Programmation Python

"Le nécéssaire sans le superflu"

Auteur : Sophocle

Release : Juin 2006

SOMMAIRE

1/Introduction.....

2/Dictionnaire de la Programmation.....

3/Qu'est ce que Python.....

4/Les Outils Nécéssaires.....

5/Les Variables.....

6/Les Dictionnaires.....

7/Les Opérateurs.....

8/Les Chaines de Caractères.....

9/Les Instructions.....

10/Les Fichiers.....

11/Les Fonctions.....

12/Les Classes.....

13/Les Librairies.....

14/Corrigés des Exercices.....

15/Conclusion.....

"Serpent arboricole, le Boa émeraude d'Amérique du Sud, ou Boa canin (Boa caninus), arbore une vive coloration verte qui lui permet de se camoufler aisément au sein de la végétation dans laquelle il chasse à l'affut."

I)Introduction

1)Pourquoi ce document

Une question simple équivaut à une réponse simple. J'avais écrit ce document dans l'unique but de passer le temp, et quit à passer le temp, autant le faire pour quelque chose de concret. J'ai lu un bon nombre de Howto, tutoriaux, et autres textes sur Python, et la chose qui m'a la plus interpelé, au sens propre du terme, c'est le nombre impressionant de commentaires, de lignes, de paragraphes et autres masturbation d'esprit que fournissent les auteurs dans leurs écrits.

N'avez-vous jamais lu de cour de programmation ? Moi si, et j'ai vraiment été stupéfait que ce genre de document passe des lignes et des lignes, voir des paragraphes entiers, juste pour expliquer une chose toute bête qui pourrait se voir détaillé simplement en quelques lignes. Bien entendu, ce genre de pratique nécéssite un minimum de logique, ne pas tourner autour du pot.

C'est ce que je vais essayer de faire dans ce cour de programmation Python. Lorsque j'aurais quelque chose à vous expliquer, je le ferais court, simple et précis. Il n'y a nul besoin de taper tout un livre sur : « L'art et la manière de déclarer une variable ». C'est ce genre de pratique que font un grand nombre d'auteur de ce monde, sans le vouloir directement, mais les choses sont tout de même là.

Il n'y a qu'à voir, le cour le plus complet sur Python que j'ai pu lire dans ma vie est celui de Gérard Swinnen. Hum, quatre pages pour expliquer comment on doit déclarer une variable, c'est jolie pour passer le temp mais pour apprendre un langage, il y a tout de même des limites.

C'est pour cette raison que je vais donc vous faire un cour, court et précis. Je tiens tout de même à rappeller que ce document n'est qu'une initiation basique à Python, et non un cour des plus enrichie. Néanmoins, si vous arrivez à comprendre et à maîtriser tout ce qui est dit dans ce document, vous devriez pouvoir vous débrouiller correctement avec Python, et avoir la possibilité de faire un peu tout ce que vous souhaitez.

2)Comment lire ce texte

Ne vous en faîte pas, je ne vais pas vous ressortir les vieilles blagues que (encore) beaucoup de sites sur le Hacking, la Programmation ou autres, vous ressorte lorsqu'ils vous détails les prérequis à posséder pour pouvoir comprendre leurs documents.

Si je vous dit cela, c'est pour la raison simple que vous pouvez être débutant ou expert informatique souhaitant apprendre rapidement le Python, il vous faudra traverser le même chemin.

Les débutants préfèreront sans doute commencer leur apprentissage par le tout début, tandis que les plus connaisseurs d'entre-vous souhaiterons allez au vif des choses, et ne passerons que quelques jours à apprendre le langage en entier, quand les débutants passerons énormément de temp sur des choses banale, et c'est parfaitement logique et compréhensible.

Ce cour comporte un dictionnaire de programmation détaillé un peu plus bas dans ce document, si vous êtes débutant, je vous conseil fortement de commencer par cette partie, en revanche, si vous connaissez quelque langages ou que vous êtes un peu plus avancé dans le domaine de la programmation, vous pouvez sans nul doute commencer par le chapitre 4, voir 5.

II)Dictionnaire de la Programmation

Programmer : Action de créer des programmes, des logiciels.

Langage de programmation : Tout comme dans la vrai vie, l'ordinateur a besoin qu'on lui explique les choses, les langages de programmation sont là pour ça. En effet, il s'agit en somme d'une suite d'instructions logique que l'on écrit détaillant très précisément les actions que l'ordinateur devra suivre à la lettre.

Programmation : Mot servant à désigner l'art de créer des logiciels, d'apprendre à l'ordinateur à faire des choses.

Programmeur : Personne créant des logiciels.

Coding : Synonyme de programmation.

Coder : Synonyme de programmer.

Codeur : Synonyme de programmeur.

Compilateur : Programme servant à transformer un code source en un programme executable par le système d'exploitation.

Compilation : Action de transformer un code source en un programme executable par l'OS.

Code source : Fichier contenant le code d'un logiciel. C'est à dire les opérations que devra faire le futur programme.

Interpreteur : Logiciel spécifique servant à executer un code source. Chaque interpreteur est spécifique à chaque langage de script.

Langage compilé : Langage qui nécéssite d'être compilé par un compilateur afin d'être executé. Les exemples de langage compilé sont nombreux, mais je pense que C & C++ vous seront plus familier à lire.

Langage interprété : Aussi nommé "Langage de Script", ces langages nécéssite d'avoir l'interpréteur adéquate afin d'être executé par le système. Un exemple connue est Python.

Langage de bas niveau : Langage de programmation le plus "Simple du Monde". Ce genre de langage n'est presque plus utilisé car trop long et fastidieux à écrire. En effet, il faut à chaque fois ré-inventer la roue, reprogrammer chaque fonction, instruction [...]. Un exemple simple et connue, l'ASM.

Langage de haut niveau : Ces langages sont tout l'inverse des langages de bas niveau, c'est à dire qu'ils intègrent des centaines de fonctions et d'instructions préprogrammé qui permettent au programmeur de gagner du temp et facilite la création de son logiciel.

Débugage : Action de débuguer un logiciel.

Débuguer : Mot servant à désigner l'action de modifier un programme dans le but de le rendre plus stable et plus robuste. Ce genre de pratique s'entreprend lorsque le programmeur ne connaît pas l'origine réel d'un problème dans son logiciel.

Variable : Sert à stocker des données quelconque pour le programme. Les variables sont énormément utilisé en programmation, voir même toujours utilisé.

Dictionnaire : Ils sont un peu comme des grandes variables. En effet, les dictionnaires servent à stocker de grande quantité de données que l'on pourra récupérer au choix grâce à leur indice, c'est à dire leur "nom", en quelque sorte.

Fonction : Les fonctions sont des bouts de code préprogrammé servant à faire des choses précises, comme calculer la racine carré d'un nombre, se connecter à un serveur, résoudre une équation complexe, éteindre l'ordinateur [...]. On pourrait comparer les fonctions à des bouts de programmes, ou encore, à des "petits programmes" dans les programmes.

Instruction : Les instructions sont l'atout même des langages de programmation. En effet, les instructions sont intégrés au langage et permettent de faire des choses précise, comme des boucles, des structures conditionnels [...].

Tableau : Synonyme de dictionnaire.

POO : Programmation Orienté Objet. C'est une méthode de programmation utilisé par des langages comme le C++, le Python, le Perl et autre, et qui intègrent ce que l'on nomme des "classes".

Polymorphisme : Méthode de programmation utilisé en POO qui permet de faire adopter des comportements différents à des objets dérivants les un des autres.

Instanciation : Permet de fournir un nom à une classe afin de l'utiliser dans un code source. Autrement dit, cela permet d'appeler une classe afin d'utiliser ses fonctions internes.

Héritage : Mot servant à désigner une action simple. Cette action est de fait, et grossièrement stipulé, que les classes peuvent se "piquer" des fonctions, des variables ou autres les une aux autres.

Classe : Les classes sont très utilisé en POO. Elle sont un peu comme des grandes fonctions, intègrant elles-même d'autre sous-fonctions. Les classes servent à faire des choses simple ou complexe, mais ne sont pas forcément nécéssaire à la création d'un code source.

Librairie : Fichier intégré au langage de programmation et contenant pléthore de fonctions servant à faire des choses diverse et varié. Les librairies sont très utilisé en programation. En Python par exemple, la librairie servant à créer des interfaces graphique (GUI) se nomme "Tkinter", et "os" est utilisé pour s'interfaser avec le système d'exploitation.

Module : Synonyme de librairie.

CGI : Common Gateway Interface. Terme désignant un protocole, si on peut dire ça comme ça. En réalité, les CGI sont de simple script créer dans diverse langages (comme Perl ou Python par exemple), executé du côté du serveur et permettant de traiter des données et d'interragir avec celles-ci (Base de Donnée, Calcul, Classement, Formulaire [...]). Ces scripts sont invisible pour le client, ces derniers ne perçoivent que l'IHM (Interface Homme Machine) de ces scripts (Page Web), qui fera l'échange avec ces scripts.

Packer : Système logistique de protection logiciel permettant en somme, de rendre la tâche des Cracker et des Reverser plus complexe. Ce mot sert également à désigner l'action même d'établir cette sécurité.

Unpacker : Contraire du packer. En effet, il supprime les protections mis en oeuvre par le packer du même type, autrement dit, il établit l'action inverse du packer (Action de Réversibilité). Ce mot sert également à désigner l'action même d'enlever ces protections.

III)Qu'est ce que Python

Python est un langage de programmation interprété et orienté objet. Cela signifie qu'il n'est pas nécéssaire de compiler les codes source créer en Python pour les faire fonctionner, une simple plateforme Python suffit. Orienté objet signifie que ce langage est un langage de programmation objet, c'est à dire qu'il intègre le système de classe, comme beaucoup de langage moderne et évolué. Python est dynamique, stable, simple, portable, et énormément soutenue par toute une communauté de programmeur à travers le monde. Il sert à créer des programmes, certe, mais il peut également être utilisé comme script CGI et permettre au Webmaster d'améliorer son site web. Pour les intéréssés, et j'espère qu'il y en a, le site officiel de Python (en anglais) est : http://www.python.org

1)Points Fort

-Portabilité : Python est un langage de script, et comme tout langage de script qui se respecte, il est portable. Que vous soyez sous Unix, Windows, Mac ou autre, la seule chose dont vous aurez besoin afin de faire fonctionner des programmes Python, c'est de l'interpréteur qui est disponible sur le site officiel, ou en cherchant un peu avec Google (http://www.google.com).

-Facilité : Python est un des langages les plus simple de la planète en matière de programmation objet. C'est un langage très facile, très rapide à apprendre, et qui contient toute une foule de librairies, ce qui signifie qu'il ne faut presque plus rien écrire sois-même les choses que l'on aurait pû avoir besoin, tout est déjà disponible et très bien documenté.

-Stabilité : Pyhon est aussi très stable. Si vous faites des dizaines et des dizaines d'erreurs, vu que celui-ci est interprété, vous pourrez retoucher votre code presque directement après avoir vu celles-ci et reéxecuter le code source, sans avoir à passer par la phase de compilation, ce qui fait gagner énormément de temp sur les gros projets. L'interpréteur est aussi capable de detecter si il y a des erreurs de programmation, cela signifie que si vous oubliez de mettre les « : » après un « while var1 < var2 » par exemple, avant même l'execution du programme, l'interpréteur vous dira où est l'erreur afin que vous puissiez la corriger. Qui plus est, Python est également capable de détécter des erreurs comme les débordements de tampons, évitant ainsi toute faille de sécurité potentiel.

2)Point Faible

-Interprétation : Python est un langage interprété, ce qui est plutôt ennuyeux lorsque l'on souhaite créer des logiciels, et que nous souhaitons en « cacher » la source original. En effet, lorsque l'on code un script en Python, on peut ensuite relire la source avec n'importe quel éditeur de texte traditionnel. Mais il existe tout de même quelques compilateurs Python, ce qui n'est donc pas réellement un problème.

IV)Les Outils Nécéssaires

Python est, comme je vous l'ai déjà dit, un langage intérprété, ce qui signifie que l'on a nul besoin de compilateur pour transformer une source en un programme executable. Que vous soyez sur Unix, MAC ou Windows, il vous faudra exactement le même logiciel pour faire fonctionner vos scripts...j'ai nommé : l'interpréteur Python, libre et gratuit. Il est très facilement trouvable sur Internet en cherchant un peu avec Google. Essayez tout de même d'obtenir une version assez récente, comme la 2.5 par exemple. Si vous possédez Windows, l'installation se fera sans mal, pour MAC, je ne peux vous dire car je ne connait pas ce système, pour ce qui est des systèmes Unix (Linux, BSD [...]), la plateforme est normalement préinstallé par défaut sur les distribtuions, dans le cas contraire, une recherche sur Google sera à prescrire. C'est tout ce que vous aurez besoin d'installer afin d'utiliser plainement Python. Si vous souhaitez tester des scripts CGI, vous pouvez également vous installer un serveur Apache (HTTP) en local afin de tester vos codes. Sur les systèmes à base de Kernel BSD ou Linux, Apache est normalement sur vos CD d'installation, soit téléchargeable sous forme de sources ou de packetages (RPM, DEB) sur Sourceforge. Et pour les utilisateurs des systèmes Windows, EasyPHP vous permettra d'installer Apache, avec les modules MySQL & PHP en primes.

V)Les Variables

1)Le Cour

```
>>> var1 = 5  #On déclare une variable 'var1' contenant 5
>>> var2 = 5  #On déclare une variable 'var2' contenant 5
>>> résultat = var1 + var2 #On déclare un variable 'résultat' contenant la somme de 'var1' et de 'var2'
>>> print résultat #On affiche le contenue de la variable 'résultat'
10
```

Ci-dessus, voici un très court script codé en Python, il n'a aucun but précis si ce n'est de vous illustrer mes propos. Si vous avez lu le dictionnaire en début de ce document, vous devriez savoir à quoi sert une variable (à stocker des données pour le programme). Analysons un peu ce bout de code.

La première instruction que nous voyons est « var1 = 5 », ce qui signifie « Mettre "5" dans "var1" ». Ensuite « var2 = 5 », signifiant lui aussi « Mettre "5" dans "var2" ». Pour notre troisième variable, que j'ai nommé « résultat », elle contient la somme de "var1" et de "var2", c'est à dire "5 + 5", par conséquent notre variable "résultat" doit normalement contenir "10".

On vérifie cela en affichant le contenue de cette variable grâce à l'instruction « print » de Python. En effet, « print résultat » signifie « Afficher le contenue de la variable "résultat" ». Vous pouvez nommer vos variables comme bon vous semble, mais attention tout de même, une variable nommé « VARIABLE » est totalement différente d'une variable nommé « variable », ou encore « Variable ». C'est ce que l'on nomme la « casse », et il faut vraiment faire attention à cela, car sinon...

```
>>> var1 = 5
>>> var2 = 5
>>> Résultat = var1 + var2
>>> print résultat #Aucune variable nommé 'résultat' n'a été défini !!
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in -toplevel-
    print résultat
NameError: name 'résultat' is not defined #Erreur !!
>>>
```

Cela signifie que vous n'avez déclaré aucune variable nommé "résultat", par conséquent une erreur s'affiche afin de vous prévenir. Nous pouvons également afficher le contenue d'une variable à l'intérieur même d'une chaine de caractères (String).

```
>>> TEST = 10 #On déclare notre variable 'TEST'
>>> print "Cinq et cinq font %s" %TEST #On affiche un texte (Cinq et cinq font) en insérant le contenue de notre variable grâce à l'instruction '%s'
Cinq et cinq font 10
```

J'espère que vous avez compris ce premier chapitre, car c'est une base indispensable à comprendre pour continuer ce cour correctement.

PS : Vous avez pu remarquer les phrases précédées d'un signe dièze dans les scripts, c'est ce que l'on nomme des commentaires. Vous pouvez en mettre autant que vous le voulez, c'est d'autant mieux pour vous, mais également pour ceux qui liront vos sources, cela permet de mieux comprendre un code source, et ceci est valable pour tout les langages de programmation existant. Néanmoins, les commentaires ne se déclare pas de la même façon dans tout les langages, "#" est spécifique de Python. En C et en PHP par exemple, il faudra taper "/*Commentaires*/".

Au passage, j'en profite pour signaler aux personnes ayant déjà programmé en C, ou dans des variantes de BASIC, que c'est tout à fait normal que vous n'aperceviez aucune déclaration de type de variable (String, Int, Float [...]). Cela est dû au fait que Python est dynamique, c'est à dire qu'il interprète le contenue de la viriable nommé, et en définie son type de façon totalement transparente pour l'utilisateur.

Qui plus est, Python intègre au coeur même de son langage des fonctions de transtypage, permettant de transformer une variable de type Int par exemple, en une variable de type String, à la seule condition que les deux soient compatibles. Vous ne pourrez pas transtyper par exemple, une variable de type String, avec une contenance alphabétique, en une variable de type Int, il faudra que la variable à transtyper soit un nombre. En fin bref, je pense que vous comprendrez au fil du temps...

2)Exercices

-1.1 : Ecrivez un petit script permettant d'afficher « Bonjour le monde ».

-1.2 : Ecrivez un petit script qui calculera la somme de PI (3.141592654) et de 10.

-1.3 : Ecrivez un petit script qui affichera « Dix et dix = 20 » en utilisant l'instruction "%s".

Faites ces petits exercices afin de vous entraîner, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

VI)Les Dictionnaires

1)Le Cour

```
>>> Dico = ('a' 'b' 'c' 'd' 'e') #On déclare un dictionnaire nommé 'Dico' qui contient les lettres 'a', 'b', 'c', 'd' et 'e'
>>> print Dico[0] #On affiche le premier objet du dictionnaire
a
>>> print Dico[4] #On affiche le dernier objet du dictionnaire
e
```

Voici un dictionnaire (Il s'agit en réalité d'un tuple, mais laissez, ce n'est pas très important), je l'ai nommé « Dico ». Les dictionnaires sont comme des grandes variables contenant plusieurs "choses" à l'intérieur. Pour déclarer un dictionnaire, il suffit de taper « nom_du_dictionnaire = (le contenue) », puis de taper « print nom_du_dictionnaire[indice_de_l'objet] ». En programmation, les objets des dictionnaires possède ce que l'on appel un "indice". C'est en fait un numéro commençant par 0 et se terminant en fonction de la longueur du dictionnaire. Par exemple, dans le script ci-dessus, l'indice de la lettre "c" est 2. Pourquoi ? Simplement parce que nous commençons à compter par zéro, donc en clair...

Dico[0] est égal à la lettre "a"

Dico[1] est égal à la lettre "b"

[...]

Dico[25] pourrait être égal à la lettre "z" si nous avions continué ce dictionnaire. Pourquoi ? Tout simplement parce que nous commençons par zéro, et chez moi (comme partout), 26-1 = 25 ("z" est la 26ème lettre de l'alphabet).

Comme pour les variables, vous êtes libre de nommer vos dictionnaires comme bon vous semble, du moment que vous faites attention à la casse (Majuscule/Minuscule). Mais il existe également une autre méthode, plus "appronfondie", qui permet de déclarer des dictionnaires.

```
>>> Dico = {'c': 54, 'b': 22, 'e': 15, 'd': 45} #On cré notre nouveau dictionnaire
>>> print Dico #On affiche le contenue de tout le dictionnaire
{'c': 54, 'b': 22, 'e': 15, 'd': 45}
>>> print Dico['c'] #On affiche que le contenue de 'c'
54
```

Je vous conseil fortement d'utiliser la deuxième méthode pour créer vos dictionnaires, sous peine que certaines commandes cité ci-dessous ne fonctionne pas. Le fonctionnement est des plus simple, il suffit simplement de taper : nom_du_dico = {"nom_de_objet" : nombre_de_cet_objet}. Par exemple, vous avez des fruits, et vous souhaitez faire un dictionnaire de tout ce que vous avez de disponible. Il suffit alors d'écrire : Dico = {"pêche" : 10, "banane" : 5, "cerise" : 102}. Le dictionnaire n'est pas exaustif, vous pouvez insérer plus ou moins de choses, ou plutôt, vous pouvez insérer tout ce que vous voulez, mais en respectant simplement la façon de créer un dictionnaire.

2)Les Commandes

Les commandes pour manipuler les dictionnaires sont nombreuse, mais un petit aperçu ne fait pas de mal. Je vais donc vous lister quelques commandes basique.

len() = Permet d'afficher le nombre d'objet présent dans un dictionnaire. L'utilisation est des plus simple : len(nom_du_dictionnaire). Exemple pour notre dictionnaire plus haut : len(Dico).

copy() = Permet de copier le contenue d'un dictionnaire dans un nouveau. Par exemple, pour créer un nouveau dictionnaire totalement indépendant du premier, il suffira de taper : nouveau_dico = ancien_dico.copy(). Un autre exemple, pour créer un nouveau dictionnaire à partir de notre ancien dictionnaire écrit plus haut (dans le script d'exemple), il suffira de taper : nouvoDico = Dico.copy().

del = Permet de supprimer un objet d'un dictionnaire, ou même un dictionnaire tout entier. Par exemple, écrire « del Dico », supprimera notre dictionnaire. À l'inverse, nous pouvons enlever qu'une partie du dictionnaire en tapant : del Dico ['banane'], avec cette commande, on enlève les bananes de notre dictionnaire, vu plus haut dans la deuxième méthode.

values() = Permet de retrouver toute la liste des valeurs numérique cité dans un dictionnaire. Par exemple, pour afficher l'ensemble de toute les valeurs de notre dictionnaire Dico(), il suffira simplement de taper : print Dico.values(), et toutes les valeurs numériques présentes dans le dictionnaire s'afficheront.

Voilà pour les commandes, je ne suis pas une encyclopédie, et c'est tout ce que j'ai en tête au moment où j'écris ces lignes, mais vous pouvez sûrement en trouver d'autre en cherchant sur Internet, ou dans les cours en PDF de programmation Python disponible au club des développeurs francophone : http://www.developpez.com

3)Les Exercices

-1.1 : Créez un dictionnaire contenant 20 bananes, 40 cerises et 32 pommes.

-1.2 : Créez un dictionnaire contenant 4 bouteilles, 3 bols, et 2 assiettes, et copiez celui-ci dans un nouveau dictionnaire totalement indépendant.

-1.3 : Créez un dictionnaire, contenant tout ce que vous souhaitez, copier celui-ci dans un nouveau dictionnaire, puis effacer le premier dictionnaire créé.

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

VII)Les Opérateurs

1)Le Cour

```
>>> 1 + 1 #On additionne 1 et 1
2
>>> 2 * 2 #On multiplie 2 par 2
4
>>> 4 - 2 #On soustrait 2 à 4
2
>>> 10/2 #On divise 10 par 2
5
>>> 105 % 10 #On calcul le reste de la division 100/10
5
>>> 10 < 1 #On vérifie si 10 est plus petit que 1
False
>>> 10 > 1 #On vérifie si 10 est plus grand que 1
True
>>> 10 != 15 #On vérifie si 10 est différent de 15
True
>>> 10 != 10 #On vérifie si 10 est différent de 10
False
>>> var = "Mon nom "
>>> var2 = "est "
>>> var3 = "Sophocle"
>>> print var + var2 + var3 #Concaténation des trois variables de type string
Mon nom est Sophocle
```

Les opérateurs Python sont un peu les même que partout. Je ne vais pas m'étendre, à la place, je vous fourni un petit tableau récapitulatif.

+	-	*	/	%	<	>	<=	>=	!=
Addition	**Soustraction**	**Multiplication**	**Division**	**Modulo**	**Plus petit que**	**Plus grand que**	**Plus petit ou égal**	**Plus grand ou égal**	**Différent de**

Nul besoin d'avoir fait un doctorat de mathématique pour comprendre la signification de ces signes, en revanche, je vais expliquer rapidement ce que signifie « Modulo ». Modulo est un opérateur spécifique en programmation et est utilisé pour calculer des restes.

Par exemple, le reste de 11/2 en ne conservant que des valeurs entières, est 5 (5+5 = 10 et 10 +1 = 11). Donc le reste de cette opération est 1. Ce qui signifie que si vous faite 11%2, vous obtiendrez le reste de la division 11/2, en clair, le résultat sera 1.

Il existe également d'autres opérateurs, mais vous les connaîtrez en temp voulu. Pour ce qui est des règles de calcul, elles sont les même qu'en mathématique classique, un mémo-technique que j'ai pu lire dans un cour de programmation Python est PEMDAS, pour Parenthèse, Exposant, Multiplication, Division, Addition, Soustraction.

Pour ceux se demandant comment faire des exposants, l'opérateur est "**".

Donc pour calculer le cube de 3 par exemple, il suffit d'écrire "3**3", ce qui affichera 27 (car 3x3x3 = 27). Parlons à présent des opérateurs logique. Ces opérateurs sont utilisé partout, dans tout les domaines. Ils sont utilisé dans la logique Booléenne en mathématique, mais également dans les circuits electrique, electronique [...]. Voici le tableau récapitulatif en Python.

or	*and*
A or B	A and B

Je sais, ce n'est pas très parlant comme ça, donc voici un petit exemple.

```
>>> A = 10 #Notre première variable
>>> B = 20 #Notre seconde variable
>>> demo = 10 #La variable qui va nous permettre de faire le choix
>>> if demo == A or demo == B: #Le choix conditionnel
      print "Si 'demo' est égal à 'A' OU à 'B', alors on affiche...
else:
      print "Sinon, sa affiche ce texte..."

Si 'demo' est égal à 'A' OU a 'B', alors on affiche...
```

Voici donc le petit exemple, très simple, très concret. J'explique. Nous déclarons nos trois premières variables, rien de très compliqué jusque là. Viens ensuite le test conditionnel, la ligne la plus importante, celle contenant notre opérateur logique.

En effet, nous avons une variable nommé "demo" égal à 10 (donc égal à A), nous faisons alors un test stipulant que si "demo" est égal à "A" ou à "B", c'est à dire si "demo" vaut soit 10, soit 20, alors on affiche `"Si 'demo' est égal à 'A' OU a 'B', alors on affiche..."`.

Ceci est un cas extrêment simple, alors je vous demanderais de bien le retenir et le comprendre. Je vais tout de même re-éxpliqué dans un autre "langage". Je suis conscient que vous n'avez pas encore appris les tests conditionnel en Python, mais vous pouvez visualiser leurs fonctionnements au chapitre suivant. En somme, « if demo == A or demo == B » signifie « si la variable "demo" est égal à la variable "A" ou à la variable "B", alors... ».

Voilà, j'ai pas plus simple, à vous après de vous entrainer avec l'interpréteur. Cet exemple vaut également pour l'opérateur « and », le fonctionnement est exactement le même sauf qu'au lieu de choisir entre l'un ou l'autre, il faut que "demo" soit à la fois égal à la variable "A", et à la fois égal à la variable "B" (1 & 1, par exemple).

2)Les Exercices

-1.1 : Déclarer une variable contenant le reste de la division de 3.141592654/3

-1.2 : Déclarer une variable contenant le résultat du calcul 2*2-6/2+1+10%2 et l'afficher grâce à l'instruction « print ».

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

VIII)Les Chaines de Caractères

1)Le Cour

```
>>> a = "Je "
>>> b = "me nomme "
>>> c = "Sophocle"
>>> name = a+b+c
>>> print name
Je me nomme Sophocle
```

Même sans posséder un MASTER informatique, on peut comprendre ce que signifie ce script. Le mot servant à désigner ce genre de pratique est « concaténation », par conséquent ce script à concaténé les variables "a", "b" et "c", pour n'en faire plus qu'une que j'ai nommé "name".

Au passage, les chaines de caractères en informatique sont aussi nommé "string", donc lorsque vous entendrez parlé de string en informatique, vous pouvez être sûr qu'il s'agira d'une chaine de caractères.

2)Les Commandes

Les commandes pour manipuler les chaines de caractères sont nombreuse, mais un petit aperçu ne fait pas de mal. Je vais donc vous lister celles que j'ai en tête à l'instant, comme pour les dictionnaires.

replace() = Permet de remplacer tout les caractères x par les caractères y. Le fonctionnement est simple. Supposons que nous avons une variable nommé "prénom" contenant mon nom (Sophocle). Supposons aussi que nous voulions remplacer tout les "o" par des "a", il suffira alors d'écrire « print prénom.replace ('o', 'a') ».

swapcase() = Permet de changer la casse d'une chaine de caractères. Par exemple, si nous avons une variable nommé "var" et contenant "Quel BELLE cHAINE de caractères !!", alors il suffira de taper « print var.swapcase() » pour obtenir "qUEL belle Chaine DE CARACTÈRES !!".

count() = Permet de compter combien il y a de chaine de caractère C2, dans une chaine de caractère C1. Le fonctionnement est simple, imaginez que vous avez une chaines nommé "C1", et contenant la chaine "Je ne suis pas programmeur de proffession...". Imaginez à présent que nous ayons une autre variable, mais qui contienne la chaine de caractères "programmeur". Il suffira alors de taper « print C1.count(C2) » pour obtenir le résultat qui doit normalement être 1.

int() = Permet de transformer une chaine de caractères en un nombre entier, à condition que cette chaine de caractères soit une suite de chiffre, par exemple dans une variable "a", il faudrait que celle-ci soit égal à quelque chose comme « a = "65535" », ainsi il suffira d'écrire « int(a) », pour voir s'afficher les chiffres normalement (C'est cela que j'ai nommé transtypage cité plus haut).

float() = Permet la même fonctionnalité que int(), à la différence près que cette fonction ne sert qu'à transformer les nombres flotant, c'est à dire à virgule.

Exemple : « float("3.141592654") », et nous obtenons un nombre décimal, et non plus une chaine de caractères.

len() = Permet de connaître la longueur d'une chaine de caractères. Essayez de taper « len("Voici un texte d'exemple") », l'interpréteur doit normalement vous retourner le nombre 24.

upper() = Permet de transformer une chaine de caractères en majuscule. Par exemple, si vous avez une variable "c", contenant le texte "la chaine de caractères qui est toute belle", il suffira alors de t'écrire « print c.upper() ».

lower() = Permet de faire exactement l'inverse de upper(), c'est à dire qu'au lieu de changer une chaine en majuscule, celle-ci va faire l'inverse, soit, changer une chaine de caractère en minuscule. Son utilisation est identique à la précédente.

Voilà pour les commandes, je ne suis pas une encyclopédie, et c'est tout ce que j'ai en tête à cet instant présent, mais vous pouvez sûrement en trouver d'autres en cherchant sur Internet, ou dans les cours de programmation Python disponible au club des développeurs francophone : http://www.developpez.com

2)Les Exercices

-1.1 : Déclarer une variable contenant uniquement des chaines de caractères, puis afficher ensuite le contenue en majuscule.

-1.2 : Déclarer une variable contenant votre nom, puis déclarer une autre variable contenant la longueur de la chaine de caractères présente dans la première variable. Afficher ensuite le contenue de cette dernière à l'écran.

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

IX)Les Instructions

1)Le Cour

if, elif, else

```
>>> if 4 < 2: #Si 4 est plus petit que 2, alors...
    print "4 est plus petit que 2" #On affiche '4 est plus petit que 2'
elif 4 > 2: #Sinon, on continue, et on regarde si 4 est PLUS GRAND que 2
    print "4 est plus grand que 2" #Si le résultat est positif, alors on affiche '4 est plus grand que 2'
else: #Dans tout les autres cas...
   print "Apparament l'ordinateur ne sais plus compter..." #On marque que l'ordinateur ne sais plus compter


4 est plus grand que 2
```

Ci-dessus, une instruction de type « choix conditionnel ». C'est à dire que vous demandez à l'ordinateur de faire un choix en fonction de différents paramètres que vous avez préalablement déclaré. En anglais, « if » signifie « si » et « else » signifie « sinon » (« elif » est spécifique de Python).

Une chose importante à laquelle il faut faire très attention, sont les deux point après le test, par exemple « if 4 < 2: ». Ici nous avons nos deux point, si vous les oubliez, alors Python (l'interpréteur) vous signalera une erreur de syntaxe, c'est à dire une erreur de programmation en quelque sorte.

while

```
>>> i = 0 #On déclare une variable 'i' contenant 0
>>> while i < 10: #On boucle tant que le contenue de la variable 'i' est plus petit que 10
        i = i+1 #On incrémente 'i'
        print i #On affiche 'i'
1
2
3
4
5
6
7
8
9
10
```

Ceci est un boucle. Une boucle, est un mot servant à désigner une action en programmation. Cette action, du moins pour ce script, c'est d'afficher les chiffres de 1 à 10.

Le fonctionnement est des plus simple, « while » signifie « tant que ». Pour ce petit exemple cela veux donc dire "boucler tant que i est plus petit que 10". Que fait cette boucle ? C'est très simple, nous avons une variable de départ initialisé à 0. Notre petite boucle va donc se servir de ce point de départ pour compter, la variable "i" n'est donc qu'un simple compteur de boucle, un peu comme un chronomètre si vous voulez, ou encore un compte à rebours.

Dès que "i" sera égal à 10, alors la boucle va s'arrêter. Comment fait-on pour faire passer "i" à 10 de façon régulière ? Grâce à l'incrémentation, c'est à dire augmenter une variable d'un certaine valeur. Pour faire ceci, il suffit simplement de marquer dans notre boucle : « i = i + 1 », c'est à dire que "i" est égal à la valeur de "i" + 1. Donc si nous avons notre variable "i" déclaré à 0, dès que cette ligne sera executé, "i" vaudra 1, si "i" vaut 1, lorsque cette ligne sera re-éxecuter, "i" vaudra 2 [...].

C'est ce que l'on nomme l'incrémentation. Une petite précision tout même. Il vous faudra faire très attention lorsque l'on vous écrirez une boucle, car si vous écrivez ce genre de chose...

```
>>> i = 0 #On met le compteur à zéro
>>> while i == i: #On boucle tant que 'i' est égal à 'i'
        i = i+1 #On incrémente 'i'
        print i #On affiche 'i'
```

Et bien je vous laisse découvrir par vous-même, vous verrez c'est très drôle, et encore plus quand cela arrive dans un gros projet, la seule façon d'arrêter cette boucle est d'appuyer sur "CTRL+C". Retenez juste qu'une simple petite erreur comme celle-ci, correcte dans sa syntaxe, mais érroné dans la logique, vous fera perdre énormément de temp si vous ne l'avez pas fait exprès, car il faudra établir un débugage de votre script si vous souhaitez retrouver de tel erreur dans un projet plus important, donc soyez vigilant lorsque vous programmez.

for...in

```
>>> moi = ['Je', 'me', 'nomme', 'Sophocle'] #On déclare un tableau
>>> for car in moi:
       print car #On affiche la variable 'car'
Je
me
nomme
Sophocle
```

Cette instruction permet, comme vous pouvez le constater par vous-même, de créer des boucles. Dans les faits, nous avons déclaré un tableau (dictionnaire) contenant un chaine de caractères, puis nous déclarons une variable nommé "car" qui contiendra tout le contenue du précédant tableau, et pour finir, nous affichons le contenue de cette nouvelle variable.

Le mieux pour que vous compreniez, c'est de tester par vous-même. Voici un autre script d'exemple.

```
>>> nom = 'Sophocle'
>>> for test in nom:
        print test
```

Essayez d'executer ce script, et voyez ce qu'il en résulte.

try...except

```
>>> try:
        print "Boucle 'try/except' : OK"
except:
        print "Boucle 'try/except' : WARNING"

Boucle 'try/except' : OK
```

Les « boucles » try/except sont vraiment enfantine à comprendre et à réaliser. Elles permettent, entre autre, de créer des scripts de façon à automatiser la gestion des erreurs (quel qu'elle soit). En effet, si une erreur se produit, elle sera redirigé et traité par le programmeur grâce à l'instruction « except », au lieu de laisser Python faire le travail, qui n'est pas des plus exemplaire.

Il est également possible de fournir des paramètres à « except ». Par exemple, si nous devons afficher un message d'erreur lors d'un « CTRL+C » effectué par l'utilisateur, il suffira dès lors de taper « except KeyboardInterrupt », mais il est également possible de mettre « except IOError » en cas d'erreur d'E/S avec des fichiers par exemple. C'est vous qui voyez.

raw_input()

```
>>> var = raw_input('Veuillez entrer un texte :')
Veuillez entrer un texte: Je suis Sophocle
>>> print var #On affiche 'var'
Je suis Sophocle
```

Cette fonction offre la possibilité à l'utilisateur d'entrer des données quelconque. Pour ce petit exemple, la fonction va nous servir à enregistrer dans une variable nommé "var" le texte que va entrer l'utilisateur. Puis nous affichons le contenue de cette variable, en clair, nous affichons ce que viens d'entrer l'utilisateur.

return

Cette instruction est utilisé dans les fonctions et permet de retourner un résultat en fonction de certains critères préalablement établie dans cette fonction. Je traite les fonctions dans le chapitre suivant, vous pourrez donc comprendre un peu mieux cette instruction. Un petit exemple avec des commentaires pour essayez d'expliquer celle-ci.

```
>>> def exemple(x): #On déclare une fonction nommé 'exemple' et qui contiendra un paramètre 'x'
      return x #On affiche 'x'
>>> exemple(5) #On appel la fonction 'exemple()' en donnant à 'x', la valeur 5
5 #La fonction nous retourne la valeur 5
```

Cette procédure est préférable à « print » pour les fonctions complexe, même si « print » fonctionne de la même manière, le code sera plus compréhensible. Si vous avez déjà programmé dans d'autres langages, vous devez sûrement savoir qu'en C, « return » existe également.

2)Les Exercices

-1.1 : Ecrivez un script permettant d'afficher la table de 17.

-1.2 : Ecrivez un script qui affiche toute les lettres de l'alphabet en minuscule, mais présente en majuscule dans un tableau. Par exemple, vous avez un tableau Alpha = ['A', 'B', 'C'...], vous devez créer une boucle qui permette de transformer ce tableau en Alpha = ['a', 'b', 'c'...]

-1.3 : Ecrivez un script permettant de savoir si une variable nommé PI et contenant le nombre 3.141592654, est égal à une autre variable contenant la valeur de cette dernière.

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

X)Les Fichiers

1)Le Cour

Ouvrir un fichier

Nous allons supposer ici qu'il existe un fichier nommé "TEST.txt" à la racine du disque dur. Il suffit dès lors des taper ceci afin d'ouvrir le document.

```
>>> f = open('C:/TEST.txt') #On ouvre le fichier simplement
```

Vous voyez qu'il n'y a rien de très compliqué la dedans. Ici, "f" est un nom servant à instancier l'objet « open() », vous pouvez inscrire le nom que vous souhaitez, cela ne pose aucun problème, mais il faudra tout de même ne pas l'oublier, car il sera ré-utilisé plus tard.

Ouvrir un fichier en lecture

La même chose, mais en lecture cette fois.

```
>>> f = open('C:/TEST.txt', 'r')  #On rajoute 'r', qui signifie 'read'
```

Ouvrir un fichier en écriture

Encore la même chose, mais en écriture cette fois.

```
>>> f = open('C:/TEST.txt', 'w') #On modifie le 'r' par 'w'(write)
```

Lire un fichier

À présent que le fichier est ouvert, il serait intéréssant de pouvoir y lire le contenue, voici le code source Python.

```
>>> f.readlines() #On lit le contenue du fichier tout entier
>>> f.readline() #On lit le fichier ligne par ligne
```

Vous aurez pû remarqué que nous nous servons toujours de notre "f", qui sert maintennant de "nom" (si on peut dire ça comme ça) à notre fichier.

Ecrire dans un fichier

Nous allons à présent écrire dans ce fichier.

```
>>> f.write('Ceci va être écrit dans le fichier...') #On écrit dans le fichier simplement
```

Il existe tout de même une autre possibilité pour travailler avec les tableaux, les variables ou autres.

```
>>> tab = ['Bonjour toi\n', 'Je me nomme Sophocle\n', 'Au revoir'] #On déclare un tableau
>>> f.writelines(tab) #On inscrit le contenue du tableau dans le fichier
```

Fermer un fichier

Pas la peine d'expliquer plus en détail, il suffit juste de taper ceci.

```
>>> f.close() #On suppose bien sûr avoir donné le nom 'f' à notre objet, si vous l'avez appelé 'crevette', il faudra écrire crevette.close()
```

Créer un fichier

Encore plus simple, il suffit de taper ceci.

```
>>> h = open('C:/Nouvo.txt', 'w') #On cré un nouveau fichier nommé 'Nouvo.txt' qui sera créé en racine du disque dur, il ne faut surtout pas oublier le 'w', sinon cela ne fonctionnera pas
```

Copier un fichier

Ce n'est pas réellement une fonction, mais plutôt une petite astuce qui pourrait peut-être vous aider un jour ou l'autre. Supposons que nous avons un fichier vierge nommé "Vierge.txt", et une lettre nommé "Lettre.txt", le code ci-dessous va ouvrir les deux fichiers correctement préparé, et copier le contenue de la lettre dans le fichier vierge, puis refermer tout ça.

```
>>> f = open('C:/Lettre.txt', 'r') #On ouvre en lecture
>>> h = open('C:/Vierge.txt', 'w') #On ouvre en écriture
>>> h.writelines(f) #On écrit le contenue de 'f'(la lettre) dans 'h' (fichier vierge)
>>> f.close() #On referme le premier fichier ouvert
>>> h.close() #On referme le second
```

2)Les Exercices

-1.1 : Ecrire un programme permettant de copier le contenue d'un tableau dans un nouveau fichier texte qui devra être créé par le script à la racine du disque dur, et non être déjà présent sur celui-ci.

-1.2 : Ecrire un script permettant de copier le contenue d'un fichier texte qu'aura préalablement choisie l'utilisateur dans un nouveau fichier texte vierge qui sera créé à la racine du disque dur.

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

XI)Les Fonctions

1)Le Cour

```
>>> def cube(x): #On déclare une fonction nommé 'cube' qui aura le paramètre 'x'
      cubeX = x**3 #On déclare une variable nommé 'cubeX' qui contiendra le cube du nombre de l'utilisateur
      print cubeX #On affiche cubeX
>>> cube(5) #On appel la fonction que l'on vient d'écrire en donnant 5 au paramètre 'x'
125 #Le résultat, celui contenue dans cubeX
```

Voilà donc les fonctions, ces bouts de code permettant de faire ce dont pourquoi le programmeur l'a écrit. La fonction ci-dessus permet de calculer le cube d'un nombre qu'aura préalablement déclaré l'utilisateur. Afin de déclarer une fonction, il suffit juste de taper « def nom_de_la_nouvelle_fonction(paramètre) ».

Bien entendu, les paramètres ne sont pas obligatoire, c'est à vous de choisir d'en demander ou pas en fonction de ce que devra faire votre fonction. Par exemple, si vous souhaitez créer une variable servant à contenir le nom de l'utilisateur, alors il serait préférable d'écrire.

```
>>> def dico(): #On déclare une nouvelle fonction nommé 'dico'
      namae = raw_input('Veuillez saisir votre nom : ') #On demande le nom de l'utilisateur, cette information sera stocké dans la variable 'namae'
      print "Votre nom est donc %s" %namae #On rappel la variable 'namae' grâce à l'instruction %s pour dire le nom de l'utilisateur


>>> dico() #On appel la fonction
Veuillez saisir votre nom : Sophocle
Votre nom est donc Sophocle
```

Vous voyez, rien de très compliqué ldans ce code. Vous pouvez, comme pour les variables et les tableaux, nommé vos fonction comme vous le souhaitez, mais il faut tout de même faire attention à la casse. Un dernier exemple pour la route.

```
>>> def end(x, y, z):
      return x+y+z


>>> end(2, 2, 6)
10
```

Aucun commentaire pour celui-ci, je vous laisse chercher par vous-même. Je viens donc de vous montrer comment créer vos propres fonctions, mais il serait intéréssant de pouvoir également se servir de fonction déjà présente dans des librairies, c'est ce que je vais vous expliquer dans le chapitre XII, qui est intégralement consacré aux librairies, à leurs classes et à leurs fonctions.

2)Les Exercices

-1.1 : Ecrivez une fonction qui permettra de calculer le carré d'un nombre choisie par l'utilisateur.

-1.2 : Ecrivez une fonction capable d'afficher la table de multiplication qu'aura préalablement choisie l'utilisateur dans un paramètre. Par exemple, en tapant : multiplication(10), l'utilisateur devra pouvoir visualiser la table de 10.

-1.3 : Ecrivez une fonction étant capable d'agir comme une calculatrice, par exemple, en tapant : calc(10, *, 10), la fonction devra être capable de retourner le résultat de 10x10.

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

XII)Les Classes

1)Le Cour

Rapellez-vous du dictionnaire vu en début de cour, vous devriez normalement savoir ce qu'est une classe. Les classes sont des sortes de grande fonctions qui contiennent elles-même d'autre sous-fonctions. Cette partie du cour est un peu délicate, et c'est pour cette raison que je vous demanderais d'être le plus attentif possible, et d'essayer de bien comprendre ce qui est expliqué. Enfin je dis ça, c'est vous qui voyez, je ne suis pas à votre place.

Créer une classe tout simple

Afin de pouvoir se servir des classes, il est nécéssaire d'en posséder au préalable, et pour créer une classe il suffit uniquement de taper ceci.

```
>>> class exemple:
        def bonjour (self):
            print 'Exemple de classe la plus simple du monde...'
```

Voici une classe toute neuve. Elle est vraiment très basique, et ne sert pas réellement à grand chose, mais ce n'est juste qu'un exemple de démonstration. Pour executer la fonction interne de cette classe il suffit de taper ceci.

```
>>> hello = exemple() #On instancie notre classe
>>> hello.bonjour() #On appel une de ses fonction interne
Exemple de classe le plus simple du monde...
```

Voilà, c'est tout simple. J'ai nommé mon instanciation "hello", mais vous pouvez l'appeler comme bon vous semble, ce n'est pas un problème, il faudra juste faire attention par la suite à ne pas se tromper d'instance, sinon... enfin vous verrez.

Créer une classe en utilisant un constructeur

Nous dorénavant créer des classes d'objet, voyons à présent comment les optimiser un peu. La première méthode est des plus basique, mais il existe une deuxième méthode, presque toujours utilisé, et surtout plus intéréssante.

```
>>> class presentation :
        def __init__(self): #Voici le constructeur
            self.name = 'Sophocle' #Variable contenant mon nom
            self.age = 15 #Une deuxième variable contenant mon âge
        def go(self): #Notre première fonction
            print "Je me nomme %s" %self.name
            print "J'ai %s ans" %self.age

>>> start = presentation() #On instancie la classe
>>> start.go() #On execute la fonction go()
```

```
Je me nomme Sophocle
J'ai 15 ans
```

Personnelement je vous conseil cette méthode, cela vous permettra de faire des héritages, ce qui est plutôt intéréssant.

Héritage simple

À présent, vous devriez savoir créer quelques petites classe intéréssantes, donc voyons comment mettre en oeuvre un héritage. Nous allons créer deux classes distinctes, et faire hériter la seconde classe de la première. Voici un exemple.

```
>>> class Premiere: #On déclare notre première classe
    def __init__(self, nom = 'Sophocle', age = 15): #Notre constructeur et ses variables prédéfinie dans des paramètres
        self.nom = nom #Une première variable qui pourra se faire hérité
        self.age = age #Notre seconde variable qui pourra se faire hérité
    def name(self): #Notre fonction principal
        print "Je me nomme %s" %self.nom #On inscrit mon nom...
        print "J'ai %s" %self.age #Mon age...


>>> class Seconde (Premiere): #On déclare le deuxième classe avec le nom de la classe mère entre parenthèse, ce qui permettra de faire l'héritage
    def __init__(self, nom = 'Sophocle', age = 15): #Le constructeur, avec redéfinition des contenue des variables afin de pouvoir utiliser celle de la classe parente
        Premiere.__init__(self, nom, age) #On appel notre classe mère et on redéfinit son constructeur pour pouvoir l'utiliser
    def hello(self): #Notre fonction principal
        print "Je me nomme %s" %self.nom #On se sert des variables définit dans le constructeur de la première classe, mais redéfinit en partie ici dans les paramètres du constructeur de la seconde classe
        print "J'ai %s ans" %self.age #Pareil...


>>> t = Seconde()
>>> t.hello()
Je me nomme Sophocle
J'ai 15 ans
```

N'ayez pas peur de la longueur, mes commentaires prennent beaucoup de place, c'est tout. Je sais, c'est un peu bizarre à première vue, mais avec le temp et les exercices, cela devrait rentrer sans trop de difficultés. Essayez de bien comprendre cette partie, car cela est très important en POO, c'est une base fondamental.

Je vais quand même essayer de vous refaire cela en mieux, et détailler un peu tout ça, je préfère.

```
>>> class Esclave:
        def __init__(self, nom = 'Sophocle'):
            self._nom = nom #Retenez bien cette ligne...
        def Name(self):
            print "Je me nomme %s" %self._nom #On réutilise la variable ici
>>> class Maitre (Esclave):
        def __init__(self):
            Esclave.__init__(self)
        def Hello(self):
            print "Je me nomme %s" %self._nom #Mais également ici !!
>>> T = Maitre()
>>> T.Hello()
Je me nomme Sophocle
```

Héritage multiple

À présent, passons aux choses sérieuses. Dans beaucoup de tutoriaux et autres cours de programmation Python, ils passent des pages et des pages sur ce sujet, personnelement je ne vais passer qu'un paragraphe, cela sera amplement suffisant. Enfin j'espère...

```
>>> class A:
        def __init__(self, nom = 'Sophocle'):
            self._nom = nom #Une variable ici...
        def Name(self):
            print "Je me nomme %s" %self._nom
>>> class B:
        def __init__(self, age = 15):
            self._age = age #Une autre là...
        def Age(self):
            print "J'ai %s ans" %self._age
>>> class Maitre (A, B): #Le secret réside ici
        def __init__(self):
            A.__init__(self) #Et ici...
            B.__init__(self) #Idem...
        def Hello(self):
            print "Je me nomme %s" %self._nom #Variable de A
            print "J'ai %s ans" %self._age #Variable de B
>>> T = Maitre()
>>> T.Hello()
```

Je me nomme Sophocle

J'ai 15 ans

Voyez vous, il n'y a rien de très compliqué dans tout ceci. Si vous maîtrisez tout ce qui a été dit, alors vous en avez fini avec Python. Il ne vous restera plus par la suite, qu'à fignoler un peu dans la maîtrise des librairies, mais cela n'est pas très difficile en soi. Et rappellez-vous, il n'y a qu'avec la pratique intensive que l'on apprend les choses, il ne sert à rien de lire 15 pages sur un sujet si c'est pour ne pas s'en servir par la suite, c'est une perte de temps inutile.

Au passage, n'ayez pas peur de faire des erreurs, bien au contraire, si vous arrivez à comprendre vos erreurs, vous serez à même (normalement), de ne pas les refaire et de les corriger. Et puis, il n'y a que ceux qui ne font rien que ne font pas d'erreurs dans la vie. Alors trompez-vous le plus possible !! Enfin, au début toujours, parce qu'après cela devient légèrement ennuyeux...

2)Les Exercices

-1.1 : Ecrivez un script permettant d'afficher votre profil (nom, prénom, age, ville, passion...etc...) en vous servant des méthode d'héritage simple.

-1.2 : Ecrivez une classe qui permettent d'afficher la table de multiplication de 158 en utilisant un constructeur et une fonction principal nommé 'table'. La table devra s'arrêter à 158x10, donc 1580.

-1.2 : Créez une classe qui permettent de créer des fichiers à la racine du disque dur, mais en demandant à l'utilisateur le nom qu'il souhaite donner à son fichier au préalable, il devra également avoir le choix d'en choisir l'extension (*.txt, *.py, *.swf, *.html, *.c, ...etc...).

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

XIII)Les Librairies

1)Le Cour

Cette partie du cour est peu moins délicate que celle des classes, mais elle reste tout de même assez complexe. Pour une meilleure compréhension de votre part, je vais donc « découper » ce chapitre en plusieurs parties, tout comme pour le chapitre précédant. Ce chapitre n'est pas un cour exaustif des possibilités des librairies qui vont être étudiées ici, il ne s'agit là que d'un aperçu. Si vous souhaitez réellement avoir toute les informations nécéssaire à vos besoins, je préfère que vous vous dirigiez sur la documentation de Python intégré à l'interpréteur, ou mieux encore, allez voir les codes sources des librairies, vous y verrez plus clair.

Importer des modules

Afin d'avoir la possibilité de créer des interfaces graphiques, de travailler avec les réseau, le système d'exploitation et j'en passe, il est absolument nécéssaire de posséder quelques fonctions préprogrammé, à moins que vous ne souhaitiez tout réécrire vous même. La commande qui permet de faire ce travail est :

```
>>> from nom_du_module import *
```

Mais cette commande possède un défaut, c'est d'importer la totalité de la librairie. C'est à dire que si il existe plusieurs fonctions ou classes du même nom dans d'autre modules que vous avez également importé en entier, il risquerait d'y avoir de gros problème lorsque vous allez vouloir utiliser certaines fonctions, il y aurait conflit, ou alors, il se pourrait que vous utilisiez la fonction d'un module autre que celui que vous croyiez être en train d'utiliser, et vous pourriez vous apercevoir de cette « catastrophe » uniquement lorsque vous aurez executé la fonction, donc votre script. C'est pour cette raison que je vais vous dire d'oublier tout de suite cette commande et d'utiliser à la place :

```
>>> from nom_du_module import fonction1, fonction2...
```

J'espère que vous avez compris comment cela fonctionne. Lorsque je dit « fonction1 » ou « fonction2 », cela signifie qu'il faut appeller vos fonctions de cette manière, et non taper ce que j'ai inscrit à la lettre. Il existe également une dernière méthode assez intéréssante :

```
>>> import nom_du_module
```

Cette méthode d'appel fonctionne majoritèrement bien, et ne pose presque pas de problème, mais la seconde méthode est tout de même préférable pour les gros projet afin d'éviter toute erreur potentiel.

Les modules important

Tkinter : Permet de créer des interfaces graphiques.

math : Permet de faire des calculs algébriques.

socket : Permet de travailler avec les socket.

os : Permet de travailler avec le système d'exploitation et les fichiers.

winsound : Permet de travailler avec les son.

platform : Permet de travailler avec la machine.

sys : Permet de travailler avec l'OS.

Ces libraires ne sont bien sûr pas exaustive, et heureusement. Il en existe un grand nombre, ce qui offre la possibilité de faire un peu tout et n'importe quoi avec Python, et ce, très facilement et très rapidement. Si vous souhaitez obtenir les informations nécéssaire sur une librairie précise, je vous conseil d'allez regarder l'aide de Python, intégré à l'interpréteur. Voyons donc les fonctions intéréssante que propose les modules ci-dessus.

Le fonctionnement des modules

1.Tkinter : Les Classes

Voici un petit tableau contenant un nombre convenable de classe implémenté dans le module « Tkinter », ce qui vous permettra déjà de construire des GUI (Graphic User Interface) assez satisfaisant.

Nom de la classe	*Explication de la classe*
Tk	***Permet de créer des fenêtres***
Pack	***Permet d'organiser la géométrie d'une interface***
Grid	***Permet d'organiser la géométrie d'une interface***
Place	***Permet d'organiser la géométrie d'une interface***
Button	***Permet de créer des boutons***
Canvas	***Permet de créer des canvas (espace pour disposer des éléments)***
Checkbutton	***Permet de créer des cases à cocher***
Frame	***Permet de créer des frames (espace rectangulaire pour disposer des éléments...)***
Label	***Permet de créer des label (texte, images...)***
Listbox	***Permet de créer une litsbox (boite contenant une liste de choix...)***
Menu	***Permet de créer des menus***
Radiobutton	***Permet de créer des boutons radio***
Scale	***Permet de créer des règles(barre avec un curseur...)***
Scrollbar	***Permet de créer des ascenseurs (barre de défilement...)***
Text	***Permet d'afficher du texte formatté, l'utilisateur peut également éditer ce texte***

Nom de la classe	*Explication de la classe*
Entry	***Permet de créer des espaces pour que l'utilisateur puisse entrer des données (Nom, Prénom, Adresse, IP, Port...)***
PhotoImage	***Permet d'insérer des images***
Toplevel	***Permet d'afficher une fenêtre pop-up***

2.Tkinter : Fonctionnement

Vous connaissez à présent les classes utilisés dans le module Tkinter. Voyons maintenant comment les utiliser.

Tk()

Je vais prendre le nom "fen" comme instanciation afin d'illustrer mes propos. Cette classe est utilisé afin de créer des fenêtres, et elle s'utilise le plus simplement du monde. Pour créer une nouvelle fenêtre, il suffit d'inscrire : « fen = Tk() », pour lui donner un titre : « fen.title = "Le titre que vous voulez" ». Il ne faudra tout de même pas oublier que "fen" sera le nom de votre fenêtre par la suite, donc lorsque vous appellerez d'autres classes pour faire d'autres choses, il faudra fournir le nom "fen" pour déclarer votre fenêtre au nouvel objet, afin qu'il se place au bon endroit, par exemple, pour créer un bouton dans votre fenêtre, il faudra noter : « bou = button(fen, text="test") ».

Pack()

Cette classe est encore plus simple que la précédante, il suffit uniquement d'inscrire les choses de cette manière : « objet.pack() », et l'objet sera placé automatiquement sur la fenêtre. Il existe tout de même une autre solution : « objet.pack(side = LEFT) », ce qui placera l'objet à gauche, vous pouvez changer "LEFT" par "RIGHT", ou encore par "TOP" ou "BOTTOM".

Grid()

Ma classe préféré pour positionner un objet. Elle offre un grand nombre de choix, et est très facilement manipulable. Cette classe fait environ le même travail que Pack(), à la différence près qu'il y un peu plus de choix disponible afin de placer un objet. En effet, avec Grid() vous aurez la possibilité d'insérer un objet sur un axe voulue, dans tout les sens du terme. Vous pourrez donc placer un objet voulu sur la colonne et sur la ligne que vous aurez choisie, sur une coordonné (x;y), toute la longeur d'un ligne [...]. L'appel de grid() se fait de cette manière : « objet.grid(row=0, column = 1) », vous pouvez bien évidement changer les valeurs de "row" et de "column" pour qu'il réagisse à vos besoins. Les paramètres de grid pour placer un objet sont : column, columnspan, row, rowspan, ipdy, ipdx, padx, pady, et sticky. Le mieux pour comprendre comment fonctionne cette classe est de la tester par soi-même. Je vous laisse faire.

Place()

Place(), encore une autre classe pour placer des objets à des endroits voulu. L'utilisation est toujours aussi simpliste. Il suffit de taper : « objet.place(x=50, y=50) ». Comme vous avez pu le constater, cette classe sert à placer un objet sur un axe (x;y), vous pouvez bien entendu modifier les valeurs de "x" et de "y" à vos besoins, vous n'êtes pas obliger de noter 50 comme moi.

Button()

Cette classe sert à créer des boutons, et l'utilisation est assez simple. Je vais choisir le nom "bou" pour instancier mon boutton d'exemple. En premier lieu, on déclare son bouton : « bou = Button(fen, text = "Le text ici" »). Viens ensuite la mise en place de notre objet sur notre fenêtre. Pour se faire, il faut taper : « bou.place(x=50, y=50) ». Pour cet exemple je me suis servie de la classe Place() pour placer le bouton, mais vous pouvez également utiliser les classes Grid() ou Pack().

Les valeurs que j'ai inséré ne sont là que pour vous montrer comment les classes fonctionnent, donc soyez logique et ne tapez pas aveuglément ce que j'écris. Pour la classe bouton il existe également d'autre paramètres, voici la liste : command, bg, fg, font, relief, padx, pady [...]. Il en existe d'autres, mais ceci sont les plus utilisé. Donc pour vous montrer la façon de tout utiliser : « bou = Button(fen, text = "Quitter", fg = "green", bg = "black", command = fen.destroy) ». Une fois fait, il suffit de tout mettre en place : « bou.pack() ».

Canvas()

Cette classe permet de créer des espaces permettant de placer des objets dessus. Pour ce faire, il suffit d'écrire « canv = Canvas(fen) », puis on pose tout cela avec « canv.pack() ». Il existe aussi d'autres paramètres pour les Canvas, je vous liste tout cela : bg, bd, relief, height, width [...]. Il en existe également d'autres, mais il ne sont pas vraiment très utilisé.

Checkbutton()

Pour utiliser cette classe, il suffit d'écrire « case = Checkbutton(fen, text = "Le texte ici") », vous pouvez bien sûr rajouter d'autres paramètres, en fait, cette classe fonctionne exactement comme la classe Button(), donc vous la connaissez. Une fois déclaré, il suffit de taper « case.place(x=50, y=50) », vous pouvez aussi utiliser la classe que vous souhaitez pour poser votre case à cocher, comme Grid() par exemple. Une petite précision tout de même, il existe deux fonctions importante pour cette classe, les fonctions "select()" et "deselect()".

Frame()

Pour utiliser cette classe, rien de plus simple : « fram = Frame(fen, height = 150, width = 150) », et comme toujours, il vous faudra vos propres paramètres bien entendu. Pour la mise en place, je pense que vous connaissez à présent, mais je me répète encore : « fram.pack() », ou alors la classe de positionnement que vous voulez.

Label()

Afin d'utiliser cette classe, tout comme pour les autres, c'est toujours aussi simple, il suffit de taper : « label = Label(fen, text = "Ceci est un label textuel") », puis on pose tout ça de cette manière : « label.pack() », et comme toujours, libre à vous de choisir la classe de positionnement géométrique que vous souhaitez.

Listbox()

Je ne maîtrise pas très bien cette classe, mais je peux tout de même vous proposer un aperçu. On instancie avec « box = Listbox(fen) », et pour la mise en place, il suffit de taper « box.pack() », pareil que précédement, libre à vous de choisir la classe de positionnement que vous souhaitez.

Menu()

Cette classe est très utilisé, elle permet en effet de créer des menus (Fichier, Edition, Aide [...]) sur une fenêtre. Le fonctionnement est assez simple, il suffit de taper « menu = Menubutton(fen, text='Fichier') », en supposant que votre fenêtre se nomme "fen" (comme toujours), viens ensuite la mise en forme, donc comme plus haut : « menu.pack() », ou autre, libre à vous de choisir la classe de positionnement de vos objets (Grid, Pack ou Place).

À présent, il faut ajouter des boutons dans ce menu, et pour ce faire il faudra écrire « men = Menu(menu) », vous sauter une ligne, puis vous tapez « men.add_command(label = 'Quitter', command = fen.destroy) », ensuite vous écrivez « menu.configure(menu = men) », et c'est fini. Et comme toujours, vous êtes libre de nommer vos objets comme bon vous semble, ce qu'il faut c'est faire attention à ne pas s'y perdre. Si je vous dit ceci, c'est que cela m'ai déjà arrivé dans mes débuts avec Python.

Radiobutton()

Créer des boutons radio très facilement, c'est possible avec cette classe. Il suffit juste de taper « radio = Radiobutton(fen, text = "Quitter", command = fen.destroy) », puis mettre celui-ci en place sur le fenêtre, et pour ce faire : « radio.pack() », mais vous pouvez aussi utiliser les classes de positionnement (Pack(), Grid() ou Place(), c'est vous qui voyez).

Scale()

Comment créer des règles ? Comme ceci « scal = Scale(fen) », puis on pose tout ça avec « scal.pack() ». Je sais, c'est très simple, mais il existe bien sûr d'autres paramètres possible à mettre en oeuvre pour que vos règles puissent servir à quelque chose. Voici une petite liste : bd, bg, command, cursor, digits, fg, font, from, relief, to, width, height, label, orient, length [...]. Ce sont les principaux, et je vous laisse chercher par vous-même pour ce qui est de leur utilisation afin que vous soyez entraînez à bien les utiliser. Je vous donne tout de même quelques tips rapide : fg = foreground, bg = background, font = police, width & height = géométrie, label = texte/image [...]. Vous voyez comment il faut penser à présent, donc à vous de faire le reste du travail.

Scrollbar()

Programmez vos ascenseurs en quelques lignes, rien de plus simple avec cette classe. Il suffit de taper « bare = Scrollbar(fen) », puis de le positioner grâce à « bare.pack() ». Les paramètres possible sont : activerelief, bg, bd, command, jump, orient, cursor, width, relief [...]. Voilà pour le petit aperçu, vous en apprendrez sûrement bien plus en regardant la source du module Tkinter et en vous exerçant avec l'interpréteur.

Text()

Cette classe permet, comme vous le savez sans doute, de créer du texte formaté. Ce texte pourra être édité par l'utilisateur, mais il existe également une autre utilité de cette classe, comme créer un petit bloc-note. Pour l'utiliser, il suffit de taper « txt = Text(fen) », puis « txt.pack() » pour positionner tout cela. Il existe aussi padx, pady, cursor, setgrid, font, background, foreground [...], pour ce qui est des paramètres. Je rappel que les paramètres se place dans les parenthèses comme ceci : « (fen, paramètre = valeur) ».

Entry()

Cette classe est très utile, surtout pour créer des scanner, des formulaires [...] et sommairement, tout ce qui demande certaines informations à l'utilisateur (IP, HOST, Nom, Prénom, Age, Lieux [...]). Il suffit de taper « enter = Entry(fen) », puis « enter.pack() » pour la mise en forme. Les paramètres possibles sont : bd, bg, fg, font, cursor, validate, width, show, state, relief [...]. Il en existe d'autres, comme toujours, mais ceux-ci sont les plus important.

PhotoImage()

Vous souhaitez intégrer des images dans vos programmes ? Aucun problème avec un petit canvas fait maison et la classe PhotoImage. Supposons que nous ayons une image nommé "SILE.gif" présente à la racine de notre disque dur C:/. Le code sera donc « foto = PhotoImage(file = 'C:/SILE.gif') », ensuite nous déclarons un canvas avec « canv = Canvas(fen, width = 500, height = 500) », puis nous posons notre image dans celui-ci grace à « item = canv.create_image(500, 500, image = foto) », libre à vous de modifier les valeurs bien entendu.

Toplevel()

Cette classe va vous permettre de créer des fenêtres pop-up, très utile pour les messages importants, les fenêtres d'aide ou les "About" du logiciel. Pour ce faire, il suffit juste de taper « popup = Toplevel(fen) ». Nul besoin non plus de positionner celui-ci grâce aux classe Grid(), Pack() ou Place(). Les paramètres possibles sont bg, use, screen, relief, visual, class, bd [...]. Il en existe d'autres bien évidement.

Bien, nous en avons terminé avec Tkinter, si vous maîtrisez tout cela, alors vous pourrez faire pas mal de belles et convenables interfaces graphique pour vos programmes. Passons à la suite à présent.

3.math : Les Fonctions

Ce module n'intègre aucune classe préprogrammé si mes souvenirs sont bon. Néanmoins, il possède un bon nombre de fonctions permettant de travailler avec l'algèbre. Voic un petit tableau des fonctions présentes dans "math".

Nom de la fonction	*Explication de la fonction*
fabs(x)	*Retourne la valeur absolue de x*
floor(x)	*Retourne la partie entière de x*
ceil(x)	*Retourne l'arrondie de x*
fmod(x, y)	*Retourne le reste de la division x/y*
frexp(x)	*Retourne la mantisse et l'exposant de x*
ldexp(x, i)	*Retourne x*(2**i)*
modf(x)	*Retourne la partie flotante et la partie entière de x*
exp(x)	*Retourne l'exposant de x*
log(x, y)	*Retourne le logarithme de x en fonction de la base y*
log10(x)	*Retourne le logarithme en base 10 de x*

Nom de la fonction	*Explication de la fonction*
pow(x, y)	***Retourne le résultat de x exposant y***
sqrt(x)	***Retourne la racine carré de x***
cos(x)	***Retourne le cosinus de x***
sin(x)	***Retourne le sinus de x***
tan(x)	***Retourne la tangeante de x***
hypot(x, y)	***Retourne le résultat de sqrt(x*x + y*y)***
degrees(x)	***Converti un angle radians en degrés***
radians(x)	***Converti un angle degrés en radians***
cosh(x)	***Retourne le cosinus hyperbolique de x***
sinh(x)	***Retourne le sinus hyperbolique de x***
tanh(x)	***Retourne la tangeante hyperbolique de x***
pi	***Variable PI contenant 3.141592654***
e	***Variable contenant le logarithme népériens 2.718289***

Voilà pour les fonctions mathématique de la librairie math.

4.socket : Les Fonctions

Identique à la la librairie math, "socket" n'intègre pas de classe à ma connaissance, mais en revanche, elle intègre un bon nombre de fonction et de méthode.

Nom de la fonction	*Explication de la fonction*
has_ipv6	***Permet de savoir si le système intègre IPv6***
getfqdn([name])	***Retourne le nom d'hôte local à partir de name***
gethostbyname(hostname)	***Retourne l'IP à partir d'un nom de machine fourni en paramètre***
gethostbyname_ex(hostname)	***Retourne l'IP, l'hostname et l'aliaslist à partir d'un nom de machine donné en paramètre***
gethostname()	***Retourne le nom de la machine local***
gethostbyaddr(ip_adress)	***Retourne l'IP et le nom et la liste d'alias de la machine noté en paramètre***
getservbyname(service)	***Retourne le port du service cité en paramètre***
getservbyport(port)	***Retourne le nom du service à partir du port donné en paramètre***
socket([family[,type[,proto]]])	***Permet de créer des sockets***
inet_aton(ip_adress)	***Retourne la valeur hexadécimale de ip_adress***
inet_ntoa(packed_ip)	***Retourne l'IP à partir de la valeur héxadécimale donné comme paramètre à packed_ip***
getdefaulttimeout()	***Retourne la valeur du timeout par défaut***

Nom de la fonction	*Explication de la fonction*
setdefaulttimeout(timeout)	*Permet de définir la valeur du timeout par défaut*

5.socket : Les Méthodes

Voici à présent les méthodes permettant d'utiliser les sockets.

Nom de la fonction	*Explication de la fonction*
accetp()	*Permet d'accepter les connexions extérieur*
bind(address)	*Met le socket en écoute de connexion de la part de address*
close()	*Permet de fermer une connexion active*
connect(address)	*Permet d'établir des demandes de connexion*
getpeername()	*Retourne l'adresse surlequel le socket est connecté*
fileno()	*Retourne le fichier descripteur du socket*
getsockname()	*Retourne la propre adresse du socket*
listen(backlog)	*Permet d'écouter des connexions établie préalablement avec le socket*
recv(buffer)	*Permet de recevoir des informations textuel de la part de la machine connecté au socket*
send(string)	*Permet d'envoyé du texte à la machine connecté au socket*
sendto(string, address)	*Permet d'envoyé du texte à une adresse choisie*
shutdown(how)	*Permet de couper l'envoie ou la réception de données, ou les deux à la fois*

Etablir une connexion

Je ne vais pas m'étendre en parole inutile, j'ai dit en introduction que je ne souhaitais pas tourner autour du pot. Pour cette partie, je vais vous fournir un script d'exemple fait maison, cela sera plus simple.

```
import socket #On importe le module afin de posséder les fonctions nécéssaire

HOST = '127.0.0.1' #On déclare une variable contenant l'IP de la machine distante

PORT = 555 #On déclare une variable contenant le numéro du port de connection

My_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #On déclare notre socket, qui est de type SOCK_STREAM

try: #On essaye...

    print ''Connection en cour....'' #On affiche 'Connection en cour...'

    My_Socket.connect((HOST, PORT)) #On demande une connection sur l'hôte et le port contenue dans nos variable, et en se servant de notre socket

    print ''Connection établie...'' #Si la connection est obtenue, alors on affiche 'Connection établie...'
```

```
except: #Dans tout les autres cas (refus de connection, interruption par l'utilisateur...etc...)
    print ''Connection impossible !!'' #On affiche 'Connection impossible'
```

Vous avez donc pu voir par vous-même la façon d'établir des connexions, cela se passera toujours de la même manière, avec plus ou moins de chosex en plus en fonction de vos besoins réel. Le type de socket que j'ai choisie pour ce script est le plus simple, je n'aborderais pas les socket de type SOCK_DGRAM ou encore SOCK_SEQPACKET, qui sont bien plus compliqué à mettre en oeuvre. Pour ceux qui ne saurait pas encore, ces sockets servent à faire des transactions au travers d'Internet, alors que SOCK_STREAM reste en local.

Ecouter une connexion

J'en vois déjà qui sont impatient de savoir faire afin de créer leur propre sniffer ou programme de chat perso, mais j'ai légèrement bridé la cadence avec le type de socket, donc à par renifler votre propre réseau local, vous ne pourrez rien faire de plus. SOCK_STREAM est vraiment un cas d'école, et si j'utilise ce socket c'est pour la bonne raison que ce cour est là pour vous apprendre un langage, et non à le maîtriser de A à Z. Il vous faudra sûrement un peu de temp avant d'arriver à bien manipuler Python. Ci-dessous, encore un script d'exemple pour faire office de cour (Prenez cela comme une sorte de TP), celui-ci est une fonction agissant en tant que serveur de chat, je l'avais programmé un jour pour passer le temp (j'ai aussi créé le client qui va avec), mais je pense qu'il fera l'affaire pour ce petit exemple.

```
def _connect(): #On définit une nouvelle fonction
    IP = raw_input("Veuillez saisir l'adresse IP du systeme distant : ")#On demande une IP à l'utilisateur
    SERVER_NAME = raw_input("Veuillez donner un nom a votre server :") #On demande à l'utilisateur le nom qu'il veut donner à son serveur
    PSEUDO = raw_input("Votre pseudo :") #On lui demande son pseudo
    PORT = 30 #On définit le port de communication
    My_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #On déclare notre socket
    try: #On essaye...
        My_Socket.bind((IP, PORT)) #On met le socket en écoute de l'IP sur le port 30
        print "" #On saute une ligne
        print "Serveur en attente de connection de la part de %s" % IP #On inscrit la phrase ci-dessus en indiquant l'IP que le serveur attend
        print "" #On saute une ligne
    except: #Si une erreur se produit...
        print "Impossible d'etablir une liaison avec l'hote sur le port choisie !!"
        raw_input() #On fait une pause tant que l'utilisateur n'appuie pas sur [ENTER]
        start() #Une fonction de mon script que j'avais écrite, c'est juste l'interface que j'appele pour revenir au menu principal
```

```
    while 1: #Boucle 1
        My_Socket.listen(2) #On écoute tout ce qui se passe sur le port une fois la connection avec le client établie, c'est cette commande qui est la plus importante pour cet exemple...
        connection, hote = My_Socket.accept() #On déclare 2 variables qui accepterons les connections
        print "[-Connection etablie avec %s sur le port %s-]" % (hote[0], hote[1]) #La phrase disant que le serveur et le client sont branché
        connection.send("[-Bienvenue sur %s-]" % SERVER_NAME) #Le serveur envoie une message de bienvenue au client distant
        chatClient = connection.recv(5120) #On est près à recevoir les string du client, nous avons réservé 5Mo d'espace pour ces données
        try: #On essaye...
            print "Client: ", chatClient #On inscrit les données du client
            if chatClient.upper() == "DISCONNECT" or "": #On déclare une commande pour le client, si le serveur reçoit le mot 'DISCONNECT' ou un vide, alors la connection s'estompe
                connection.send("[-SERVER OFFLINE-]") #Donc le serveur envoie 'SERVER OFFLINE' pour dire au client que tout s'est bien dérouler et que la connection est à présent interrompue
                break #Enfin...maintennant elle l'est réellement
            chatServeur = raw_input("%s:"%PSEUDO) #On laisse l'utilisateur du serveur de chat taper son message
            connection.send(chatServeur) #Dès qu'il tape sur [ENTER], alors en envoie son message au client
            chatClient = connection.recv(5120) #Ensuite on déclare un buffer de 5Mo pour recevoir le message du client, puis nous recevons son message bien entendu
        except socket.error: #Si le socket à un problème alors...
            print "Le Client est hors ligne !!" #On indique à l'utilisateur du serveur que le client est OFF-LINE
            raw_input() #On fait une pause...
            start() #Puis ma fonction qui n'est pas écrit dans cet exemple mais qui me sert en réalité à retourner au menu
        connection.send("[-Good Bye-]") #On envoie 'GOOD BYE' au client
        print "Connection fermer par l'hote distant..." #On dit ça à l'utilisateur du serveur...
        connection.close() #On ferme la connection...attention, regardez plus haut, vous verrez que j'ai dabord déclaré cette variable comme étant un My_Socket.accept(), donc vous il ne faudra pas marquer ça à l'aveugle...
        raw_input() #On fait une pause...
        start() #Puis on retourne au menu, je répète, cette fonction est dans mon 'vrai' script, donc sa n'aura aucun effet chez vous...mis à part que vous aurez une erreur si vous ré-écrivez cette ligne, je vous conseil plutôt de taper 'exit' afin de quitter le programme...
```

Attendre une connexion

Ce genre de pratique sert, entre autre, à créer des serveurs. Vive les Troyens et les systèmes de Chat invisible. Voici ci-dessous le script d'exemple. Je l'ai créé à la manière d'un serveur qui attendrait des connexions.

```
import socket #On importe le module socket

HOST = '127.0.0.1' #On déclare une variable contenant l'adresse de la machine distante

PORT = 555 #On déclare une seconde variable qui contient le port de connection


My_Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #On créé notre socket

try: #On essaye...

    print ''Démarrage du serveur...'' #On affiche 'Démarrage du serveur...'

    My_Socket.bind((HOST, PORT)) #On lance l'écoute de connection sur un port donné, c'est cette ligne qui permet d'attendre une connection de la part de '127.0.0.1' sur le port 555 (faite le test vous-même avec telnet, prenez soin de lancer ce script auparavant, c'est préférable)

    print ''Serveur lancé, en attente de connection...'' #On affiche 'Serveur lancé, en attente de connection...'

except: #Si une erreur surgi, alors...

   print ''Impossible de lancer le serveur...'' #On affiche le message d'erreur signalant l'impossibilité de lancer l'écoute de connection
```

J'espère que vous avez compris le fonctionnement de ce module, car si vous souhaitez établir des scripts devant êtres capable de dialoguer sur un réseau, il serait préférable pour vous que vous la connaissiez et la maîtrisiez convenablement.

6.os : Les Fonctions

Un autre module intéréssant. Il permet de dialoguer avec l'OS (système d'exploitation). Cette librairie ne contient aucune classe, mais possède tout de même un bon nombre de fonctions.

Nom de la fonction	*Explication de la fonction*
name	*Retourne le type d'OS utilisé*
path	*Retourne le répertoire ou Python est en cour d'utilisation*
environ	*Retourne un mapping de l'OS*
getcwd()	*Retourne le repertoire de Python*
close(fd)	*Permet de fermer un fichier grâce à son indice*
open(file, flag[, mode])	*Permet d'ouvrir des fichiers en lecture ou en écriture*
read(fd, n)	*Permet de lire n byte d'un fichier grâce à son descripteur*
abort()	*Envoie un signal SIGABRT au processus courant*

Nom de la fonction	*Explication de la fonction*
system(command)	***Execute une commande système***
urandom(n)	***Retourne une chaine aléatoire de n byte***

Il en existe d'autres, mais je ne les ai pas cité ici. Le mieux est encore que vous regardiez par vous-même, auquel cas je vais y passer plusieurs pages. J'ai déjà fait un énorme tri afin de créer ce tableau, et pour cette raison, je vous laisse chercher vous même.

7.winsound : Les Fonctions

Cette librairie ne fonctionne que sur Windows, et ne possède aucune classe. Néanmoins, voici un petit tableau récapitulant son potentiel.

Nom de la fonction	*Explication de la fonction*
Beep(frequency, duration)	***Permet d'émettre une fréquence avec l'UC***
PlaySound(sound, flag)	***Permet d'émettre un son système***
MessageBeep([type = MB_OK])	***Permet d'émettre un son de MessageBox***

Beep()

Il suffit de taper par exemple : « Beep(500, 1000) », et l'on obtient une fréquence de son aigu pendant une seconde (1000 millisecondes).

PlaySound()

Il existe plusieurs paramètres pour cette fonction, la liste des sons possibles est "SystemExit", "SystemHand", "SystemAsterisk", "SystemExclamation" et "SystemQuestion". Pour ce qui est des flags, il existe SND_NOWAIT, SND_LOOP, SND_ALIAS, SND_NOSTOP, SND_MEMORY, SND_PURGE, SND_NODEFAULT, SND_ASYNC et SND_FILENAME. Pour le fonctionnement, il suffit de taper « winsound.PlaySound("SystemExit", winsound.SND_ALIAS) », et vous entendrez le son du système lorsqu'il s'éteind, essayez-en plusieurs, vous verrez par vous-même.

MessageBeep()

Rien de très compliqué ici non plus, il suffit juste d'écrire « MessageBeep (MB_OK) ». La liste des paramètres existant est MB_OK, MB_ICONQUESTION, MB_ICONHAND, MB_ICONASTERISK, MB_ICONEXCLAMATION.

8.platform: Les Fonctions

Cette librairie est tout de même intéréssante, surtout lorsque l'on souhaite connaître des choses technique vis à vis de son ordinateur (le nom de son processeur par exemple), je vous ais établie un petit récapitulatif.

Nom de la fonction	*Explication de la fonction*
machine()	***Retourne le type de machine***
node()	***Retourne le nom du système sur le réseau***
platform()	***Retourne le nom type de platform***
processor()	***Retourne le vrai nom du processeur***

Nom de la fonction	*Explication de la fonction*
python_version()	***Retourne la version de Python***
release()	***Retourne la date de release du système***
system()	***Retourne le nom de l'OS***
version()	***Retourne le bitrate du système***
uname()	***Retourne le nom de l'Os, le nom du système et son bitrate***

9.sys: Les Fonctions

Cette librairie offre la possibilité de dialoguer avec le système, et d'obtenir divers renseignements (version par exemple). Voici un tableau récapitulatif de ce module.

Nom de la fonction	*Explication de la fonction*
copyright	***Retourne le copyright de Python***
exit	***Permet de quitter Python***
getwindowsversion()	***Retourne la version de Windows***
modules	***Retourne la liste des modules présent dans Python***
platform	***Retourne le nom de la plateforme***
version	***Retourne la version de Python***
version_info	***Retourne la version de Python***
winver	***Retourne la version de Windows***

2)Les Exercices

-1.1 : Ecrivez une fenêtre possédant un bouton dont l'unique but est de détruire cette fenêtre.

-1.2 : Ecrivez un petit programme de chat local, vous devez créer le serveur ainsi que le client qui va avec. Vous devez laissé l'utilisateur choisir l'IP, en revanche vous devez paramétrer un port par défaut pour le serveur et le client vous-même.

-1.3 : Ecrivez un petit script d'OSfingerprinting (détection de système d'exploitation) local.

Faites ces petits exercices afin de vous entraînez, bien sûr, je ne vous force à rien, c'est vous qui voyez. Les corrigés sont disponibles à la fin de ce document.

XIV)Les Corrigés

Ici, vous avez les corrigés des exercices des différentes leçons de ce cour, le chapitre 5 est le chapitre sur les variables, donc il n'y a rien à craindre, je ne me suis pas trompé rassurez-vous.

Chapitre 5

Exercices 1.1 :

```
>>> Hello = 'Bonjour le monde'
>>> print Hello
```

Ou encore comme ci-dessous.

```
>>> print 'Bonjour le monde'
```

Exercices 1.2 :

```
>>> PI = 3.141592654
>>> Somme = PI + 10
>>> print Somme
```

Ou encore comme ci-dessous.

```
>>> print 3.141592654 + 10
```

Exercice 1.3 :

```
>>> A = 20
>>> print 'Dix et dix = %s' %A
```

Chapitre 6

Exercices 1.1 :

```
>>> Dico = {'bananes' : 20, 'cerises' : 40, 'pommes' : 32}
```

Exercices 1.2 :

```
>>> Dico = {'bouteilles' : 4, 'bols' : 3, 'assiettes' : 2}
>>> Two = Dico.copy()
```

Exercices 1.2 :

```
>>> Premier = {'ballons' : 10, 'CD' : 50}
>>> Dico = Premier.copy()
>>> del Premier
```

Chapitre 7

Exercices 1.1 :

```
>>> Reste = 3.141592654%3
```

Exercices 1.2:

```
>>> Resultat = 2*2-6/2+1+10%2
>>> print Resultat
```

Chapitre 8

Exercices 1.1 :

```
>>> Chaine = 'Bonjour La Vie !!'
>>> print Chaine.upper()
```

Exercices 1.2 :

```
>>> Name = 'Sophocle'
>>> Long = len(Name)
>>> print Long
```

Chapitre 9

Exercices 1.1 :

```
>>> i = 0
>>> while i < 170:
      i = i + 17
      print i
```

Exercices 1.2 :

```
>>> PI = 3.141592654
>>> TEST = PI
>>> if PI == TEST:
      print "PI est égal à TEST"
else:
      print "PI n'est pas égal à TEST"
```

Chapitre 10

Exercices 1.1 :

```
>>> try:
        Tablo = ['Voitures : 50\n', 'Tank : 30\n', 'Hommes : 1000\n']
        f = open('C:/Tablo.txt', 'w')
```

```
        f.writelines(Tablo)
        f.close()
    except:
        print 'Impossible de copier le tableau dans le fichier !!'
```

Exercices 1.2 :

```
>>> User = raw_input('Veuillez saisir le nom du fichier à copier : ')
>>> try:
      f = open('C:/Copie_Utilisateur.txt', 'w')
      h = open(User, 'r')
      f.writelines(h)
      f.close()
      h.close()
    except:
      print 'Impossible de créer une copie de votre fichier !!'
```

Chapitre 11

Exercices 1.1 :

```
>>> def carre(x):
      print x*x
```

Exercices 1.2 :

```
>>> def table(x):
      z = x
      try:
            while x == x:
                  print x
                     x = x + z
      except KeyboardInterrupt:
            print "Affichage interrompue par l'utilisateur..."
```

Exercices 1.3 :

```
>>>>> def calc(x, y, z):
      "Exemple : calc(5, '+', 5)"
      if y == '*':
            print x*z
      elif y == '-':
            print x-z
      elif y == '+':
```

```
            print x+z
      elif y == '/':
            print x/z
      elif y == '%':
            print x%z
      elif y == '<':
            print x<z
      elif y == '>':
            print x>z
      else:
            print "Valeur des paramètre incorrect !!"
            print "x et z doivent être des chiffres, y doit être un signe
mis entre guillemet"
```

Chapitre 12

Exercices 1.1 :

```
  class Mere:
      def __init__(self, age = 15, nom = 'Sophocle', lieu = 'Secret...',
diplome = 'Secret...'):
            self._age = age
            self._nom = nom
            self._lieu = lieu
            self._diplome = diplome


  class Presentation(Mere):
      def __init__(self):
            Mere.__init__(self)
      def Start(self):
            print "Nom : %s" %self._nom
            print "Age : %s" %self._age
            print "Lieu : %s" %self._lieu
            print "Diplôme : %s" %self._diplome
```

Exercices 1.2 :

```
  class Compteur:
      def Table(self):
            i = 158
            s = i*10
            while i <= s:
                  print i
                  i = i + 158
```

Exercices 1.2 :

```
>>> class File:
        def Create(self):
           Name_File = raw_input('Veuillez saisir un nom pour le fichier à
créer : ')
            Extension = raw_input('Choisissez une extension pour ce fichier
(Exemple : .py): ')
            Place = 'C:/'
            try:
                  f = open(Place+Name_File+Extension, 'w')
                    f.close()
            except:
                  print "Impossible de créer le fichier..."
```

Chapitre 13

Exercices 1.1 :

```
>>> from Tkinter import *
>>> fen = Tk()
>>> fen.title('Kill-me')
>>> bou = Button(fen, text = 'QUITTER', command = fen.destroy)
>>> bou.pack()
>>> fen.mainloop()
```

Exercices 1.2 -> Le serveur:

```
>>> IP = raw_input("Veuillez saisir l'adresse IP du systeme distant : ")
    SERVER_NAME = raw_input("Veuillez donner un nom a votre server :")
    PSEUDO = raw_input("Votre pseudo :")
    PORT = 30
    FirstSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        FirstSocket.bind((IP, PORT))
        print ""
        print "Serveur en attente de connexion de la part de %s" % IP
        print ""
    except:
        print "Impossible d'etablir une liaison avec l'hote sur le port
choisie !!"
        raw_input("")
        exit
    while 1:
        FirstSocket.listen(2)
```

```
        connection, hote = FirstSocket.accept()
        print "[-Connexion etablie avec %s sur le port %s-]" % (hote[0],
hote[1])
        connection.send("[-Bienvenue sur %s-]" % SERVER_NAME)
        chatClient = connection.recv(5120)
        try:
            print "Client: ", chatClient
            if chatClient.upper() == "DISCONNECT" or "":
                connection.send("[-SERVER OFFLINE-]")
                break
            chatServeur = raw_input("%s:"%PSEUDO)
            connection.send(chatServeur)
            chatClient = connection.recv(5120)
        except socket.error:
            print "Le Client est hors ligne !!"
            raw_input("")
            exit
        connection.send("[-Good Bye-]")
        print "Connexion fermer par l'hote distant..."
        connection.close()
        raw_input("")
        exit
```

Exercices 1.2 -> Le client:

```
>>> IP = raw_input("Veuillez saisir l'adresse IP du systeme distant : ")
    PSEUDO = raw_input("Veuillez entrer votre pseudonyme :")
    PORT = 30
    FirstSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        FirstSocket.connect((IP, PORT))
        print "Connection a %s en cour..." %IP
    except:
        print "Impossible d'etablir une connection avec le serveur
distant !!"
        raw_input("")
    print "[- Connexion etablie avec",IP,"sur le port",PORT,"-]"
    print ""
    chatServeur = FirstSocket.recv(5120)
    while 1:
        if chatServeur.upper() == 'DISCONNECT' or "":
            connection.send("[-CLIENT OFFLINE-]")
```

```
            break
        try:
            print "Serveur: ", chatServeur
            chatClient = raw_input("%s:" % PSEUDO)
            FirstSocket.send(chatClient)
            chatServeur = FirstSocket.recv(5120)
        except:
            print "Le Serveur est hors ligne !!"
            raw_input("")
            exit
    print "Connexion fermer par l'hote distant..."
    raw_input("")
    exit
```

Exercices 1.3:

```
from os import name
from sys import getwindowsversion, platform, version_info
from platform import machine, node, processor, python_build, system,
platform

MACHINE = machine()
NETWORK_NAME = node()
PLATFORM = platform()
PROCESSOR = processor()
PYTHON_DATE = python_build()
PYTHON_RELEASE = version_info
OS_NAME = system()
OS_TYPE = name
WINDOWS_VERSION = getwindowsversion()
if MACHINE == '':
    MACHINE = 'Indéfini'
else:
    MACHINE = MACHINE
print "Type de processeur : %s" %MACHINE
print "Nom de votre système : %s" %NETWORK_NAME
if PROCESSOR == '':
    PROCESSOR = 'Indéfini'
else:
    PROCESSOR = PROCESSOR
print "Nom du processeur : %s" %PROCESSOR
```

```
if PLATFORM == '':
    PLATFORM = 'Indéfini'
else:
    PLATFORM = PLATFORM
print "Plateforme : %s" %PLATFORM
if OS_NAME == '':
    OS_NAME = 'Indéfini'
else:
    OS_NAME = OS_NAME
print "Nom de l'OS : %s" %OS_NAME
if  OS_NAME == 'Windows':
    if WINDOWS_VERSION == '':
        WINDOWS_VERSION = 'Indéfini'
    else:
        print "Version de Windows : ", WINDOWS_VERSION
else:
     OS_NAME = OS_NAME
if OS_TYPE == '':
    OS_TYPE = 'Indéfini'
else:
    OS_TYPE = OS_TYPE
print "Type de système : %s" %OS_TYPE
print "Date de release de Python :",PYTHON_DATE
print "Version de Python :", PYTHON_RELEASE
raw_input()
exit
```

XV)CoNcLuSiOn

Voilà, ce cour de programmation Python est finalement terminé. Je vous laisse donc à vos interpréteurs pour apprendre ce que je ne vous ai pas encore enseigné via ce document. Rassurez-vous, Python est un des langages les plus simple et les plus évolué du monde, par conséquent la maitrise parfaite de ce langage n'est qu'une simple question de temps.

Si vous avez des questions ou des problèmes en matières de programmation Python, vous pouvez essayer de trouver une réponse sur le site des développeurs Francophone (plusieurs fois cité en début de cour), mais si vous ne trouvez toujours pas de solution à votre problème, vous pouvez tenter de me le poser par mail, et j'essairais d'y répondre le plus vite possible. Vous trouverez mon adresse sur le site de [-Digital Escape-], ou encore sur la Board de l'Hackademy School.

Sophocle