Soutien en algorithmique et programmation

Séance 2 : itérations simples et traitements à la volée

Itérations simples

On commence par utiliser un itérateur simple (avec le mot-clé `for`). Créez un fichier `fact.py` et implantez-y :

- une fonction `fact(val)` qui calcule (de façon itérative) et renvoie la factorielle de `val` ;
- une fonction principale qui lit un entier entré par l'utilisateur, vérifie qu'il est bien positif ou nul, et affiche la factorielle de cet entier (calculée avec la fonction `fact`).

On va utiliser ensuite une boucle traditionnelle (avec le mot-clé `while`). Créez un fichier `pgcd.py` et implantez-y :

- une fonction `pgcd(val1, val2)` qui calcule et renvoie le Plus Grand Commun Diviseur des entiers strictement positifs `val1` et `val2` ;
- une fonction principale qui lit deux entiers, vérifie qu'ils sont bien strictement positifs, et affiche le PGCD des deux entiers en utilisant la fonction `pgcd`.

On utilisera pour calculer le PGCD l'algorithme classique par soustractions successives, que l'on peut décrire en pseudo-code sous la forme suivante :

```
tant que val1 ≠ val2 faire
    si val1 < val2 alors
        val2 = val2 - val1
    sinon
        val1 = val1 - val2
    fin si
fin faire
```

Traitement à la volée d'une suite de caractères

On va ensuite écrire un programme très simple qui lit une succession de caractères entrés par l'utilisateur et qui s'arrête dès qu'il a vu passer `NBR` voyelles (on posera `NBR = 5` par exemple). Le programme affichera un message et le nombre de voyelles déjà vu à chaque fois qu'il verra passer une nouvelle voyelle. Une trace d'exécution possible du programme pourra être :

```
Entrez la suite de caractères à analyser (entrée après chaque caractère) :
> z
> a
Voyelle : 'a' (1/5)
> r
> t
> y
Voyelle : 'y' (2/5)
```

```
> e
Voyelle : 'e' (3/5)
> u
Voyelle : 'u' (4/5)
> d
> f
> g
> o
Voyelle : 'o' (5/5)
```

Traitement à la volée d'une suite d'entiers

On va écrire maintenant un programme lisant une suite d'entiers positifs et détectant les *extremums locaux* de cette suite au fur et à mesure qu'on saisit les valeurs. Un extremum local est :

- soit une valeur aux bornes de la suite (i.e. la première et la dernière valeur) ;
- soit une valeur x telle que $x_{-1} < x > x_{+1}$;
- soit une valeur x telle que $x_{-1} > x < x_{+1}$.

Le programme va donc lire des entiers, s'arrêter dès qu'on saisit une valeur négative, et afficher le nombre d'extremums détectés. Pour faciliter la mise au point du programme, on affichera aussi un message dès qu'on verra passer un extremum dans la suite. Evidemment, il n'est pas possible de raisonner sur les éléments futurs (pas encore saisis) de la suite : on détectera donc les extremums avec un coup de retard, vu qu'on devra attendre d'avoir l'élément x_{+1} pour savoir si x est un extremum. Dans le programme, on travaillera avec l'élément courant (`cour`), le précédent (`prec`) et le précédent du précédent (`prec_prec`) pour plus de clarté.

On donne quelques exemples pour illustrer ce que doit faire exactement le programme :

- la suite vide n'a pas d'extremum local ;
- une suite constante (par exemple : 1 1 1 1 1 1 1) a un seul extremum local ;
- une suite monotone (par exemple : 1 1 2 2 2 3 4 5 5 5 6) a deux extremums locaux ;
- la suite 1 1 1 2 2 2 3 3 3 2 1 0 a trois extremums locaux ;
- la suite 1 2 3 2 1 2 3 a quatre extremums locaux ;
- la suite 1 2 3 2 1 2 1 a cinq extremums locaux.

On voit grâce à ces exemples que le programme doit ignorer les bégaiements (c'est à dire les valeurs consécutives égales). Par exemple, on traitera la suite 1 2 2 3 3 4 exactement comme la suite 1 2 3 4.

Il est souvent compliqué de chercher à résoudre ce type de problèmes en considérant tous les cas possibles. Ici, on voit que :

- la suite vite et la suite constante ne sont pas régulières, dans le sens qu'on ne peut pas considérer qu'il y a au moins 2 extremums (le premier et le dernier entier de la suite) et ainsi ne pas avoir à traiter le cas des valeurs aux bornes ;
- les bégaiements compliquent le calcul car on doit les ignorer ;
- vu qu'on va devoir tester l'élément courant, le précédent et le précédent du précédent, les suites de tailles 1 et 2 sont des cas particuliers.

Pour commencer, on va donc considérer qu'on travaille sur une suite ayant au moins 3 éléments, sans bégaiements et non-constante, et on relâchera ensuite au fur et à mesure ces contraintes pour compléter le programme et arriver au résultat final. En partant de cette hypothèse simplificatrice, on peut donc initialiser le nombre d'extremums à deux. Puis on ajoutera au fur et à mesure :

- la gestion de la suite vide ;

- la gestion de la suite constante ;
- la gestion des bégaiements.

Créez un fichier `extremums.py` et implantez-y :

- une fonction `extremum(prec, cour, suiv)` qui détecte si `cour` est un extremum en affichant un message et en renvoyant `True` si c'est le cas, et en renvoyant `False` sinon ;
- une fonction principale qui lit les entiers et utilise la fonction `extremum` pour détecter les extremums locaux et calculer leur nombre, qu'elle affiche à la fin.

On rappelle un schéma de boucles qui pourra être utile pour cet exercice :

```
while True:
... # on a besoin de lire une valeur au clavier pour calculer la condition
if condition:
break # on sort de la boucle ssi la condition est True
... # on fait quelque-chose avec la valeur lue si on n'est pas sorti
```

Ce schéma et le mot-clé `break` permettent en général d'éviter l'utilisation abusive de booléens dans les boucles.

Elements de correction

On peut commencer par une version simplifiée qui ne gère pas les cas particuliers :

```
def extremum(prec, cour, suiv):
    """
      Verifie si cour est un extremum local et affiche un message si oui
    """
    if ((prec < cour) and (cour > suiv)) or ((prec > cour) and (cour < suiv)):
        print(" => extremum local :", cour)
        return True
    return False

def lire_entier():
    """
      Lit un entier
    """
    return int(input("> "))

def main():
    """
      Fonction principale
    """
    prec_prec = lire_entier()
    prec = lire_entier()
    nbr = 2
    while True:
        cour = lire_entier()
        if cour < 0:
            break
        if extremum(prec_prec, prec, cour):
            nbr += 1
        prec_prec = prec
        prec = cour
    print("Nombre d'extremums locaux :", nbr)
```

Ensuite on rajoute simplement la gestion des différents cas particuliers dans la fonction principale :

```
def main():
    """
```

```
  Fonction principale
"""
prec_prec = lire_entier()
if prec_prec < 0:
    print("La suite vide n'a pas d'extremum local")
    return
while True:
    prec = lire_entier()
    if prec != prec_prec:
        break
if prec < 0:
    print("La suite constante a un seul extremum local")
    return
nbr = 2
while True:
    cour = lire_entier()
    if cour < 0:
        break
    if cour != prec:
        if extremum(prec_prec, prec, cour):
            nbr += 1
        prec_prec = prec
        prec = cour
print("Nombre d'extremums locaux :", nbr)
```