Algorithmes élémentaires

Méthodes de programmation

1 Algorithmes élémentaires opérant par boucles simples

Il est question ici d'écrire des scripts permettant de valider votre capacité à écrire des algorithmes élémentaires, utilisant des boucles simples (non imbriquées).

1.1 input, print, if

1.1.1 input

La commande « a=input("message") » a pour conséquence d'afficher le « message » dans la console (« Shells »). Pyzo attend ensuite la saisie de données directement dans la console. Les données saisies au clavier sont alors stockées sous forme d'une **chaîne de caractères** dans la variable « a ».

Si un programme nécessite la saisie d'un nombre, il faudra impérativement convertir la chaîne de caractères saisie grâce à la commande int() ou float()...

1.1.2 print

La commande « print(a) » permet l'affichage, au niveau de la console, du contenu de la variable « a », quel que soit le type de cette variable (nombre, chaîne de caractères, liste, ...).

— La commande « print » peut être utilisée avec plusieurs paramètres d'affilée : « print (a, b, c) » imprimera sur une même ligne les trois contenus des variables, séparés par un espace.

— On peut bien évidemment imposer le nombre de chiffres affichés après la virgule pour un float.
La commande « print(' :.4f'.format(pi)) » affiche la valeur de π avec 4 chiffres après la virgule (constatez par la même occasion que la valeur est arrondie, et non pas juste tronquée)

— De la même façon, on peut imposer un format scientifique pour un float. La commande « print(' :.3e'.format(0.000404)) » affiche « 4.040e-04 »

1.1.3 if

Rappel de la Syntaxe :

```
if condition :
        séquence d'instructions 1
else :
        séquence d'instructions 2
```

où :

- « condition » est une expression logique dont le résultat peut être vrai ou faux
- « séquence d'instructions 1 » est la séquence d'instructions à exécuter dans le cas où « condition » est vraie et « séquence d'instructions 2 » est la séquence d'instructions à exécuter dans le cas où « condition » est faux.

Voici les syntaxes principales pour écrire les conditions :

if condition :	if X > Y :	if X < Y :	if X >= Y :	if X <= Y :	if X == Y :	if X != Y :
teste ...	si X > à Y	si X < à Y	si X > ou égal à Y	si X < ou égal à Y	si X égal à Y	si X différent de Y

Activité 1 **Année bissextile, ou pas ?**

Ecrire un programme qui demande à un utilisateur de rentrer une année quelconque, et qui lui précise en retour si celle-ci est bissextile ou pas.
Pour information, pour qu'une année A soit bissextile, A doit être divisible par 4, mais ne doit pas être un multiple de 100 à moins d'être un multiple de 400.

Aide : a//b donne le **quotient** de la division euclidienne de a par b, et a%b donne le **reste**.
Ainsi, 10//2 donne 5, et 10%2 donne 0.

1.2 Instructions while, for

Un des intérêts de la programmation est de pouvoir répéter sans se tromper, rapidement et aussi souvent que l'on veut une certaine instruction, ce que l'être humain ne peut faire qu'avec beaucoup de patience, de temps et sans aucun doute : des erreurs... Il existe deux instructions permettant la répétition d'opérations, chacune d'elle possédant ses propres avantages et inconvénients.

1.2.1 while

Syntaxe :

```
while condition :
    séquence d'instructions
```

où :

- « condition » est une expression logique dont le résultat est vrai (True ou 1) ou faux (False ou 0)
- « séquence d'instructions » est l'ensemble des opérations à effectuer tant que « condition » est vraie.

Attention : Si la séquence d'instructions ne permet pas la modification de la condition, alors c'est le début d'une boucle sans fin, et l'ordinateur continuera de ramer tant que vous ne tuerez pas le programme en cours d'exécution (fermez la console python dans ce cas).

De manière générale, while est utilisé lorsque l'on ne connaît pas à l'avance le nombre de répétitions à réaliser. C'est la réalisation de la condition qui permet de sortir de la boucle.

Activité 2 **Calcul de légende**

Une légende de l'Inde ancienne raconte que le jeu d'échecs a été inventé par un vieux sage, que son roi voulut remercier en lui affirmant qu'il lui accorderait n'importe quel cadeau en récompense. Le vieux sage demanda qu'on lui fournisse simplement un peu de riz pour ses vieux jours, et plus précisément un nombre de grains de riz suffisant pour que l'on puisse en déposer 1 seul sur la première case du jeu qu'il venait d'inventer, deux sur la suivante, quatre sur la troisième, et ainsi de suite jusqu'à la 64^{e} case.
Écrire un script (dans un fichier .py correctement nommé et sauvegardé) qui affiche le nombre total de grains à déposer sur l'ensemble des 64 cases du jeu sous forme d'entier, puis en écriture scientifique avec 2 chiffres significatifs.

Activité 3 **Approche d'une limite de série**

Écrire un script (dans un fichier .py correctement nommé et sauvegardé) qui affiche le nombre d'entiers nécessaires pour que la somme des inverses des carrés des premiers entiers approche $\frac{\pi^2}{6}$ à 0,01 près.

Aide : pour que Python connaisse π, il faut exécuter `from math import pi`.

Activité 4 **Jeu de la devinette, à l'envers**

Le jeu de la devinette « classique » est le suivant : l'ordinateur tire un entier au hasard entre 1 et 100, et le joueur, en encadrant cet entier, doit deviner cet entier en fonction des réponses de l'ordinateur.
A vous de construire le jeu complémentaire, tout aussi classique : l'ordinateur doit deviner un nombre fixé par l'utilisateur. L'ordinateur doit proposer des entiers entre 1 et 100 jusqu'à ce que l'entier proposé soit égal à celui choisi par l'utilisateur. L'utilisateur guide l'ordinateur en répondant "+" si l'entier choisi est plus grand que l'entier proposé, "-" si l'entier choisi est moins grand que l'entier proposé, et "=" s'il s'agit du bon entier.

Aide : après avoir importé la bibliothèque `random` par `from random import *`, on peut obtenir un entier aléatoire entre a **inclus** et b **inclus** par `randint(a,b)`.

1.2.2 for

Elle n'est utilisable que lorsque l'on connaît à l'avance le nombre de répétitions à réaliser.
Syntaxe :

```
for indice in ensemble :
    séquence d'instructions
```

où :

- « indice » est une variable appelée l'indice de la boucle qui prend successivement les valeurs contenues dans la donnée composite « ensemble » (on développera plus tard la notion de donnée composite = entité qui rassemble dans une seule structure un ensemble d'entités plus simples : chaîne de caractères, liste, range, tuple, dictionnaire, ...).
- « séquence d'instructions » est l'ensemble des opérations à réaliser pour chacune des valeurs prises par l'indice de la boucle. On parle aussi du corps de la boucle.

Cas pratique très fréquent : l'ensemble contient une liste de n entiers, généralement compris entre 0 et n-1, avec un écart d'une unité entre deux entiers, c-à-d un pas de 1. Pour créer l'équivalent de ce type de liste, la fonction prédéfinie « range(valeur initiale (inclue), valeur finale (exclue), pas) » est très pratique, avec possibilité d'omettre le premier et le troisième argument de la fonction (valeurs par défaut : 0 pour la valeur initiale, 1 pour le pas).

Activité 5 **La légende, le retour ...**

Reprendre le calcul de légende en utilisant la commande « for » au lieu de la commande « while ». Attention à ne pas écraser l'ancienne version ...

Activité 6 **Calcul de e**

Rédiger un script qui calcule $e = \lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{i\,!}$ en demandant au préalable jusqu'à quelle valeur de « n » l'utilisateur souhaite aller. On rédigera deux versions :

1. Version simple : en utilisant la fonction prédéfinie « factorial » du module math.
2. Version plus complexe : sans utiliser la fonction « factorial ».

2 Algorithmes opérant par boucles dans un tableau unidimensionnel

2.1 Rappels sur les listes

Une liste est une séquence ordonnée et indexée d'objets variés, facilement modifiable. Le premier élément de la liste a pour indice 0. Les listes sont définies entre crochets. Elles sont concaténables, multipliables par un entier, et on peut « extraire » la partie de notre choix grâce au slicing.

```
>>> maliste=['élément 0', 1, 2 ,'élément 3']

>>> maliste+[3,2]
['élément 0', 1, 2, 'élément 3', 3, 2]

>>> maliste*2
['élément 0', 1, 2, 'élément 3', 'élément 0', 1, 2, 'élément 3']

>>> maliste[1:3]
[1, 2]

>>> maliste[:-1]
['élément 0', 1, 2]

>>> maliste.append(1)

>>> maliste
['élément 0', 1, 2, 'élément 3', 1]

>>> maliste.remove(1)

>>> maliste
['élément 0', 2, 'élément 3', 1]
```

```
>>> maliste.insert(0,"coucou")

>>> maliste
['coucou', 'élément 0', 2, 'élément 3', 1]

>>> maliste.index(2)
2

>>> maliste.pop(0)
'coucou'

>>> maliste
['élément 0', 2, 'élément 3', 1]

>>>
```

2.2 Recherche du maximum (et de maxima secondaires) dans une liste

On considère une liste donnée. Le travail consiste à coder des fonctions permettant d'extraire le maximum de la liste, et d'extraire également le n^{e}maximum de la liste.

Vous aurez à compléter un fichier préalablement rempli : `Maxi_et_Maxi_secondaires_a_completer.py`. Ce fichier contient, entre autres lignes. de code, la liste sur laquelle vous allez travailler.

Activité 7 **Ecriture d'une fonction**

nom : `maximum`
arguments : L (list)
effet : doit renvoyer le maximum de la liste L.

Testez ensuite votre fonction.

Activité 8 **Ecriture d'une fonction**

nom : `nieme_maximum`
arguments : L,n (list,int)
effet : doit renvoyer le n^{ème} maximum de la liste L.

Testez ensuite votre fonction.

2.3 Valeurs d'un polynôme

On considère un polynôme écrit sous la forme $P(x) = \sum_{i=0}^{n} a_i.x^i$. On suppose que les coefficients du polynôme sont contenus dans une liste. Par exemple, le polynôme $P(x) = 1 + 2,3.x^2 + 4,7.x^3$ est caractérisé par la liste des coefficients `coeff_polynome=[1.0,0,2.3,4.7]`.

Le but de l'exercice est de coder une fonction permettant d'évaluer la valeur du polynôme en une valeur de x, connaissant les coefficients du polynôme.

Activité 9 **Ecriture d'une fonction**

nom : `polynome`
arguments : coeff,x (list,float)
effet : doit renvoyer la valeur de P(x).

Testez ensuite votre fonction.