TP Python : les boucles « Tant que »…

La syntaxe de la boucle « Tant que » :

```
while condition :
      instruction1
      instruction2
      etc.
suite du programme
```

Ne pas oublier les tabulations (les *indentations*) !

Les conditions s'écrivent : *variable* *symbole* *variable ou nombre ou texte ou calcul*

où *symbole* est un des symboles suivants :

< > <= >= == != in

Exercice 1 : la punition

Bart doit régulièrement recopier un texte un certain nombre de fois.
Le texte change change chaque semaine (oui, il est collé toutes les semaines…) et le nombre de fois qu'il doit le recopier change aussi.
Sa sœur, trop gentille, lui propose une fonction qui demande à Bart la phrase à recopier, le nombre de fois qu'il doit la recopier et qui fait la punition à sa place…

Recopiez dans Thonny puis complétez le programme suivant :

```
def recopier(phrase, nb_fois):
    ……
    while …… :
        print(……)
        ……

# voici un essai de la fonction :
recopier("Je dois venir en cours", 100)
```

Testez ce programme bien sûr en faisant quelques essais.

Exercice 2 : code secret

Samuel a oublié son code bancaire. Il a droit à 5 essais, après quoi sa carte restera bloquée dans le distributeur.
Le code oublié est 1234 (dur à oublier pourtant…).

Écrivez et testez un programme qui demandera à Samuel son code et bloquera sa carte au bout de cinq erreurs (votre programme affichera simplement « Carte bloquée » dans ce cas).

Indication : pour que Samuel puisse entrer un code, utilisez l'instruction :

```
essai_code = input("Entrez votre code")
```

Pour aller plus loin : affichez le nombre d'essais restants avant le blocage.

Exercice 3 : à la recherche d'une formule magique

On raconte que le jeune Carl Friedrich Gauss avait un jour été puni et devait calculer la somme de tous les entiers de 1 à 100 pendant sa récréation.
Le jeune homme, plus doué en math que la moyenne, a trouvé une formule et terminé le calcul rapidement, ce qui lui a permis de profiter de sa récréation.

1°) Écrivez un programme qui calcule la somme des entiers de 1 à *n* où *n* est choisi par l'utilisateur.

2°) Testez ce programme avec différentes valeurs de *n* et essayez de trouver la formule qu'aurait (re)trouvé Gauss.

Exercice 4

Écrivez et testez une fonction qui donne la première puissance de 3 supérieure à un certain seuil :

```
def puiss(seuil):
    ……
    …… # le nombre de lignes n'est pas forcément celui-là
    return ……
```

Par exemple, `puiss(1000)` doit renvoyer 7
car $3^6 = 729 < 1000$ et $3^7 = 2187 > 1000$.

et `puiss(2000000)` doit renvoyer 14
car $3^{13} = 1594323 < 2000000$ et $3^{14} = 4782969 > 2000000$.

Exercice 5 : encadrement

Nous cherchons ici à encadrer la valeur de $\sqrt{2}$ à 10^{-n} près.
Par exemple, un encadrement de $\sqrt{2}$ à 10^{-5} près est : $1{,}41421 < \sqrt{2} < 1{,}41422$.
Bien sûr, nous ne pouvons pas utiliser la fonction `sqrt` de Python !
Voici donc la méthode employée :

- je sais que $1 < \sqrt{2} < 2$ (pourquoi ?) ;
- je prend le juste milieu entre 1 et 2 : 1,5 ;
- je calcule son carré : $1{,}5^2 = 2{,}25$;
- j'en déduis que $1 < \sqrt{2} < 1{,}5$ (pourquoi ?) ;
- l'amplitude de l'encadrement est alors 0,5 : c'est plus grand que 10^{-5} ;
- je recommence en prenant le juste milieu entre 1 et 1,5 ;
- etc. tant que l'amplitude est trop grande.

Petite aide :

```
def encadrement_racine_deux(n):
    ……
    …… # le nombre de lignes n'est pas forcément celui-là
    return (inf, sup)
```

Par exemple, `encadrement_racine_deux(3)` doit renvoyer (1.414, 1,415)
`encadrement_racine_deux(5)` doit renvoyer (1.41421, 1,41422).

Le hasard avec Python

Il est souvent utile de créer des nombres aléatoires, que ce soit pour des simulations informatiques ou pour des jeux. Python ne sait pas faire cela à la base, on doit donc ajouter à Python de nouvelles fonctions.
Il y a pour cela deux instructions possibles, à taper au début du programme :
soit :

```
>>> from random import randint
```

qui veut dire : dans la bibliothèque random, aller chercher la fonction randint.
soit :

```
>>> import random
```

qui veut dire : aller chercher toutes les fonctions de la bibliothèque random.

Avec la première syntaxe, vous pourrez taper :

```
>>> randint(1, 6)
```

pour simuler un lancer de dé (utilisez la flèche vers le haut du clavier pour relancer cette commande)

et avec la seconde :

```
>>> random.randint(1, 6)
```

Exercice 6 : un jeu

Écrivez et testez un programme qui :

- choisit un nombre entier au hasard ;
- demande au joueur de le trouver :
- le joueur a le droit à autant d'essai qu'il le souhaite ;
- la machine dit à chaque essai si le nombre mystère est plus petit ou plus grand que le nombre choisi par le joueur.

On peut à la fin attribuer un score au joueur en fonction du nombre d'essai qu'il a réalisé.

Exercice 7 : problème du duc de Toscane

Le duc Cosme II de Médicis, protecteur de Galilée lui a un jour posé le problème suivant :

« J'ai remarqué que quand je lance trois dés, j'obtiens plus souvent la somme 10 que la somme 9, alors qu'il y a six façons d'obtenir l'un ou l'autre !».

1°) Quelle sont les six façons d'obtenir un 9 ? les six façons d'obtenir un 10 ?
2°) Écrivez un programme qui permet de vérifier l'affirmation du duc de Toscane.