REVISIONS PYTHON

Variable, affectation, types

Variables, affectation

En informatique, une variable peut être vue comme une case mémoire:elle est repérée par un nom (c'est l'adresse à laquelle la donnée est rangée dans la mémoire de l'ordinateur) et elle possède un contenu (la donnée en elle-même). Lorsqu'on enregistre une donnée dans une variable, on dit qu'on affecte la donnée à la variable. Pour cela, on utilise la syntaxe :

```
>>> x=y
```

où x est le nom de la variable et y le contenu.

Le nom de la variable :

- est une séquences de lettres et de chiffres qui doit toujours commencer par une lettre.
- ne peut pas utiliser les accents ou les caractères spéciaux (&,$,@, etc...) sauf le symbole _.
- est sensible à la casse.
- Doit être court et explicite.
- Ne peut pas être l'un des mots-clés réservés par le langage (`and, del, from, none, true, false, as, elif, global, nonlocal, try, assert, else, if, not, while, break, except, import, or, with, class, False, in, pass, yield, continue, finally, is, raise, def, for, lambda, return`).

Types de variables et opérations

Les données numériques :

on distingue le type entier (*int*) et le type *float* (les « réels » ou nombres à virgule flottante). Attention, en Python, le séparateur des décimales est le point et non la virgule (convention anglaise comme dans beaucoup de logiciels). Pour ces données, on pourra utiliser les opérations de somme (+), de produit (*) , différences, division (/) et de division euclidienne dans le cas des entiers (// pour le quotient et % pour le reste) et puissance (**).

Les chaînes de caractères

il s'agit des lettres, mots, phrases et ensembles de symboles quelconques. On parle de « chaînes de caractères (*str*). Une chaîne de caractères est délimitée par des quotes (simples ou doubles ou triples).

Les fonctions `int(), float()` permettent de changer une chaîne de caractère en

nombre quand c'est possible.
`>>> nom[i]` permet d'accéder à l'élément d'indice `i` dans la chaine `nom`.
`>>> nom[-1]` affiche le dernier élément de la chaîne.
`>>> nom [n:p]` extrait tous les caractères entre l'indice `n` et l'indice `p-1`. Sans n, on extrait tout le début et sans p on extrait toute la fin de la chaîne.
`>>> chaine1+chaine2` est l'opération de concaténation des deux chaînes.
`>>> len()` permet d'obtenir la longueur (le nombre d'éléments) dans la chaîne.
`>>> nom_de_la_chaine.split()` (transforme une chaîne de caractères en liste de ces caractères).

Les listes :

Ce sont des collections d'objets séparés par des virgules, l'ensemble étant entouré par des crochets. On peut concaténer deux listes avec le symbole +.

Quelques fonctions utiles : `len(nom_de_la_liste)` (longueur de la liste), `del(nom_de_la_liste[indice])` (supprime un élément)

La méthode `nom_de_la_liste.append(élément)` : ajoute un élément à la fin de la liste.

`nom_de_la_liste.remove(élément)` : enlève la première occurrence d'un élément dans la liste

`nom_de_la_liste.sort()` : classe les éléments

Les booléens :

Dans cette catégorie, on ne trouve que deux données, vraie (`TRUE`) et faux (`FALSE`). Ce type de donnée permet de créer des conditions. Les symboles pour créer des données booléennes sont : == (condition d'égalité), `!=` (différent), `>` (strictement plus grand), `<` (strictement plus petit), `<=` (inférieur ou égal), `>=` (supérieur ou égal), `or` (la condition est vraie si l'une des deux condition est vraie), `and` (la condition est vraie si les deux conditions sont vraies).

Les fonctions :

Certaines fonctions sont prédéfinies dans PYTHON. Par exemple : `input()`, `int()` et `print()`. D'autres peuvent être définie par la syntaxe :

```
>>> def nom_fonction(arg1,  arg_facultatif1 = donnée_par_défaut...) :
>>>         bloc d'instructions
>>>         return
```

Modules

La version de base de Python contient certaines fonctions usuelles déjà programmées

comme `input(), print(), int(), str(), list(), type()...`
Mais il est possible d'utiliser d'autres fonctions qui sont programmées dans des « bibliothèques » que l'on appelle modules en Python, c'est à dire des fichiers qui regroupent des ensembles de fonctions et des constantes. Pour obtenir des informations sur ces fonction, on peut utiliser la fonction `help()`.
Pour ouvrir un module, on peut utiliser les syntaxes :

```
>>> from nom_du_module import * #importe tout le module
>>> from nom_du_module import fonction #importe seulement la fonction
>>> import nom_du_module as diminutif #attribue un diminutif au module, l'appel à une fonction s'écrit alors diminutif.fonction()
```

Module math

permet d'accéder à des fonctions mathématiques comme `sqrt()` (la racine carrée) ou `sin()` (le sinus) et à des constantes comme `pi` (le nombre pi).

Module random

Il permet d'accéder à des fonctions liées aux probabilités comme `random(), randrange()`

Module numpy

Il permet d'importer des fonctions qui créent ou modifie des matrices, comme `array(), shape(), zeros(), ones(), random(), exp(), linspace(deb,fin,nb_de_points)...`

Module matplotlib.image

Ce module permet d'importer, traiter et sauvegarder des images. En Python, les images sont soit en noir et blanc, soit en RVB (rouge, vert,bleu). Dans le premier cas, c'est un tableau à deux dimensions et chaque valeur est un entier entre 0 et 255 représentant une nuance de gris (0=noir, 255=blanc). Dans le second cas, c'est un tableau à trois dimensions, les deux premières donnent la taille de l'image, la troisième est un triplet donnant l'intensité de rouge, de vert et de bleu (représentée par un nombre entre 0 et 255).

On peut utiliser la fonction `imread('' nom du chemin de l'image '')` pour ouvrir une image.

Module matplotlib.pyplot

Il permet d'importer des fonctions permettant de créer ou modifier des graphiques comme
`plot(x,y)` : trace la courbe obtenue à partir des vecteurs x (abscisses) et y

(ordonnées). On peut ajouter l'option 'o' pour faire apparaître seulement des points.
`plot(x,y,x,z)` : trace deux graphiques dans la même fenêtre.
`show()` : stoppe l'exécution de python jusqu'à ce que la fenêtre soit fermée
`clf()` : ferme la fenêtre
`savefig('nom.png')` : enregistre la figure dans le répertoire courant
`ylabel('nom')` : nomme l'axe des ordonnées
`xlabel('nom')` : nomme l'axe des abscisses
`title('nom')` : donne un nom au graphique
`axis([xmin,xmax,ymin,ymax])` : rectangle de présentation
`figure(2)` : crée une deuxième fenêtre graphique (on revient à la première en tapant `plt.figure(1)`)
`grid(True)` : affiche un quadrillage.
`hist(x,c)` : trace l'histogramme correspondant à la série x en créant c classes.
`imshow(image)` pour créer l'image à afficher (faire suivre d'un `plt.show()` pour voir l'image) (`cmap='gray'` pour un niveau de gris).
`imsave(''nom du fichier'', image)` pour sauvegarder l'image en lui donnant un nom d'accès.

Instructions structurées

Instruction conditionnelle

```
>>> if condition1 :
>>>     première ligne du bloc d'instructions 1
>>>    ….
>>>    dernière ligne du bloc d'instructions 1
>>> suite du programme
```

Il existe des variantes avec les mots-clés `else` ou `elif`.
Penser à indenter les blocs d'instruction, penser aux deux points.

Boucle while

```
>>> while condition1 :
>>>    première ligne du bloc d'instructions 1
>>>    …
>>>    dernière ligne du bloc d'instructions 1
>>> suite du programme
```

La condition1 est celle qui permet à la boucle de tourner tant qu'elle a la valeur TRUE. La condition doit prendre la valeur false en un temps fini pour que le programme s'arrête. Penser à modifier la condition1 dans le bloc d'instruction.

Boucle for

```
>>> for <indice> in <liste> :
>>>    <bloc d'instructions>
>>>...
```

La boucle for se réalise en un nombre fini d'itération (correspondant à la longueur de la liste.
L'indice de la boucle peut être utilisé dans le bloc d'instruction ou non.
La liste peut contenir les entiers entre 0 et n-1 (`range(0,n)`) ou tout autres éléments. Attention `range()` renvoie une liste virtuelle. Pour l'utiliser afin de générer une liste réelle, il faut utiliser la syntaxe `list(range(0,n))`
On peut utiliser l'instruction for pour créer une liste.

Quelques conseils pour écrire un programme :

- Identifier clairement l'objectif, les données à manipuler et le type dans lequel ces données vont être codées.
- Déterminer l'ensemble des étapes nécessaires pour obtenir le résultat à partir des données.
- faire tourner le programme pour vérifier qu'il n'y a pas d'erreurs de syntaxe ou de sémantique (notamment pour les valeurs particulières).
- Commenter le programme en commençant le commentaire par le symbole `#`. Il faut commenter tout ce qui est possible afin de rendre le programme facile à lire pour vous même et pour d'autres.
- En Python, les indexation commencent à 0 et les intervalles sont semi-ouverts à droite.