Conseils de programmation

Exemple 1 :

Écrire une fonction `mention(note)` qui affiche la mention obtenue à partir d'une `note` sur 20. Les mentions possibles sont "Très bien", "Bien", "Assez bien", "Pas de mention", "Échec".

```
>>> mention(12)
Assez bien
>>> mention(17.5)
Très bien
>>> mention(3)
Échec
```

Exemple 2 :

Quelqu'un met une somme `init` sur un compte en banque et n'y touche plus. Chaque année, la somme sur son compte est mutlipliée par `coeff`, un nombre strictement supérieur à 1. Écrire une fonction `compte_banque(init, coeff, objectif)` qui renvoie le nombre d'années qu'il faut pour que la somme sur le compte soit supérieure ou égale à `objectif`.

```
>>> compte_banque(100,2,400)
2
>>> compte_banque(100,1.1,400)
15
```

Exemple 4 :

Écrire une fonction `palindrome(texte)` qui renvoie un booléen indiquant si le texte est palindrome, c'est à dire s'il s'écrit de la même façon dans les deux sens.

```
>>> palindrome("kayak")
True
>>> palindrome("baba")
False
>>> palindrome("o")
True
```

Commencer par mettre le début et la fin de la fonction

Tout d'abord il faut commencer à écrire la définition de la fonction puis déterminer si elle doit afficher (donc print) ou retourner (donc return) quelque chose.

Exemple 1 :
Il faut afficher un message. On crée une variable message, qu'on initialise avec un texte vide et on l'affiche à la fin de la fonction.

```
def mention(note):
    message = ""
    ...
    print(message)
```

Exemple 2 :
On doit renvoyer un nombre. Nous allons créer une variable n qui correspond au nombre d'années. On commence à 0 et on le renvoie à la fin.

```
def compte_banque(init, coeff, objectif):
    n = 0
    ...
    return n
```

Exemple 4 :
On doit renvoyer un booléen. On crée une variable rep qu'on renverra à la fin. On commence par le mettre à **True**.

```
def palindrome(texte):
    rep = True
    ...
    return rep
```

Choisir une structure

Il faut ensuite déterminer si nous allons faire une boucle, des tests, des tests dans une boucle...Pour cela, on peut essayer de traiter quelques cas à la main en regardant si on répète plusieurs fois la même chose (boucle), si on fait une étude de cas (tests)...Dans le cas de boucles, il faut déterminer si connaît le nombre d'itérations (boucle for) ou si on ne peut pas le déterminer à l'avance (boucle while).

Exemple 1 :
On va comparer note avec 16, 14, 12 et 10 pour savoir dans quel intervalle elle se trouve. C'est donc une étude de cas, donc on peut utiliser une structure **if**, **elif**, **else**.
Pour chaque cas, on met à jour message avec le bon message. Il faut bien faire attention à bien attribuer la bonne mention pour les valeurs limites.

```
def mention(note):
    message = ""
    if note >= 16:
        message = "Très bien"
    elif 14 <= note < 16:
        message = "Bien"
    elif 12 <= note < 14:
        message = "Assez bien"
    elif 10 <= note < 12:
        message ="Pas de mention"
    else:
        message = "Échec"
    print(message)
```

Exemple 2 :

On va calculer, année après année, la somme présente sur le compte en banque. On va continuer jusqu'à atteindre ou dépasser `objectif`. On va donc utiliser une boucle **`while`**.
Il faut créer une variable pour enregistrer la somme sur le compte. On va l'appeler `compte`.
La condition sera donc `compte < objectif`.
À chaque tour de boucle, on calcule la nouvelle valeur de `compte` en multipliant par `coeff`.
Il faut aussi rajouter 1 à `n`.

```
def compte_banque(init, coeff, objectif):
    compte = init
    n = 0
    while compte < objectif:
        compte *= coeff
        n += 1
    return n
```

Exemple 4 :

On va parcourir le mot en comparant la première lettre avec la dernière, puis la deuxième et l'avant dernière et ainsi de suite. On a donc besoin de la position de la lettre courante.
On va donc utiliser une boucle **`for i in range(len(texte))`**.
On va regarder la lettre `texte[i]`. Il faut trouver la position de la lettre qui doit être égale.
On peut faire un tableau :

Position regardée	0	1	2	...	i
Position symétrique	-1	-2	-3	...	?

On remarque qu'à chaque fois, la position symétrique correspond à l'opposé de la position regardée moins 1. Donc la position symétrique de i est $-i-1$.
On doit donc comparer `texte[i]` avec `texte[-i-1]`. Le texte est un palindrome si toutes les lettres symétriques sont égales. Ce n'est pas un palindrome s'il y a un seul couple de lettres symétriques qui sont différentes. On va donc mettre `rep` à **`False`** quand on voit une différence et surtout ne rien faire si les lettres sont égales.

```
def palindrome(texte):
    rep = True
    for i in range(len(texte)):
        if texte[i] != texte[-i-1]:
            rep = False
    return rep
```

Tester la fonction

Une fois que la fonction est "terminée", il faut la tester. Vous pouvez le faire avec les exemples donnés, s'il y en a, mais vous pouvez aussi faire vos propres tests.
Mais vous n'avez pas à attendre d'avoir fini votre fonction pour commencer à la tester. Vous avez peut-être traité certains cas. Vous pouvez commencer à la tester le plus tôt possible. Plus votre code est long, plus est difficile de trouver les erreurs. Il vaut donc mieux tester en cours d'élaboration plutôt que d'attendre d'avoir tout fini.
Il faut aussi penser aux valeurs "extrêmes": listes vides, listes avec plusieurs fois la même valeur, valeurs aux bornes des intervalles pour des tests...

Corriger les erreurs syntaxiques

S'il y a des erreurs syntaxiques (messages en rouge), il faut essayer de corriger. Pour cela, il faut bien regarder le message qui indique le type d'erreur.

Erreur	NameError: name 'maxim' is not defined
Signification	Un nom apparaissant dans une expression n'a pas été défini au préalable.
Cause/solution	Soit vous n'avez pas donné une valeur initiale à une variable, soit vous avez mal écrit un nom de fonction ou de variable.
Erreur	IndentationError: unexpected indent
Signification	Il y a un changement d'indentation entre 2 lignes sans que Python n'arrive à déterminer pourquoi.
Cause/solution	Vous pouvez vérifier si la ligne incriminée est bien indenté comme elle le devrait. Si c'est le cas, c'est peut-être la ligne précédente qui est mal indentée. Si possible, évitez de faire les indentations avec des espaces et préférez des tabulations.
Erreur	SyntaxError: invalid syntax
Signification	Problème de syntaxe
Cause/solution	Vous avez probablement oublié de mettre ":" après un for, while...Au contraire vous avez peut-être mis ":" après une ligne où il n'y en a pas besoin. Ou alors vous avez mis un test après un else, oublié un test après un if...Ou vous n'avez pas fermé des parenthèses...Ou vous avez mis "=" à la place de "==". En gros, vous n'avez pas respecté les règles de syntaxe de Python.
Erreur	TypeError: unsupported operand type(s) for +: 'int' and 'str'
Signification	Opération impossible avec les valeurs données
Cause/solution	Vous avez essayé d'additionner un nombre et une liste, essayé de calculer la longueur d'un nombre, essayé de parcourir un nombre comme une liste...Vous avez soit oublié des guillemets ou au contraire vous en avez mis trop. Si cela intervient sur des variables, c'est qu'il y a peutêtre une affectation précédente où une des variables a reçu une valeur qui n'était pas du bon type. Faites un print pour afficher les valeurs des variables en cause.
Erreur	TypeError: maximum() takes 1 positional argument but 2 were given
Signification	Le nombre d'arguments donnés à la fonction est incorrect.
Cause/solution	Vous avez donné trop ou pas assez d'arguments. Vous avez peut-être mis 1,2 au lieu de 1.2.
Erreur	IndexError: list index out of range
Signification	Vous essayez d'accéder à un élément qui n'est pas dans la liste.
Cause/solution	Vous devez certainement réduire le nombre d'itération de la boucle. Ou vous vous êtes trompés dans la formule permettant de calculer la position de l'élément que vous regardez. Faites un print pour essayer de voir quelle est la position que vous avez calculée.

Chercher les autres erreurs

Si votre fonction “fonctionne”, dans le sens où il n’y a plus d’erreurs de syntaxe ou d’indentation, mais que vous n’obtenez pas le résultat obtenu c’est que vous avez fait une erreur algorithmique. C’est là que les problèmes commencent. Voici quelques conseils

Pour les boucles:

- Si vous avez une boucle while qui ne termine pas, vérifiez que vous modifiez bien la valeur des variables qui sont utilisées dans la condition de la boucle.
- Si au contraire il semblerait que lors de l’exécution, la boucle ne soit pas utilisée, c’est peut-être que vous vous êtes trompé dans la condition (> au lieu de <) ou les valeurs initiales des variables ne sont pas bonnes.
- De façon générale, mettez des print pour afficher les valeurs des variables à chaque tour de boucle. Vous pouvez en afficher plusieurs à la fois: print(a, b, c).

Pour les tests:

- Si aucun des cas ne semble réalisé, peut-être que votre étude de cas n’est pas exhaustive. Peut-être qu’il faudrait rajouter un else avec un print("Bizarre") pour voir si vous avez râté un cas. Si vous avez des tests imbriqués, vérifiez qu’il est possible que les tests
  le plus à l’intérieur puisse être vérifiés.
- Si plusieurs cas semblent être exécutés, peut-être que vous avez mis if à la place de elif.
- Si ce n’est pas le bon cas qui est exécuté, vérifiez que le premier test s’une série de if...elif n’est pas trop “large” et n’englobe pas des cas suivants.

Autres problèmes:

- Si un compteur n’augmente pas, est-ce que vous n’avez pas mis n = 1 ou n + 1 au lieu de n += 1.
- Si une variable n’est pas modifiée alors qu’elle devrait l’être, vérifiez que vous avez bien mis le bon nom pour l’affectation. Peut-être que vous donnez une valeur à une autre variable et non pas à celle que vous souhaitiez modifier. Par exemple miliue au lieu de milieu.

Quelques bonnes pratiques

Même si votre programme vous semble clair, il ne l'est peut-être pas pour quelqu'un d'autre et il pourrait très bien vous sembler incompréhensible d'ici quelques semaines. Il y a donc quelques "astuces" afin de rendre votre code lisible par n'importe qui.

Bien choisir des noms de variables et de fonctions:

`maximum`, `total` et `moyenne` sont plus compréhensibles que a, b ou z7.

Mettre des commentaires:

Si vous utilisez quelques astuces ou raccourcis, il peut être utile de l'indiquer et d'expliquer pourquoi cela marche. Vous pouvez aussi expliquer le rôle de certaines valeurs ou justifier certaines formules. Pour des tests, et surtout des tests imbriqués, il peut être utile de préciser à quels cas correspondent les else.
Par contre vous pouvez éviter "n = 1 *# mettre n à 1*".

Aérer le code:

Pensez à sauter une ligne entre les définitions des fonctions. Vous pouvez aussi sauter une ligne à l'intérieur d'une fonction, si cela permet de bien séparer des étapes différentes. Pensez aussi à mettre des espaces autour des opérateurs, pour rendre le code plus lisible "n = x + 3" au lieu de "''n=x+3''.

Découper en fonctions:

Si vous faites plusieurs fois la même chose dans votre code, vous pouvez peut-être créer une fonction qui fait cela une bonne fois pour toute et ainsi appeler la fonction à chaque fois que vous en avez besoin. Comme cela, en cas de modification, vous n'avez qu'un bout de code à mettre à jour. Vous pouvez aussi découper en plusieurs fonctions le corps d'une fonction qui serait particulièrement longue. Cela permet ainsi d'expliciter les différentes étapes.