Appendice 1 : compléments Python

Quelques compléments sur la syntaxe du langage Python

Généralités

Nombres

- Les nombres en notation scientifique : $5{,}1\times10^{-3}$ se note `5.1e-3`. Le « e » comme « exposant » peut être tapé comme lettre e (alpha + [e]) ou symbole EE (2nde + [EE]).
- Pour arrondir un nombre à un certain nombre de décimales, on emploie la fonction Python `round`. Ne pas confondre avec la fonction `int` ou « partie entière » qui convertit des valeurs à virgule (« flottantes ») en entiers.
- La division : `a/b` donne toujours une valeur flottante, tandis que `a//b` donne le quotient entier.
- Les puissances : l'opérateur « puissance » se code `**` : x^5 se note `x**5`.
- Le nombre `1e3` ne vaut pas 1000 (entier) mais 1000.0 (flottant). Le plus petit nombre strictement supérieur à 1 est $1+2^{-52}$: avec `X=1+2**-53`, le test `X==1` donne `True`.

Itérateur range

Syntaxe	Exemples

La fonction Python range ne crée pas de liste mais un objet interne qui peut servir pour des boucles (`for`) ou autres itérations. Pour avoir une liste il faut recourir à la fonction `list`.
Variantes : `range(a,b)` produit tous les entiers depuis `a` (inclus) jusqu'à `b` (exclu), et `range(a,b,-1)` fait de même en descendant, sous réserve d'avoir a>b.
	`>>> list(range(10))`
	`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
	`>>> list(range(1,10))`
	`[1, 2, 3, 4, 5, 6, 7, 8, 9]`
	`>>> list(range(9,0,-1))`
	`[9, 8, 7, 6, 5, 4, 3, 2, 1]`

Listes

Listes en extension

Premier moyen de définition de listes : on donne tous les éléments, *in extenso*.

Syntaxe	Exemples
`L=[élément1,élément2,… élémentN]` `L=[élément]*n` (répétition)	`L=[0,1,4,9,16,25,36,49]` `L=[0]*5` donne `[0,0,0,0,0]`

Listes en adjonction

Second moyen : on agrandit la liste petit à petit. Ainsi, l'ensemble des carrés de 0 à 7 : $\{k^2 / k\in\mathbb{N}, k\leq 7\}$.

Syntaxe	Exemple

`L=[] # liste vide`
`for k in range(N):`
`  L.append(élément_k)`
	`L=[]`
	`for k in range(8):`
	`  L.append(k**2)`

Appendice 1 : compléments Python

Listes en compréhension

Troisième moyen de définir des listes, proche de la notation mathématique des ensembles.

Syntaxe	Exemple
`L=[élément_k for k in range(N)]`	`L=[k**2 for k in range(8)]`

Manipulations de listes

Syntaxe	Exemples

Les listes de Python sont indexées à partir de 0. Si on a une liste `L`, sa longueur est donnée par `len(L)`, son premier terme est `L[0]`, le second `L[1]` et le dernier `L[len(L)-1]`.	`>>>L=[2,3,5,7]`
	`>>>L[1]`
	`3`
	`>>>len(L)`
	`4`

Pour ajouter un terme à la fin de la liste `L`, on passe par l'instruction `L.append(element)`. Pour retirer le terme de numéro `k`, on passe par l'instruction `L.pop(k)` qui supprime `L[k]` de `L` et renvoie sa valeur (la liste `L` est ainsi raccourcie).	`>>>L.append(11)`
	`>>>L.pop(0)`
	`2`
	`>>>L`
	`[3,5,7,11]`

Pour avoir la sous-liste débutant au terme d'indice `k` et sans limite on code `L[k:]`, alors que la sous-liste commençant au début et s'arrêtant au terme d'indice `k-1` se code `L[:k]`. Quant à chercher l'indice d'un terme valant `p` (s'il existe), on l'obtient avec `L.index(p)` (si aucun terme de `L` n'a la valeur `p`, cela provoque une erreur).	`>>>L.append(11)`
	`>>>L.pop(0)`
	`2`
	`>>>L`
	`[3,5,7,11]`

Pour mettre « bout à bout » (concaténer) deux listes `L1` et `L2`, on code cela : `L1+L2`.	`>>>L2=[13,17,19]`
	`>>>L+L2`
	`[3,5,7,11,13,17,19]`

Pour trier une liste `L`, on code : `L.sort()`. Pour le tri à l'envers on fait : `L.reverse()`.
Si on veut créer une liste triée `LS` à partir de `L`, sans écraser `L`, on code : `LS=sorted(L)`. Et à l'envers : `reversed(L)`.
	`>>>L.reverse()`
	`>>>L`
	`[11,7,5,3]`
	`>>>sorted(L)`
	`[3,5,7,11]`

Connaître la valeur la plus élevée dans une liste se fait avec la fonction `max`, la plus faible avec la fonction `min`, la somme avec la fonction `sum`.	`>>>min(L),max(L),sum(L)`
	`(3,11,26)`
	`>>> def moy(l):`
	`...  return sum(l)/len(l)`
	`>>> moy(L)`
	`6.5`

Parcourir une liste se fait avec le mot-clé `for` :
`for k in L:`
	`>>> p=1`
	`>>> for k in L: p=p*k`
	`...`
	`>>> p`
	`1155`

Appendice 1 : compléments Python

Chaînes de caractères

Définition

Syntaxe	Exemples

Les textes ou chaînes de caractères sont simplement des suites ordonnées de caractères. On les introduit entourées d'apostrophes simples ou doubles.	`>>> s="Six cent"`
	`>>> t=' six scies'`
	`>>> s+t`
	`'Six cent six scies'`

Un texte sur plusieurs lignes peut être introduit entouré de triples guillemets. Les fins de ligne sont incluses dans la chaîne.	`s="""Une ligne`
	`Seconde ligne"""`
	`print(s) #fin de ligne en plein milieu`

Manipulation de chaînes de caractères

L'extraction, la concaténation de chaînes de caractères fonctionnent comme pour les listes, excepté `pop` et `sort` qui ne fonctionnent pas.

Les chaînes de caractères, comme les listes, peuvent être « parcourues » par itération au moyen de boucles `for`.

Fonctions

Logarithme

Attention : la fonction « logarithme népérien » est notée `log` en Python.

Enchaînements de fonctions

Le langage Python permet d'écrire des fonctions appelant d'autres fonctions et ainsi de suite (jusqu'à un certain niveau). L'usage intensif de fonctions rend le code plus lisible et adaptable, à l'opposé du « code-spaghetti » que l'on voit trop souvent sur les calculatrices.

Une fonction peut s'appeler elle-même, cela s'appelle de la « **récursivité** » … à condition de ne pas aboutir à une boucle infinie. Voici deux exemples :

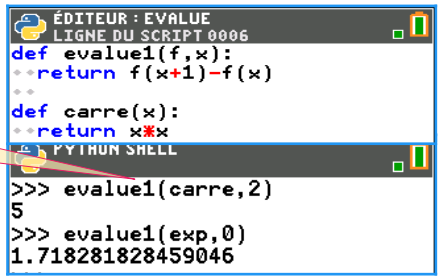
Multiplication	Puissance

`def mul(x,y) :`
`  if x==1 :`
`  return y`
`  else :`
`  return mul(x-1,y)+y`
Cette fonction réalise la multiplication de deux nombres entiers naturels (par additions successives) ; cependant, si x n'est pas entier ou est négatif, la boucle devient infinie, amenant une erreur à l'exécution.
	L'appel de la fonction `puiss` pour calculer les puissances d'un nombre crée des appels « en cascade » jusqu'à ce que `puiss(x,0)` renvoie 1. La variante `ppuiss` (mettant à profit la parité de l'exposant pour raccourcir les calculs) est plus performante ; elle montre aussi qu'une fonction peut s'appeler elle-même en plusieurs endroits.
	ÉDITEUR : PUISS
	LIGNE DU SCRIPT 0005
	`def puiss(x,n):`
	`  if n == 0:`
	`  return 1`
	`  return x*puiss(x,n-1)`
	`def ppuiss(x,n):`
	`  if n == 0:`
	`  return 1`
	`  if n % 2 == 0:`
	`  return ppuiss(x*x,n//2)`
	`  return x*ppuiss(x,n-1)`

Appendice 1 : compléments Python

Une fonction peut être passée comme paramètre à une autre fonction. L'exemple ci-contre confirme cette possibilité.

Ici carre désigne une fonction Python.



Bibliothèque (ou module) random

Quand on importe le module random, on obtient quelques fonctions Python utiles pour simuler des lois de probabilité, notamment :

Fonctions	Exemples

`randint(a,b)`
Tire au hasard un nombre entier compris entre a et b (inclus).
	`>>> randint(0,1)`
	`0`
	`>>> randint(0,1)`
	`1`

`choice(liste)`
Tire au hasard un élément de la liste donnée.
	`>>> choice([-1,1])`
	`-1`
	`>>> choice([-1,1])`
	`1`

`random()`
Tire au hasard un nombre compris entre 0 et 1 suivant une « distribution uniforme ».
	`>>> random()`
	`0.3208598777888272`
	`>>> random()`
	`0.4957570280169452`