TP n°1 : Introduction à Python

Instruction générale : hormis pour les exercices corrigés collectivement, vous ferez valider votre travail par l'enseignant.

1 Bonnes pratiques

Ils existent de bonnes pratiques en Python afin de rendre son code lisible et bien organisé. On peut retrouver celles-ci sur le site du PEP 8. On veillera à respecter les pratiques suivantes pour chacun de nos codes.

1. Commencer son code en précisant l'encodage du fichier, la version de Python sous lequel il est édité et son auteur. Par exemple :

   ```
   # Encodage : UTF8
   # Version Python : 3.10
   # Auteur : Homer Simpson
   ```

   Cela est parfois fait de façon automatique par les logiciels d'édition de code.

2. Donner des noms explicites aux variables (pas trop longs). Pour des noms de variables composés, on utilisera des notations facilitant la lisibilité à l'aide d'underscores `_` ou de majuscules afin de séparer les mots. Par exemple, pour une variable associée à Homer Simpson, `hs` n'est pas un bon nom ; on privilégiera `homer_simpson` ou `homerSimpson`.

3. Aérer le code en sautant des lignes et des doubles lignes afin de bien identifier les différentes parties du code.

4. Mettre des commentaires afin de décrire les actions des différentes parties du code, leurs fonctionnements, les types et utilités des variables, etc. Les commentaires courts sont déclencher à l'aide du symbole `#` ; ils se terminent en passant à une nouvelle ligne. Par exemple les informations de début de code ci-dessus. Les commentaires longs et s'étalant sur plusieurs lignes commencent et se terminent à l'aide de triples guillemets `"""`. Par exemple :

   ```
   # Ceci est un petit commentaire

   """ Ceci est un très très,
   mais alors vraiment très long
   et très inutilement long
   commentaire."""
   ```

2 Affectation et types de données

En Python, la création d'une variable et l'affectation d'une valeur à celle-ci se font simultanément grâce à l'opérateur `=`.

Exemple : dans le code ci-dessous, on affecte aux variables `a` et `b` les valeurs 3 et "mot" en même temps qu'on les crée.

```
a=3
b="mot"
```

Le tableau ci-dessous donne les correspondances des types de données en Python.

Type de donnée	Booléen	Entier	Flottant	Chaîne de caractères
Python	`bool`	`int`	`float`	`str`

Exercice 1.

1. Dans le code Python ci-contre, identifier les variables présentes et leurs types.
2. Copier ces lignes de code dans l'interpréteur Python. À l'aide de la commande `type()`, vérifier les types des variables.

```
x=5
y=x/2
t="1"
b=False
```

3 Entrée et sortie

3.1 Entrée

La commande `input()` permet en Python au programme de demander à l'utilisateur d'entrer des valeurs qui seront affectées à des variables lors de son exécution. Si l'on ne met rien entre les parenthèses de la commande `input()`, l'ordinateur effectue une demande « muette », il attend qu'on rentre la valeur sans rien dire. Pour remédier à cela, on peut insérer un texte entre parenthèse afin d'aider l'utilisateur, par exemple `input("Quelle valeur voulez-vous entrer ?")`.

Exemple : dans le code ci-dessous, l'utilisateur doit entrer une valeur entière qui sera affectée à `a`.

```
a=int(input("Quel entier pour a ?"))
a=a**2
```

Exercice 2. On reprend le code ci-dessus.

1. Que vaut `a` à la fin du programme si on entre pour valeur 2 ?
2. Que se passe-t-il si l'utilisateur entre pour valeur de `a` 2.5 ? Que faudrait-il changer pour que `a` soit un flottant ?
3. Si l'on ne précise pas le type au `input()`, quel sera le type par défaut de `a` ?

3.2 Sortie

L'ordinateur ne nous montre pas les résultats de l'exécution d'un programme à moins qu'on ne lui commande. Les instructions à utiliser pour cela en Python diffère selon ce que l'on veut afficher (texte, graphe, etc). La commande `print()` permet d'afficher le contenu entre parenthèses ; cela peut être le contenu d'une variable ou directement le résultat d'une opération. Si l'on veut afficher plusieurs choses, on peut utiliser un seul `print()` en séparant les contenus par des virgules.

Exemple : dans le code ci-dessous, l'utilisateur doit entrer une valeur entière qui sera affectée à `a`.

```
mot="truc"
print(mot) # affichera "truc"
print("machin",mot) # affichera "machin truc"
print(3*2+1) # affichera 7
```

Exercice 3. Écrire un programme qui :

1. prend en entrée un flottant et donne en sortie son carrée ;
2. prend en entrée un entier et donne en sortie le quotient et le reste de sa division euclidienne par 5 ;
3. prend en entrée un flottant et donne en sortie sa partie entière (sous forme d'entier).

4 Le booléen, le test et l'instruction conditionnelle

4.1 Le booléen et le test

Le résultat d'un test est un booléen, soit ce qui est testé est Vrai, soit c'est Faux. En Python, les opérateurs de tests d'égalité et d'inégalités sont `==`, `>=`, `<=`, `<` et `>`. Si on veut que le résultat du test soit affectée à une variable (quelle sera son type ?), il faut mettre le résultat entre parenthèse. L'instruction `not()` permet d'obtenir le contraire d'un booléen.

Exemple :

```
x==y # test permettant de savoir si les variables x et y sont égales
b=(x==y) # booléen défini par le test x==y
```

Remarque : attention, pour les tests d'inégalités larges, le `=` est toujours à droite par convention. À gauche : `=>`, `=<`, on obtient des erreurs.

Exercice 4. Que donne les instructions suivantes lorsqu'on les tape dans l'interpréteur Python ?

Instruction	Résultat et commentaire
`3==3`	
`8==3`	
`8!=3`	
`(7+3)==(5*2)`	
`(3==3) and (2<2)`	
`(3==3) and (2<=2)`	
`(2==15) or (3>2)`	
`1<4<7`	
`b=(3==3)`	
`b`	
`type(b)`	
`not(b)`	
`True and False`	
`True or False`	
`not(False)`	

4.2 L'instruction conditionnelle

En Python, les instructions conditionnelles Si, SinonSi et Sinon correspondent à `if`, `elif` et `else`. Après les conditions à vérifier, on doit mettre un : avant de passer à la séquence d'instructions conditionnelles, cette dernière étant marquée par une indentation (espace en début de ligne). La fin de l'indentation marque la fin de l'instruction conditionnelle.

Exemples :

— Le programme suivant dit si un nombre est positif, négatif ou nul.

```
nbr=float(input())
if nbr>0 :
    print("Le nombre est positif.")
elif nbr<0 :
    print("Le nombre est négatif.")
else :
    print("Le nombre est nul.")
```

— Le programme ci-dessous renvoie la valeur absolue d'un nombre donné en entrée.

```
nbr=float(input())
if nbr<0 :
    nbr=-nbr
print(nbr) # il n'y a plus d'indentation ici, on est sorti du if.
```

Remarque : on peut voir dans l'exemple ci-dessus qu'un `if` n'est pas nécessairement suivi de `elif` ou `else`.

Exercice 5. [Résultat à l'examen] Écrire un algorithme puis le coder en Python prenant en entrée la note d'un candidat et en fonction de celle-ci donnant son résultat selon le tableau ci-dessous (la borne supérieure sera considérée comme strictement ouverte). On veillera à bien optimiser les critères et le nombre d'opérations puis on donnera un jeu de notes permettant de tester l'algorithme.

Note	**Résultat**
Entre 0 et 8	Recalé-e
Entre 8 et 10	Rattrapage
Entre 10 et 12	Admis-e
Entre 12 et 14	Admis-e mention assez bien
Entre 14 et 16	Admis-e mention bien
Supérieure à 16	Admis-e mention très bien

Exercice 6. Écrire un algorithme puis le coder en Python prenant en entrée un nombre entier et disant s'il est pair ou impair. On pourra considérer le reste r de la division euclidienne de l'entier n par 2 : $n = 2q + r$ (q quotient).

5 Ressources supplémentaires

— Les cours sur Python d'OpenClassRooms
— Le cours du W3 Schools
— Vidéo sur les variables
— Vidéo sur les instructions conditionnelles