont correctement interprétés à l'affichage. Il est possible d'afficher plusieurs données en une seule fois, en les séparant par des virgules. Essayez et commentez :

```
>>> s = "Chaîne\ns'étale \"ici\" sur\n\tplusieurs\n\tlignes."
>>> s
>>> print(s)
```

En réutilisant les variables définies précédemment, essayez et commentez :

```
>>> print("Bonjour,",prenom,nom,"tu dis avoir",age,"ans.")
>>> print("bonjour","à","tous",sep="---")
```

Amélioration de l'affichage : str.format

L'affichage brut, tel qu'il est réalisé par la fonction **print** en demandant simplement la conversion en texte des données, est parfois difficile à lire. Une *méthode* **format** des chaînes de caractères permet de réaliser une mise en forme plus fine à l'aide d'une syntaxe simple.

Quelques commentaires sur la syntaxe de formatage de la méthode **format** :

- Les *{…}* indiquent les endroits où des valeurs doivent être insérées dans le texte, lesquelles et comment les présenter.
- La première information dans les *{}* indique quelle valeur il faut prendre parmi les arguments que l'on donne à la méthode **format**.
- Par *position* avec une indication numérique, le premier argument a le n°0, le second le n°1, le troisième le n°3, etc.
- Par *nom*, il suffit de spécifier **nom=*valeur*** pour chaque argument.
- S'il n'y a pas d'indication, à chaque *{}* correspond un argument,

dans l'ordre où ils se présentent.

- La seconde information dans les `{}`, après le `:`, précise comment la valeur doit être présentée.
- Pour les *nombres flottant*, les lettres de formatage suivantes sont disponibles : `f` pour l'affichage scientifique, `e` pour l'affichage exponentiel, et `g` pour l'affichage optimal (choisit automatiquement entre `e` et `f` suivant l'ordre de grandeur de la valeur, et limite les décimales). Comme vous le voyez dans les exemples, il est possible de spécifier le nombre de décimales à afficher.
- Pour les nombres entier, l'affichage se fait par défaut en décimal (base 10). Les lettres de formatage suivantes sont disponibles : `o` pour l'affichage en octal (base 8), `x` pour l'affichage en hexadécimal (base 16), et `b` pour l'affichage en binaire (base 2).

Il existe bien d'autres options pour `format` (alignement de texte, représentation de signe, etc), pour plus de détails il faut consulter la documentation Python 3.

1.1.8 Instruction if

Cette instruction est une *structure de contrôle* de l'exécution du programme. Elle permet de n'exécuter une séquence d'instructions que si une expression logique est vraie. La séquence d'instructions liée au if est un bloc défini au moyen de l'indentation.

Importance de l'indentation

Toutes les structures de contrôles sont des *instructions composées*. Elles ont toutes la même structure : une ligne d'en-tête terminée par un double point, suivie d'une ou de plusieurs instructions indentées (retrait en début de ligne) sous cette ligne d'en-tête.

La règle est très simple :

Toutes les instructions au même niveau d'indentation appartiennent au même bloc.

⚠ **Attention**

Quel que soit l'éditeur utilisé, *ne pas* mélanger les espaces et les tabulations. Il est fortement conseillé de paramétrer son éditeur pour remplacer systématiquement toutes les tabulations par 4 espaces, selon la convention la plus utilisée dans le monde Python.

Exemple avec if

Essayez :

```
>>> if age>18 :
...                     print(prenom,"est majeur.")
```

Enregistrer dans votre repertoire le script nommé `TP1_if.py`et qui contient les instructions :

```
a = int(input("Valeur de a :
")) if a > 100 :
print("<a> dépasse la centaine") print("C'est
```

```
excessif, on le réduit de 10%.") a = int(a *
0.9)
elif a == 100 :
    print ("<a> vaut 100, rien à
redire.") else :
print("<a> ne vaut pas cher !")
print("On l'augmente de 5 unités.")
a = a + 5
print("Au final, <a> vaut",a)
```

Faire un essai en exécutant le script plusieurs fois (en appuyant sur la touche à chaque fois) et en saisissant différentes valeurs pour `a` : `12`, `100`, `150`. Commentez.

Programmation

⚠ **Attention**

Conventions de nommage : Pour stocker vos programmes, il vous est demandé d'utiliser des noms normalisés : « **`TPn_s.py`** », où n, entre 1 et 8, est le numéro de la séance, suivi d'un souligné puis de s, numéro du script du TP en cours. Le premier script de ce TP doit donc s'appeler : **`TP1_1.py`**, le deuxième **`TP1_2.py`**, etc.

Dans cette seconde période, on va enregistrer les commandes Python dans un fichier (que l'on appelle souvent un « script »).

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""Modèle de script.
Documentation du
module. """
# fichier:
modele_script.py # auteur:
Bibi

# imports

# ---------- Programme Principal ----------
```

Explication du bloc d'en-tête

- `#!/usr/bin/python3`
  ce commentaire spécial assure que vos scripts fonctionneront aussi bien sous Windows que sous Linux, MacOS X ou autre Unix, il doit absolument être sur la première ligne du fichier ;
- `# -*- coding: utf-8 -*-`
  ce commentaire spécial définit l'*encodage*, il permet d'utiliser tous les accents habituels

du français, il doit absolument être sur une des deux premières lignes du fichier ;

- `"""Modèle de script..."""`
  docstring : documentation générale du script, sert pour l'aide en ligne, accessible via **`nom_module.__doc__`**;
- `# fichier: modele_script.py`
  nom de votre script (indispensable pour l'impression) ;
- `# auteur: Bibi`
  nom de l'étudiant (indispensable aussi !) ;

Ensuite, des commentaires permettent de délimiter les différentes parties du script.

1.2.1 - Calculs de l'aire d'un rectangle

Programme TP1_1.py

- Saisir deux valeurs flottantes dans deux variables **`largeur`** et **`longueur`**.
- Calculer l'aire de la surface correspondante et stockez là dans une variable **`aire`**.
- Réaliser un affichage propre pour donner le résultat du calcul.

1.2.2 - Somme de deux résistances

Programme TP1_2.py

- Saisir deux valeurs flottantes de résistances dans des variables **`r1`** et **`r2`** en indiquant l'unité dans le message (on peut utiliser **`"\u2126"`** pour afficher un joli Ω);
- effectuer l'affichage formaté (deux chiffres significatifs) de la résistance équivalente en série et en parallèle. Ne pas oublier l'unité.

1.2.3 Equation du second degré

Programme TP1_3.py

Calcul des racines réelles du trinôme : $ax^2 + bx + c = 0$

Juste après l'en-tête du fichier, insérez l'instruction :

```
from math import sqrt
```

Ensuite :

- Saisir trois nombres flottants correspondants aux coefficients et les stocker dans des variables **`a`**, **`b`** et **`c`**.
- Calculer la valeur du discriminant, **`delta`**.

- En testant la valeur de `delta` (négatif, nul ou positif), afficher qu'il n'existe pas de racine réelle, ou la valeur de la racine réelle double, ou enfin les valeurs des deux racines réelles. N'oubliez pas de formater proprement vos affichages.

1.2.4 Calcul de la division euclidienne

Programme TP1_4.py

1. Saisir deux entiers positifs `a` et `b` ;
2. calculer le quotient `q` et le reste `r` ;
3. afficher la division euclidienne. Avec par exemple 45 et 11, on doit obtenir :

Ce calcul nécessite l'utilisation de la racine carrée. La fonction `sqrt` appartient évidemment au module `math`. Regardez sa syntaxe exacte dans la documentation (ou tapez help(sqrt) dans un interpréteur)

```
Division euclidienne: 45 = 11*4 + 1
```

1.2.5 - Règles logiques

Programme TP1_5.py

Soit la règle administrative suivante, concernant le paiement de l'impôt sur le revenu en un seul règlement (source : http://vosdroits.service-public.fr/F3118.xhtml) :

Principe

Le contribuable peut, s'il le souhaite, opter pour le prélèvement de chacun de ses impôts, à la date limite de paiement qui figure sur l'avis d'imposition.

Pour l'impôt sur le revenu, le contribuable paiera son impôt de 2010, sur ses revenus perçus en 2009, en 1 seule fois :

- si le montant de son impôt de 2009, sur ses revenus perçus en 2008, était inférieur à 337 € ,
- ou s'il est imposé pour la première fois en 2010, sur ses revenus perçus en 2009,
- ou si son impôt de 2009, sur ses revenus perçus en 2008, a été mis en recouvrement après le 15 avril 2010 (la date de mise en recouvrement doit figurer sur l'avis d'imposition),
- ou s'il n'a pas été imposable en 2009, sur ses revenus perçus en 2008, même s'il a déjà été imposé les années précédentes.
- À partir de ce texte, identifiez les différentes informations nécessaires pour savoir si le contribuable paiera son impôt obligatoirement en une seule fois. Associez une variable à chaque information, associée à une valeur possible. Écrivez-la ou les expressions logiques, utilisant les variables que vous avez définies, et permettant de connaître la décision sur le paiement obligatoire

en une seule fois.