python

```
print("Hello, world!")
```

FORMATION PYTHON : Notions de base

http://cahier-de-prepa.fr/info-ipein

said_anis@hotmail.com

Plan de formation

1. Python ? C'est quoi ?
2. Syntaxe python
3. Modules
4. Opérations d'entrée sortie
5. Structures conditionnelles
6. Structures itératives

said_anis@hotmail.com

Python ? C'est quoi ?

said_anis@hotmail.com

Plan

1. Python ? C'est quoi ?
2. Pourquoi Python ?
3. Domaines d'application
4. Utilisation de python
5. IDEs

said_anis@hotmail.com

Python ? C'est quoi ?

- Python a été créé par **Guido van Rossum** dans la fin des années 1980 à l'Institut national de recherche pour les mathématiques et d'informatique aux Pays-Bas.
- Le code source de Python est disponible sous la licence GNU [General Public License (GPL)], Python est **un logiciel libre « free » :** utilisation sans restriction dans les projets commerciaux

said_anis@hotmail.com

Python ? C'est quoi ?

- Python est dérivée de nombreuses autres langages, y compris ABC, Modula-3, C, C ++, Algol-68, Smalltalk, et shell Unix et d'autres langages de script.
- Python est un langage de programmation :
  - De haut niveau comme (java , perl , Ruby etc. ..) basé sur le langage C
  - Orienté Objet
  - Interactif
  - Interprété.

Python ? C'est quoi ?

- **Python est interprété:** Cela signifie qu'il est traité à l'exécution par l'interprète et vous ne devez pas compiler votre programme avant de l'exécuter. Ceci est similaire à PERL et PHP.

- **Python est interactif:** Cela signifie que vous pouvez réellement utiliser une invite de commande Python et d'interagir directement avec elle pour écrire vos programmes.

Python ? C'est quoi ?

- **Python est orientée objet:** Cela signifie que Python supporte le style orienté objet qui est une technique de programmation qui encapsule le code dans des objets.
  - "In Python, Everything is an object"

- **Python est le langage du débutant:** Python est un langage pour les programmeurs débutants, il soutient le développement d'un large éventail d'applications.

Caractéristiques de Python ?

- **Facile à apprendre:**
  - Python a relativement peu de mots-clés,
  - la structure est simple et une syntaxe clairement défini.
  - Cela permet à l'étudiant d'assimiler le langage dans une période de temps relativement courte.
- **Facile à lire:** Le code Python est clairement définie et si bien écrit. Il est simple à lire et à comprendre.
- **Facile à entretenir**: L'un des avantages de Python est que son code source est assez facile à entretenir.

said_anis@hotmail.com

Caractéristiques de Python ?

- **Portable:** Python peut fonctionner sur une grande variété de plates-formes logicielles (OS) et matérielles (PC, MAC, Tablettes, …) et possède la même interface sur toutes les plateformes.
- **Extensible:** Vous pouvez ajouter des modules de bas niveau de l'interpréteur Python. Ces modules permettent aux programmeurs d'ajouter ou de personnaliser leurs outils pour être plus efficace.
- **Interaction avec les bases de données :** Python fournit des interfaces de toutes les principales bases de données commerciales.
- **Programmer des interfaces graphiques**

Caractéristiques de Python ?

- Couramment utilisé pour la production de contenu HTML sur les sites Web.
- Idéal pour les fichiers texte.
- Types intégrés utiles (listes, dictionnaires).
- Multi-usages (Web, GUI, Scripting, etc.)
- Fortement typé et typé dynamiquement
- Axé sur la lisibilité et la productivité

Caractéristiques de Python ?

- Pas de délimiteurs de blocs de code d'instructions
  - "Life's better without braces“ (Bruce Eckel)
- Gestion automatique de la mémoire : Utilise le ramasse miette (Garbage-collection) pour la gestion des objets en mémoire
- Python est un langage inter-opérable (avec C Cython, Java Jython, C++, Fortran F2Py…)
- Possède un système d'exceptions

Les domaines d'application de python

- L'apprentissage de la programmation objet
- L'accès aux bases de données (relationnelles).
- La réalisation d'interfaces graphiques utilisateurs (GUI).
- Le calcul scientifique et l'imagerie.
- du traitement du son, de la synthèse vocale
- des jeux vidéo en 2D
- des applications multitouches (pour tablette et Smartphone à écran tactile)
- des applications Web
- …

Python : Versions

- Python 1.0 publiée en 1994
- Python 2.0 publiée en 2000
- Python 2.7 publiée en 2008
- Python 3.0 publiée en 2008
- La Version actuelle est la version 3.? disponible pour UNIX, PC et Mac.
- Python toujours en développement grâce à une large communauté très active

said_anis@hotmail.com

Python : Documentation

Pour plus d'informations

- The Official Python Tutorial : https://docs.python.org/3.4/tutorial/
- *Learning Python* by Mark Lutz
- *Python Essential Reference* by David Beazley
- *Python Cookbook*, ed. by Martelli, Ravenscroft and Ascher
- http://code.activestate.com/recipes/langs/python
- http://wiki.python.org/moin/PythonBooks
- The Python Quick Reference http://rgruet.free.fr/PQR2.3.html

Python : Installation

- Python pour Win / Mac / Unix / Linux est disponible sur www.python.org
- En général, une installation facile.
  - Simple exécutable à installer sous Windows.
  - Sur Mac, déjà partie de Mac OS X.
  - Sous Linux, généralement préinstallé.

Python : Installation

- Lors de l'installation vous pouvez sélectionner le répertoire où sera installé Python
- La documentation (sous Windows) se trouve dans le répertoire /Doc de l'installation, on trouve un fichier avec l'extension **.chm**
- Vous prouvez trouver cette documentation sous www.python.org/doc

Python : Installation autres paquetages

- epydoc : permet l'extraction de la documentation des sources
- py2exe : permet de générer une application sous la forme d'un exécutable
- …

said_anis@hotmail.com

Python : Utilisation

- Une fois installé, Python peut être utilisé en deux modes :
- **Le mode Interactif :** Les instructions tapées dans l'interpréteur de la ligne de commande python (Python Shell) sont exécutées directement.
  - Vous pouvez le faire depuis Unix, DOS, ou tout autre système qui vous fournit une fenêtre de l'interpréteur de ligne de commande ou un shell.

    >>>

  - Le codage avec le shell est parfait pour l'apprentissage du langage et pour tester les modules.

Python : Utilisation

- **Le mode script** : Un script Python peut être saisie dans un fichier d'extension .py et exécutée avec l'interpréteur python (Python Shell)
  1. Editer votre script avec n'importe quel éditeur de texte
  2. Ouvrez le python shell
  3. Ouvrer votre script
  4. Exécuter le script

Python : EDI

- **l'environnement de développement intégré (IDE):**
  Vous pouvez exécuter Python à partir d'une interface utilisateur graphique (GUI). Tout ce que vous avez besoin est une application GUI sur votre système qui prend en charge Python.
  - Liste des EDIs disponibles :

http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Python : Editeurs et EDIs

- Notepad ++.
- Sublime Text.
- Gedit (Gnome).
- Kate/Kwrite (KDE).
- Geany
- Eclipce PyDev / Aptana Pydev
- Pycharm
- IDLE

said_anis@hotmail.com

Python : IDLE

- IDLE :
  - est un environnement de développement intégré pour le langage Python.
  - Intégralement écrit avec Python et la bibliothèque graphique Tkinter (Tool kit interface).
- Les principales fonctionnalités de **IDLE** sont :
  - L'éditeur de texte avec coloration syntaxique, L'autocomplétion, et l'indentation (pour la création de fichiers python : scripts);
  - Un interpréteur (pour exécuter les scripts)
  - le débogueur intégré avec avancement par étape, point d'arrêts persistants et pile d'appels (pour tester les scripts).

said_anis@hotmail.com

Python : Classement

Classé:

- 5ème par ULTIME
- 4ème par IEEE
- 8ème par TIOBE software company

Ultime	IEEE	Tiobe
1. C	1. Java	1. C
2. Java	2. C	2. Java
3. C++	3. C++	3. Objective-C
4. C#	4. Python	4. C++
5. Python	5. C#	5. Visual Basic
6. PHP	6. PHP	6. C#
7. Objective-C	7. Javascript	7. PHP
8. Visual Basic	8. Ruby	8. Python
9. Ruby	9. R	9. JavaScript
10. Perl	10. MATLAB	10. Transact-SQL
11. MATLAB	11. SQL	11. Perl
12. Assembly	12. Perl	12. ASP.NET
13. R	13. Assembly	13. F#
14. ASP.NET	14. HTML	14. Ruby
15. Lisp	15. Visual Basic	15. ActionScript
16. Delphi	16. Objective-C	16. Swift
17. Go	17. Scala	17. Delphi
18. SAS	18. Arduino	18. Lisp

Python : Travail à faire

- Il vous demande d'installer Python sur vos ordinateurs personnels ou soyez à l'aise de l'utiliser ici dans le laboratoire.
- Il vous demande également de lire quelques tutoriels de Python en ligne.
- Utiliser l'aide en ligne (help) pour savoir plus à propos du fonctionnement du python
  - Exemple : help (int)

Variables et opérateurs

said_anis@hotmail.com

Plan

1. Variables
2. Types
3. Opérateurs
4. Fonctions mathématiques

said_anis@hotmail.com

Variables : Types sous python

Les types sous python :

1. Les types élémentaires (entier, réel, …)
2. Les conteneurs
    1. Non Mutables (modification non autorisée)
        1. Ordonnés
            1. Tuples (tableaux non modifiables)
            2. Chaines de caractères
    2. Mutables (modification autorisée)
        1. Ordonnés
            1. Les listes (tableaux modifiables)
        2. Non Ordonnés
            1. Dictionnaires
            2. Ensemble

said_anis@hotmail.com

Variables : Identité des variables

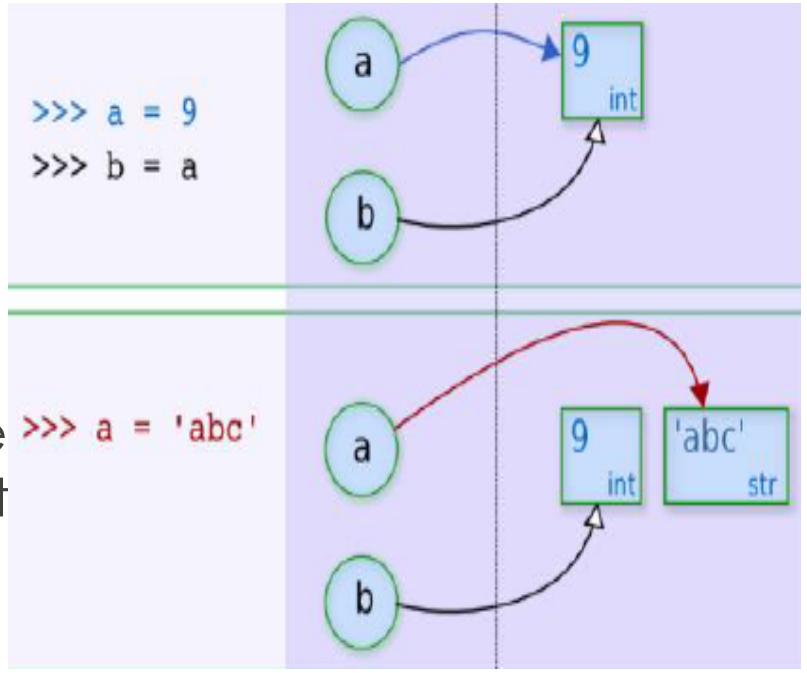
- Python est un **Langage Orienté Objet**
- Tout est objet : données, fonctions, modules...
- Un objet **B** possède:
  - identité **id** (adresse mémoire), **id(B)**
  - Un type **type(B)**: (intrinsèque : int, float, str, bool ... ou définit par l'utilisateur à l'aide d'une classe)
  - contient des données (valeur).
- Un objet ne peut changé ni d'identité ni de type !
- L'opérateur **is** compare l'identité de 2 objets (adresses)
- L'opérateur **==** compare le contenu de deux objets

Variables : Identité des variables

- Le nom de variable est une référence vers l'objet (valeur)
- Quand un objet n'a plus de nom (nombre de références nul), il est détruit automatiquement (mécanisme automatique de "ramasse miettes", ou « garbage collector »).

Exemple :

- **b = a** ne fait pas une copie de **a** mais on a affecté à la variable **b** l'adresse (référence)de l'objet **9**
- Donc ,a et b référencent le même objet
- a= 'abc' , on a affecté à la variable a la référence de l'objet contenant la valeur 'abc', alors que **b** garde sa propre référence vers **9**



Typage des variables

- Sous Python, il n'est pas nécessaire d'écrire des lignes de programme spécifiques pour définir le type des variables avant de pouvoir les utiliser.
- Il vous suffit en effet d'assigner une valeur à un nom de variable pour que celle-ci soit automatiquement créée avec le type qui correspond au mieux à la valeur fournie

said_anis@hotmail.com

Variables : Types élémentaires

- Le type d'un objet détermine :
  - les valeurs : domaine de définition
  - les opérations
- Les types élémentaires intrinsèques :
  - Le NONETYPE
  - Le type BOOL
  - Les types numériques int, float, complex
- Le NoneType : La seule valeur possible **None**
  - C'est la valeur retournée par une fonction qui ne retourne rien
  - Peut être utiliser pour vérifier l'absence de valeur

said_anis@hotmail.com

Variables : Règles de nommage

- Les noms sont sensibles à la casse et ne peut pas commencer par un nombre. Ils peuvent contenir des lettres, des chiffres et caractères de soulignement.
- Il y a quelques mots réservés :

```
and, assert, break, class, continue, def,
del, elif, else, except, exec, finally, for,
from, global, if, import, in, is, lambda, not,
or, pass, print, raise, return, try, while
```

Variables : Affectation simple

- **Affectation Simple :**

```
>>> n = 7 # définir n et lui donner la valeur 7
>>> msg = "Quoi de neuf ?"
#   affecter la valeur "Quoi de neuf ?" à  msg
>>> pi = 3.14159 # assigner sa valeur à la variable pi
```

- **Affichage:**

```
>>> n
7
>>> msg
'Quoi de neuf ?'
>>> pi
3.14159
>>> print (msg)
Quoi de neuf ?
>>> print (n)
7
```

said_anis@hotmail.com

Variables : Affichage

- Il y a une différence dans les affichages obtenus avec chacune des deux méthodes :
  - La fonction **print()** n'affiche que la valeur de la variable
  - La méthode qui consiste à entrer seulement le nom de la variable, affiche aussi des apostrophes afin de vous rappeler que la variable traitée est du type chaîne
- **Exemple :** Permutation de deux variables :

```
>>> a=2
>>> b=5
>>> a,b=b,a
>>> print(a,b)
(5, 2)
```

Variables : Affectation multiple


```
# assigner une valeur à plusieurs variables
>>> x = y = 7
>>> x
7
>>> y
7
```

```
#  effectuer des affectations parallèles
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Variables : Visualiser le contenu

- visualiser le contenu des variables instruction par instruction par les site : http://www.pythontutor.com/visualize.html#mode=edit

```
>>> x = 2
>>> print (x)
>>> y = x
>>> print (y)
>>> z = x + 1
>>> print (z)
>>> x = 5
>>> print (x)
>>> print (y)
>>> print (z)
```

Variables : Types

- **type() :** retourne le type de la variable

```
>>> print(type(n))
<class 'int'>
```

said_anis@hotmail.com

Variables : Types [Entier]

- Le type entier (int) n'est limité en taille que par la mémoire de la machine.
- Les entiers longs ont une taille illimitée
  - Limité uniquement par la mémoire disponible
- ***Opérateurs arithmétiques***
- Sous Python, les règles de priorité sont les mêmes que celles enseignées au cours de mathématiques

said_anis@hotmail.com

Variables : Types [Entier]

Opération	Opérateur	Exemple

Addition	+	>>> Age = Age + 1
		# en plus court : Age += 1
		>>> print (Age)
		18

Soustraction	-	Age = Age - 3 # en plus court : Age -= 3 >>> print (Age) 15

Multiplication	*	Age = Age * 2
		# en plus court : Age *= 2
		>>> print (Age)
		30
		>>> a = 6*3-20
		>>> print (a)
		-2

Variables : Types [Entier]

Opération	Opérateur	Exemple

Puissance	**	>>> Mo = 2**20
		>>> print (Mo)
		1048576

division réelle	/	>>> c = 14/3
		>>> print (c)
		4.66666666667

quotient de la division entière	//	>>> d = 450//360
		>>> print (d)
		1

Variables : Types [Entier]

Opération	Opérateur	Exemple
division entière	divmod(x,y)	divmod(x,y)

reste de la division entière	%	>>> reste = 450 % 360
		>>> print (reste)
		90

Conversion entre les bases	bin(x) ;
	oct(x) ;
	hex(x)
		>>> bin(2013)
		'0b11111011101'
		>>> oct(2013)
		'0o3735'
		>>> hex(2013)
		'0x7dd'

said_anis@hotmail.com

Variables : Types [Entier]

Opérateurs relationnels

x== y *# x est égal à y*

x **!=** y *# x est différent de y*

x **>** y *# x est plus grand que y*

x **<** y *# x est plus petit que y*

x **>=** y *# x est plus grand que, ou égal à y*
x **<=** y *# x est plus petit que, ou égal à y*

Exemple

>>> a=3

said_anis@hotmail.com

Variables : Types [Réel]

- Le type réel (float) : nombres à virgule flottante
- le séparateur décimal est un point (et non une virgule)
- Exemple :

```
>>> b = 17.0
>>> print(b)
17.0
>>> print (type(b))
<class 'float'>


>>> c = 14.0/3.0
>>> print(c)
```

Variables : Types [Réel]

- **Notation scientifique :**

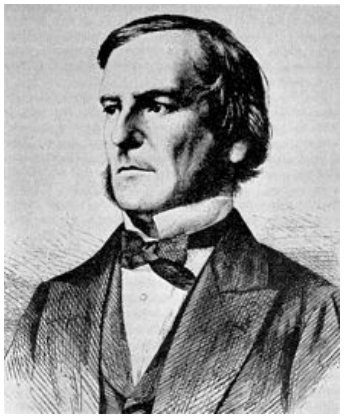
```
>>>> x = -11.7848992e4
# équivalente à  >>>> x = 11.7848992 * 10^4
>>>> print(x)
-117848.992
```

- int(x) convertit x à un entier
- float(x) convertit x à un nombre réel

Variables : Conversion de types

Fonction	Rôle	Exemple
int(x)	Convertir x à un entier	x = 1.2 int(x) x = Int("12")
float(x)	convertir x à un nombre réel	float("3.14")
str(15.5)	Convertir vers une chaine de caractères	str(15.5)

Variables : Types [Booléen]



George Boole

- Deux valeurs sont possibles : **True** et **False**

```
>>> a = True
>>> print (type(a))
<class 'bool'>
```

- 0 et None sont fausses, tout le reste est vrai
- Dans Python, tout peut être convertit à un booléen
  - a = bool ("un objet")
  - print (type(a))

Variables : Types [Booléen]

- ***Les opérateurs logiques : and, or, not***

```
>>> note=13.0
>>> mentionAB = note>=12.0 and note<14.0
>>> print(mentionAB)
True
>>> print (not mentionAB)
False
>>> print (note==20.0 or note==0.0)
False
```

- Lorsque **None** est renvoyé, l'interpréteur n'affiche rien

```
>>> None and 2
>>> None or 2
```


said_anis@hotmail.com

Variables : Types [Booléen]

- ***<u>Précédences des opérateurs sur les booléen</u>***
- L'opérateur **not** a une précédence sur **or** et **and**

```
>>> not True or True
True
>>> not(True or True)
False
```

- L'opérateur **and** a une précédence sur **or**

```
>>> True or False and False
True
>>> True or (False and False)
True
```

Variables : Types [Booléen]

- Toutes ces expressions sont équivalentes à False
  - Les nombres 0(int), 0L, 0.0(float) et 0j (complexe)
  - Toute structure de donnée vide,
  - La chaine de caractère vide " "

these_are_False = False **or** 0 **or** 0.0 **or** 0j **or** "" **or** {} **or** [] **or** None

- Tous le reste est équivalent à True
  - Les nombres non nuls, chaines de caractères non vides, toute structure de données non vide

these_are_true = True **and** 1 **and** "text" **and** {'a' : 'b'} **and** ['c' , 'd']

Variables : Types [Complexe]

- les complexes sont écrits en notation cartésienne formée de deux flottants.
- La partie imaginaire est suffixée par j.
- Mêmes opérations sont prises en charge comme le type entier et le type réel.

said_anis@hotmail.com

Variables : Types [Complexe]

opération	Exemple
complex1+ complex2	>>> (2+3j) + (4-7j) (6-4j)
complex.real	>>> (9+5j).real 9.0
complex.imag	>>> (9+5j).imag 5.0
abs(complex)	>>> abs(3+4j) 5.0
complex.conjugate()	>>> (3+4j).conjugate() (3-4j)
complex(re,im)	>>> complex(2,4) (2+4j)

Variables : Comparaison d'identités

- 1 is 1 == True #identité
- 1 is '1' == True #non identité
- bool(1) ==True
- bool(True) ==True
- 1 and True == True
- 1 is True == False

Opérations se comportent différemment sur les différents types


```
>>> 3.0 + 4.0

>>> 3 + 4

>>> 3 + 4.0

>>> "3" + "4"

>>> 3 + "4"      # Error



>>> 1 == True
```

said_anis@hotmail.com

Fonctions mathématiques

- Pour utiliser certaines de ces fonctions, vous devez ajouter à votre programme python le code suivant :

  >>> from math import * ou >>> from **c**math import *

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root
pow(x, y)	x to the power y

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

Les commentaires

- Le commentaire commence par#
  - Le reste de la ligne sera ignoré
- Un commentaire sur une seule ligne

```
# Ceci est un commentaire
```

- Un commentaire sur plusieurs lignes

```
""" ceci est un commentaire
sur deux lignes """
```

Les modules

said_anis@hotmail.com

Modules : Introduction

- Les **modules** sont des ensembles de **fonctions** appartenant au même domaine enregistrées dans un fichier dont le nom a la forme ***<nom du module>.py***
- Il n'est pas possible d'intégrer toutes les fonctions imaginables dans le corps standard de Python, car il en existe virtuellement une infinité, qui ne sont utilisés que par un petit nombre de programmes.

said_anis@hotmail.com

Modules : Introduction

- **Exemples de modules courants**:

math : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).

random : génération de nombres aléatoires.

time : permet d'accéder aux fonctions gérant le temps.

numpy: module incontournable du calcul scientifique

Tkinter : interface graphique ….

Modules : Importation

Commande	Rôle	Exemple
import nom_module	Chargement d'un module	>>> import math
import nom_module as nom	Nommer un module	>>> import math as m >>> m.sqrt(2)
from nom_module import *	Chargement de l'intégralité du module	>>> from math import *
from nom_module import nom_fonction	Chargement d'une ou plusieurs fonctions d'un module	>>> from random import randint >>> randint(0,10) 7
dir(nom_module)	Affichages des fonctions du module	>>> dir(math)
help(nom_module.fonction) ou help(nom_module)	Aide sur une fonction ou un module	>>> help(m.ceil) >>> help(math)

Modules

- Dans le cas où on charge un module dans son intégralité, on peut directement utiliser ses fonctions mais il y a problème d'encombrement de l'espace mémoire et risque de conflit entre deux fonctions de modules ayant le même nom.
- Les modules peuvent être écrits en C/C++
- Les modules contiennent des fonctions supplémentaires

Modules

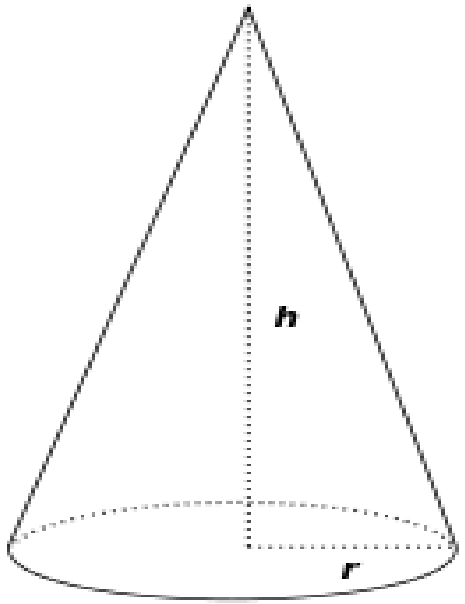
- Python possède une bibliothèque standard (ensemble de modules) accessible sans importation explicite de ses modules
- Tout programme, quelque soit sa taille, va utiliser la plupart de ses modules de façon directe ou indirecte

Modules

- Les programmes (scripts) et les modules python sont différenciés seulement par la manière d'appel
  - Les fichiers *.py qui sont précédés par import sont des modules
  - Tous les autres fichiers *.py sont des programmes qui s'exécutent directement
- Le même fichier *.py peut être à la fois un programme et un module

Exercice

- Écrire un programme qui, à partir de la saisie d'un rayon R et d'une hauteur h en cm, calcule puis affiche le volume d'un cône suivant cette formule : $\pi \times R^2 \times h / 3$.



said_anis@hotmail.com

Solution


```
from math import pi
R=float(input('entrez la valeur du rayon'))
h=float(input('entrez la valeur de la hauteur'))
print('le volume du cône est :', pow(R,2)*(pi/3)*h)
```

Les opération d'entrée sortie

said_anis@hotmail.com

Plan

1. print
2. input
3. exercice

said_anis@hotmail.com

Opération d'affichage : print()

- Syntaxe:

```
print ("Message")
print (Expression)
print (Item1, Item2, ..., ItemN)
```

- permet d'afficher sur la console n'importe quel nombre de éléments fournies en arguments
- Par défaut, ces éléments seront séparées les unes des autres par des virgules, sont affichés avec un espace entre eux, et le tout se terminera par un saut à la ligne.

Opération d'affichage : print()

- On peut remplacer le séparateur par défaut (l'espace) par un autre caractère quelconque (ou même par aucun caractère), grâce à l'argument « sep ».

Exemple 1:

```
>>> print ("Bonjour", "à", "tous", sep ="*")
Bonjour*à*tous
>>> print ("Bonjour", "à", "tous", sep ="")
Bonjouràtous
```

Opération d'affichage : print()

- De même, on peut remplacer le saut à la ligne terminal avec l'argument « end » :

Exemple 2 :

```
>>> x=12
>>> print ("x=", x, end =" ; ")
```

Opération de lecture : input()

- La plupart des scripts élaborés nécessitent à un moment ou l'autre une intervention de l'utilisateur (entrée d'un paramètre, clic de souris sur un bouton, etc.).
- **input()** : provoque une interruption dans le programme courant et l'utilisateur est invité à entrer des caractères au clavier et à terminer avec la touche <*Entrée*>.

said_anis@hotmail.com

Opération de lecture : input()

- Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour, **toujours**, une **chaîne de caractères** correspondant à ce que l'utilisateur a entré.
- Cette chaîne peut alors être assignée à une variable quelconque, convertie en une valeur numérique du type désiré par l'intermédiaire des fonctions de conversions.
  - **int()** (si on attend un entier)
  - **float()** (si on attend un réel).

Opération de lecture : input()

- On peut invoquer la fonction **input()** en laissant les parenthèses vides.
- On peut aussi y placer en argument un message explicatif destiné à l'utilisateur.

Exemple 1:

```
>>> prenom = input("Entrez votre prénom : ")
>>> print("Bonjour,", prenom)
```

Exemple 2:

```
>>>  print("Veuillez entrer un nombre positif quelconque : ", end=" ") ;
ch = input() ; nn = int (ch) ; print ("Le carré de", nn, "vaut", nn**2)
```

Opération de lecture : input()

Exemple 3 :

```
>>> a = input("Entrez une donnée numérique : ")
Entrez une donnée numérique : 52.37
>>> type(a)
<class 'str'>
>>> b = float(a) # conversion de la chaîne en un nombre réel
>>> type(b)
<class 'float'>
```

Les structures conditionnelles

said_anis@hotmail.com

Plan

1. Structures conditionnelles simples
2. Structures conditionnelles généralisées

said_anis@hotmail.com

Structures conditionnelles simples

- ***Syntaxe***

```
if condition:
        bloc_instruction1
else :
        bloc_instruction2
```

Structures conditionnelles simples

Indentation de bloc :

- Un bloc est défini par une indentation obtenue en décalant le début des instructions vers la droite grâce à des espaces en début de ligne (habituellement 4 espaces mais ce n'est pas obligatoire).
- Toutes les instructions d'un même bloc doivent être indentées exactement au même niveau (c'est-à-dire décalées à droite d'un même nombre d'espaces). L'irrespect de l'indentation génère une erreur.

Structures conditionnelles simples

Exemple 1

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note>=10.0:
    print("J'ai la moyenne")
else:
    print("C'est en dessous de la moyenne")
print("Fin du programme")
>>>
Note sur 20 : 15
J'ai la moyenne
Fin du programme
```

Structures conditionnelles simples

- ***Deux niveaux de test :***
- ***Syntaxe :***

```
if condition1:
if condition1_1:
bloc_instruction1
else:
bloc_instruction2
```

said_anis@hotmail.com

Structures conditionnelles simples

Exemple 1

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note>=10.0:
   if note>=13:
   print("mention Bien")
   else:
   print("mention Passable")
else:
   print("C'est en dessous de la moyenne")
print("Fin du programme")
>>>
Note sur 20 : 15
J'ai la moyenne
Fin du programme
```

Structures conditionnelles généralisée

- ***Syntaxe en utilisant else if***

```
if condition_1:
    bloc_instruction_1_1
else :
    if condition_2:
        bloc_instruction_1_2
    else :
        bloc_instruction_2_2
else:
   bloc_instruction _1_2
```

- ***Syntaxe en utilisant elif***

```
if expression 1:
    bloc d'instructions 1
elif expression 2:
    bloc d'instructions 2
elif expression 3:
    bloc d'instructions 3
else:
    bloc d'instructions 4
# suite du programme
```

Exemple :

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note>20.0 or note<0.0:
   print("Note invalide !")
else:
   if note>=10.0:
   print("J'ai la moyenne")
   if note==20.0:
   print("C'est même excellent !")
   else:
   print("C'est en dessous de la moyenne")
   if note==0.0:
   print(" lamentable !")
print("Fin du programme")
>>>
Note sur 20 : 20
J'ai la moyenne
C'est même excellent !
Fin du programme
>>>
Note sur 20 : 3
C'est en dessous de la moyenne
Fin du programme
```

said_anis@hotmail.com

Exemple :

```
note = float(input("Note sur 20 : "))
if note==0.0:
    print("C'est en dessous de la moyenne")
    print("... lamentable !")
elif note==20.0:
    print("J'ai la moyenne")
    print("C'est même excellent !")
elif note<10.0 and note>0.0:
# ou bien : elif 0.0 < note < 10.0:
    print("C'est en dessous de la moyenne")
elif note>=10.0 and note<20.0:
# ou bien : elif 10.0 <= note < 20.0:
    print("J'ai la moyenne")
else:
    print("Note invalide !")
    print("Fin du programme")
```

```
>>>
Note sur 20 : 20
J'ai la moyenne
C'est même excellent
!Fin du programme
```

Exercice :

- Une année est bissextile (contient 366 jours) si elle est multiple de 4, sauf les années de début de siècle (qui se terminent par 00) qui ne sont bissextiles que si elles sont divisibles par 400.

Exemples :

- 1980 et 1996 sont bissextiles car elles sont divisibles par 4
- 2000 est une année bissextile car elle est divisible par 400
- 2100 et 3000 ne sont pas bissextiles car elles ne sont pas divisibles par 400.

Ecrire un script python qui permet de déterminer si un entier positif donné correspond à une année bissextile ou non.

Correction :


```
annee= int (input ("saisir une annee"));
if annee % 4 != 0:
      print("année non bissecstile")
elif annee % 100 == 0 :
     if annee % 400 == 0 :
          print("année bissecstile")
     else :
          print("année non bissecstile")
else :
     print("année  bissecstile")
```

Les structures itératives

said_anis@hotmail.com

Plan

1. La boucle « for »
2. La boucle « while »
3. Utilisation avancée des boucles

Structures itératives : for

- ***Syntaxe***

```
for élément in itérable :
    instructions
```

said_anis@hotmail.com

Structures itératives : for

- Grâce à la fonction **range([début], fin, [pas])** la boucle **for** peut parcourir un intervalle de début à fin-1 ou en définissant la borne initiale.
- Elle peut également parcourir d'autres objets altérables (listes, chaines, tuples, ensembles, dictionnaires, tableaux…).

Exemple 1 :

```
for i in range(5):
    print(i)
print("Fin de la boucle")
```

```
>>>
0
1
2
3
4
Fin de la boucle
```

said_anis@hotmail.com

Structures itératives : for

Exemple 2 :

```
for compteur in range(1,11):
    print(compteur, '* 9 =', compteur*9)
print("Fin de la boucle")
```

```
>>>
(1, '* 9 =', 9)
(2, '* 9 =', 18)
(3, '* 9 =', 27)
(4, '* 9 =', 36)
(5, '* 9 =', 45)
(6, '* 9 =', 54)
(7, '* 9 =', 63)
(8, '* 9 =', 72)
(9, '* 9 =', 81)
(10, '* 9 =', 90)
Fin de la boucle
```

Structures itératives : for

Exemple 3 :

```
for i in range(1,8,2):
    print(i**2)
print("Fin de la boucle")
```

```
>>>
1
9
25
49
Fin de la boucle
```

said_anis@hotmail.com

Structures itératives : for

Importance des indentations : Comparer les deux programmes :

#1er programme

```
>>> x = y = 1
>>> for i in range(5):
        x += 1
        y += 1
>>>print (x,y)
```

#2ème programme

```
>>
>>
```

Résultats :

Programme 1 : 6,6 Programme2 : 6,2

Structures itératives : while

- ***Syntaxe***

```
while expression :
    bloc instructions
```

said_anis@hotmail.com

Structures itératives : while

Exemple 1 :

```
while compteur<5:
    print(compteur, compteur<5)
    compteur += 1
print(compteur<5)
print("Fin de la boucle")
```

```
>>>
1 True
2 True
3 True
4 True
False
Fin de la boucle
```

said_anis@hotmail.com

Utilisation avancée des boucles

L'instruction break

- L'instruction break provoque une sortie immédiate d'une boucle while ou d'une boucle for.
- **Exemple :**

```
while True:
    n=int(intput("Entrez un nombre 0 pour arrêter") )
if n==0 :
    break
    somme=somme+n
print("la somme des nombes est",somme)
```

- Dans cet exemple, l'expression True est toujours vraie : on a une boucle sans fin.
- L'instruction break est donc le seul moyen de sortir de la boucle.

said_anis@hotmail.com

Utilisation avancée des boucles

L'instruction continue

- Permet de passer immédiatement à l'itération suivante de la boucle for ou while;
- **Exemple :**

```
>>> for x in range(1, 11):
... if x == 5:
... continue
... print(x, end=" ")
...
1 2 3 4 6 7 8 9 10
>>> # la boucle a sauté la valeur 5
```

said_anis@hotmail.com

Utilisation avancée des boucles

while – else et for- else

- Les boucles while et for peuvent posséder une clause **else** qui ne s'exécute que si la boucle se termine normalement, c'est-à-dire sans interruption :

Exemple

```
y = int(input("Entrez un entier positif : "))
while not(y > 0) :
    y = int(input('Entrez un entier positif, S.V.P. : '))
x = y // 2
while x > 1:
    if y % x == 0:
        print(x, "a pour facteur", y)
        break # voici l'interruption !
    x -= 1
else :
    print(y, "est premier.")
```

Exercice 1

Question :

Ecrire un script python qui permet de déterminer si un entier donné est totalement divisible par 2

Exemple :

8 est totalement divisible par 2 car :

8 div 2 = 4,

4 div 2 = 2,

2 div 2 = **1**

24 n'est pas totalement divisible par 2 car :

24 div 2 = 12,

12 div 2 = 6,

6 div 2 = **3**

said_anis@hotmail.com

Solution : Exercice 1


```
n1 = int(input("Entrez un entier  "))
n = n1
while n % 2 == 0 and n ! = 1 :
    n=n//2
    print(n)
else :
    print(n1, " totalement divisible par  2")
```

said_anis@hotmail.com

Exercice 2

Question :

Ecrire un script python qui demande à l'utilisateur un nombre entier, puis affiche une liste de tous les diviseurs de ce nombre.

said_anis@hotmail.com

Solution : Exercice 2


```
n = int(input("Entrez un entier  "))
for nb in range (1,n+1):
    if n % nb == 0:
        print(nb)
```

said_anis@hotmail.com

Exercice 3

Enoncé :

- Générer un nombre aléatoire entre 1 et 9 (y compris 1 et 9).
- Demandez à l'utilisateur de deviner le nombre, puis leur dire qu'ils devinaient trop faible, trop élevé, ou tout à fait exact.

Extras:

- Garder le jeu en cours d'exécution jusqu'à ce que l'utilisateur tape "exit"
- Quand le jeu se termine, afficher le nombre d'essais faites pour arriver au bon choix.

said_anis@hotmail.com

Solution 3

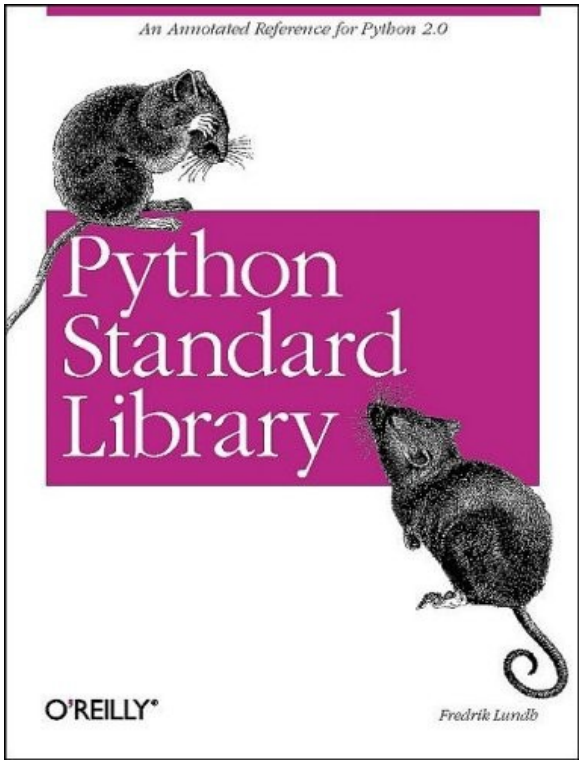

```
import random
nombre = random.randint(1,9)
choix = 0
compteur= 0
while choix != nombre and choix != "exit":
    choix = input(“Votre choix?")
    if choix == "exit":
        Break
    choix = int(choix)
    compteur += 1
    if choix < nombre:
        print("Très faible!")
    elif choix > nombre:
        print("Très élevé!")
    else:
        print("Tout à fait exact!")
        print(“Vous avez fait",compteur," essais!")
```

said_anis@hotmail.com

Et vraiment beaucoup de choses à savoir ….

said_anis@hotmail.com

Livres


An Annotated Reference for Python 2.0
Python
Standard
Library
O'REILLY
Fredrik Lundh

Liens utiles

- www.python.org
- www.python.org
- www.docs.python.org
- www.codecademy.com/tracks/python
- http://effbot.org/media/downloads/librarybook-core-modules.pdf

ESSAYEZ-LE, VOUS FINIREZ PAR L'AIMER ☺

said_anis@hotmail.com