TP d'informatique n°3 : boucle while et chaine de caractères

PCSI 2023 – 2024

I Avant le TP

- Qu'est-ce qu'une chaine de caractères ? Comment obtenir le nombre de caractères d'une chaine ?
- Qu'est-ce que la concaténation ? Comment concaténer deux chaines ?
- Si `c = "physique"`, que faudrait-il mettre entre les `[]` dans la commande `c[ ??? ]` pour obtenir la chaine `"euqisyhp"` ?
- Rappeler la syntaxe d'une boucle while.
- Lire le paragraphe ci-dessous.

Python effectue une évaluation intelligente (dite « paresseuse ») des booléens : si on écrit « A ou B » alors python regarde d'abord la condition A. Si A est vrai alors python n'évalue pas l'expression B et renvoie vrai puisque « vrai ou B » sera vrai quelque soit B. De même si on écrit A et B et que A est faux, il est inutile d'évaluer B. Ainsi la condition $1/x > 1$ peut renvoyer une erreur si $x = 0$, mais la condition $x! = 0$ and $1/x > 1$ ne renverra jamais d'erreur.

Les booléens sont donc évalués de gauche à droite et python, comme la plupart des langages de programmation modernes, s'arrête dès qu'il peut conclure, même s'il n'a pas testé toutes les conditions.

Le test $1/x > 1$ and $x! = 0$ peut-il renvoyer une erreur dans certains cas ?

II Chaines

Exercice n°1 Écrire une fonction `supprime(chaine:str, x:str)->str` qui prend comme argument une chaine de caractères `chaine` et un caractère `x` et qui renvoie la chaine de caractère sans le caractère `x`. Par exemple `supprime("chimie", "i")` doit renvoyer `"chme"`.

On pourra par exemple créer une chaine vide, puis parcourir la chaine en « recopiant » les caractères sauf s'ils valent x.

Exercice n°2 Une chaine d'ADN valide est une chaine non vide et formée exclusivement d'une combinaison arbitraire de "a" , "t" , "g" ou "c" .
Écrire une fonction `valide(ADN:str)->bool` qui renvoie `True` si la chaine ADN est valide et `False` sinon.

III While

Rappel : lorsque l'on écrit **while** `condition: instructions`, il faut prendre garde à deux choses :

- si la conditions n'est pas vérifiée avant la boucle, alors on ne rentre pas dans la boucle du tout (c'est parfois une erreur de conceptions, mais dans certains cas cela peut-être volontaire) ;
- il faut être sûr de sortir de la boucle :
  $\longrightarrow$ des instructions dans le bloc de la boucle « while » doivent être susceptibles de modifier le booléen évalué dans la condition. Sans cela, on risque de boucler à l'infini.

```
i = 0
while i < 100:
    instructions
    i = i + 1
```

```
for i in range(100):
    instructions
```

De plus, si on ajoutant un compteur (variable que l'on incrémente de 1 à chaque passage dans la boucle), il est facile d'utiliser une boucle while comme une boucle for (cf exemples ci-dessus). Toutefois, les boucles

whiles sont plus « dangereuses ». On préfèrera utiliser une boucle for si le nombre d'itération est connu à l'avance.

Exercice n°3 Testez le programme suivant puis expliquez son comportement

```
def bug():
    s,n,compteur = 0,0,1
    while(n<100):
        s = s + n
        compteur += 1
        print(compteur,'passage dans la boucle')
    return s
bug()
```

Lorsque l'on écrit une boucle while, il faut penser à vérifier que la condition dans le while peut changer, autrement on risque de faire une boucle infini qui utilise le processeur en boucle pour rien. On doit alors utiliser ctrl+C pour arrêter le programme (ou fermer la fenêtre python dans laquelle il s'exécute).

Exercice n°4 Créer une fonction (qui utilise une boucle while) qui prend en argument un nombre N et qui renvoie le plus grand entier n tel que $1^2 + 2^2 + 3^2 + \cdots + n^2 \leqslant N$.
On pourra stocker la somme partielle dans une variable. Penser à vérifier son résultat (par exemple $N = 14$ donne $n = 3$, $N = 1400$ donne $n = 15$, $N = 14000$ donne $n = 34$).

Exercice n°5 La suite de syracuse est un exemple de problème mathématique d'apparence très simple, mais dont la démonstration n'est toujours pas connue à l'heure actuelle.
La suite est définie récursivement de la façon suivante :

- $u_0 = p$ avec p un entier naturel non nul
- si u_k est pair, alors $u_{k+1} = u_k/2$; sinon $u_{k+1} = 3u_k + 1$

Lorsque le nombre 1 est atteint, les nombres suivants sont $4, 2, 1$ puis la suite se répète en boucle. Expérimentalement, la suite de syracuse atteint toujours la valeur 1, quel que soit l'entier p de départ mais cela n'a jamais été démontré.

1. Écrire une fonction `syracuseNext(u:int)->int` qui prend en argument un entier et renvoie le terme suivant dans la suite de syracuse. On prendra soin de renvoyer un entier et non un flottant en utilisant la division appropriée le cas échéant.
2. Écrire une fonction `syracuseTempsDeVol` qui prend en argument un entier p correspondant à u_0 et renvoie le plus petit entier k tel que $u_k = 1$. Cela vaut 17 si $u_0 = 15$ et 46 pour $u_0 = 127$
3. Écrire une fonction `syracuseListe` qui prend en argument un entier p et renvoie sous forme d'une liste les nombres composant la suite de syracuse tel que p est le premier terme de la suite et que l'on s'arrête dès que l'on atteint 1.
4. Écrire une fonction `syracuseTempsDeVolEnAltitude` qui prend toujours en argument un entier correspondant au premier terme de la suite et renvoie le plus petit indice n tel que $u_{n+1} < u_0$. Cela vaut 10 si $u_0 = 15$ et 23 pour $u_0 = 127$
5. Écrire une fonction `syracuseAltitudeMax` avec le même paramètre et qui renvoie la valeur maximale atteinte par la suite. Elle vaut 160 si $u_0 = 15$ et 4372 pour $u_0 = 127$

Exercice n°6 On dispose d'une liste dans laquelle on souhaite supprimer certains éléments. Pour les exemples, on prendra $L = [0,1,1,0,1,1,1,0]$ dont on cherche à supprimer les 1. On devrait donc obtenir après modificiation [1] $L = [0,0,0]$.

1. Réaliser une boucle for dans laquelle on supprime à l'aide de "del" les éléments qui valent 1. Expliquer pourquoi cela ne marche pas (on pourra ajouter des prints ou aller sur python tutor).
2. Nettoyer la liste avec une boucle while, on fera attention à bien gérer les indices lors d'une suppression pour éviter les problèmes vus à la question précédente.

Exercice n°7 Soit a,b deux entiers naturels, tels que $b \neq 0$. On cherche à implémenter la division euclidienne de a par b à l'aide de successions de soustractions.
L'idée est la suivante :

— si $a < b$ on arrête : le quotient vaut 0 et le reste vaut a.
— si $a \geqslant b$ on remplace $a \leftarrow a - b$ et on recommence.
— On obtient à la fin le reste, et le quotient est donné par le nombre de fois où l'on a enlevé b à a.

Tester votre algorithme sur plusieurs couples d'entiers et vérifier avec les fonctions déjà implémentées $\%$ et $//$.

Exercice n°8 Algorithme d'Euclide. Soit a,b deux entiers naturels, tels que $b \neq 0$ et $a \geqslant b$. On cherche le plus grand diviseur commun à a et à b. L'algorithme est le suivant :

— On calcule le reste de la division euclidienne de a par b (au choix : soit l'algorithme par soustractions successives, soit $\%$)
— Si $r = 0$ alors le pgcd vaut b.
— sinon on remplace a par b et b par r et on recommence l'opération.

les mathématiciens ont prouvé que cet algorithme s'arrêtait toujours.

IV À faire pour la prochaine séance

Exercice n°9 Soit la suite définie par $u_0 = 2$ et $u_{n+1} = 1 + 2u_n^2$. Définir une fonction `seuil(k)` qui renvoie le plus petit entier m tel que $u_m \geqslant k$.

Exercice n°10 On se propose de programmer une fonction trouvant la partie entière du logarithme à base 10 d'un nombre x. On propose l'algorithme suivant : on suppose que $x \geqslant 1$. On cherche ensuite le plus grand entier n tel que $10^n \leqslant x$. Programmer le en python.

Exercice n°11 On se propose de trouver la décomposition en base 2 d'un nombre n en utilisant l'algorithme suivant : on crée une liste vide, tant que n est strictement positif, on ajoute $n\%2$ à la fin de la liste, on divise ensuite n par 2 (division entière). Programmer une fonction réalisant cela.

Exercice n°12 On considère la suite logistique définie par $x_{n+1} = \mu x_n(1 - x_n)$ avec μ un paramètre dans $[0,4]$ et $x_0 \in]0,1[$ (on prendra $x_0 = 0,37$ par exemple). Programmer une fonction `limite(mu,x_0)` qui estime la limite de la suite de paramètre μ et de premier terme x_0. On estimera que la limite est atteinte si $|x_{n+1} - x_n| \leqslant 10^{-6}$.

1. ici, il est en fait plus simple de programmer le remplissage d'une autre liste, mais pour cet exercice on souhaite utiliser **del** et modifier directement L.