Tutorat de Programmation
L1 Informatique

I – Style

Le code que vous écrivez est censé être lu par des humains. Vous en premier.
Il est important que ce code soit correct, mais il est également important qu'il soit beau.

Qu'est-ce que du beau code, vous demandez ?
C'est du code **lisible**.

Vous devez toujours respecter les règles suivantes pour produire du code lisible :

- Insérez toujours un espace après une virgule
- Chaque '=' d'assignement doit être entouré d'espace des deux côtés.
  ```
  target = "lol"
  ```
- Chaque opérateur de comparaison doit être entouré d'espace des deux côtés.
  ```
  if last == "9":    Ou    while len(password) < 5:
  ```
- N'utilisez pas de parenthèses pour englober des conditions, comme dans les deux exemples précédents. Les parenthèses doivent uniquement être utilisées pour forcer des groupements d'opérandes.
- L'indentation est cruciale. En python, du code ne marche pas correctement si il n'est pas bien indenté et c'est une bonne chose car cela vous force à prendre des bonnes habitudes. Il est quand même possible de mettre un bloc sur la même ligne qu'une expression de contrôle, mais ce n'est pas une bonne chose. Dans l'exemple suivant, la forme de la ligne 27 est à éviter. N'utilisez que la version sur deux lignes comme aux lignes 29 et 30.

```
27 if last == 9: return False
28
29 if last == 9:
30     return False
```

- Choisissez des noms parlant pour vos noms de fonctions et vos variables. Les noms font autant partie de la documentation d'un code que les commentaires. Dans l'exemple suivant, le code sur les lignes 4 et 5 est beaucoup plus facile à comprendre que celui sur les lignes 7 et 8.

```
4 if everyBodyDied and newGameRequested == False:
5     gameOver()
6
7 if a and b == False:
8     c()
```

II – Précédence des opérateurs

Comme en mathématique, les opérateurs en Python ont divers degrés de précédence. Il s'agit de l'ordre dans lequel une expression va être évaluée.

Dans le listing suivant, au plus on descend, au plus la précédence des opérateurs augmente :

```
1 a or b      # Boolean OR
2 a and b     # Boolean AND
3 not x       # Boolean NOT
4 in, not in, <, <=, >, >=, !=, == # Membership test, and comparison operators
5 +, -        # Addition and Subtraction
6 *, /, %     # Multiplication, Division and Remainder
7 **          # Exponentiation
```

Cela veut dire par exemple, que si vous écrivez le code suivant :

```
1 c1 = False
2 c2 = True
3 c3 = True
4 b = c1 or c2 and c3
```

Après l'exécution de la ligne 4, la valeur de « `b` » sera « `False` ».
Comme « `and` » est plus bas dans la liste, « `c2 and c3` » est exécuté en premier et produit « True ». « `c1 or True` » est ensuite exécuté et produit « `False` ». La valeur « `False` » est ensuite stockée dans la variable « `b` ».