II.1 Python

Un langage de programmation est un jeu d'instructions élémentaires donné avec un ensemble de règles d'assemblage permettant de réaliser n'importe quel calcul. Nous présentons ici une introduction très succincte à Python, langage autorisant la programmation impérative et la programmation orientée-objet, portable sur la plupart des OS (Unix, Mac Os, Windows, ...), placé sous la licence BSD (pour Berkeley Software Distribution) et disponible gratuitement, aussi bien utilisable pour écrire des programmes de quelques lignes que des projets très complexes. Le site officiel de Python est http ://www.python.org/. Vous pourrez y télécharger le langage dans la version correspondant au système d'exploitation que vous utilisez, ainsi que de la documentation. En particulier, le tutoriel de Guido van Rossum, l'auteur principal de Python, est bien conçu pour apprendre le langage (http ://docs.python.org/tut/). Un très bon ouvrage en français est Apprendre à programmer avec Python, de Gérard Swinnen, Eds O'Reilly.

II.2 Programmer en python

Un programme écrit dans un langage de programmation donné est une suite d'instructions élémentaires choisies parmi les instructions proposées par le langage et respectant des règles d'assemblage, une syntaxe, précisément définies par le langage. Un programme est écrit à l'aide d'un éditeur de texte, comme Emacs, et est appelé code source. Il ne constitue pas encore un objet directement exécutable par par la machine. Pour pouvoir être exécuté, il faut qu'il soit traduit en instructions directement exécutables par la machine, 'est-à-dire en langage machine. Deux techniques sont utilisées pour cela, l'interprétation et la compilation. Dans la première, le programme source est analysé et traduit à la volée en langage machine, les instructions résultantes étant immédiatement exécutées. Dans la seconde, un compilateur analyse et traduit le programme source en un programme objet qui peut à son tour être directement exécuté.

Un programme écrit en Python peut être aussi bien interprété que compilé : nous n'utiliserons que la première technique. Les programmes sources seront écrits dans des fichiers dont le nom aura l'extension .py, par exemple PremierProgramme.py. Ils seront exécutés par la commande python PremierProgramme.py lancée dans un terminal.

II.3 Types Python

Python est un langage à typage dynamique. Cela signifie que bien que gérant différents types, lorsqu'une variable est affectée, l'interpréteur trouvera automatiquement son type sans que l'utilisateur soit contraint de le préciser.

- Booléen : True/False (type bool pour boolean)
- Entier : Nombre entier relatif (type int pour integer). Pas de limites (sauf la mémoire de la machine). 0x prefixé pour l'hexadécimal et 0 préfixé pour l'octal.
- Nombre à virgule flottante : type float. Il est spécifié avec un ≪ . ≫ou un exposant. Peut être -/+inf (infinity) ou nan (not a number).
- Complex : type complex, il est constitué de deux float. exemple : $2 + 5j$.
- Chaines de caractères : type str pour String. Représente simplement du texte. Il est encadré par des guillemet ≪ " ≫ou apostrophes ≪ ' ≫.

La fonction type vous donne le type d'un élément. Essayer avec *type*("*toto*").

Il existe plusieurs fonctions qui permettent de convertir le type d'une variable en un autre type :

- *int*() : permet de modifier une variable en entier.
- *float*() : permet la transformation en flottant.
- *complex*() : permet la transformation en nombre complex.
- *str*() : permet de transformer la plupart des variables d'un autre type en chaînes de caractère. Deux chaines de caractères se concatène avec l'opérateur ≪ + ≫.
- *repr*() : similaire à str.
- *eval*() : évalue le contenu de son argument comme si c'était du code Python.

Exercice II.1 Convertissez différent type et vérifiez les avec la fonction type.

II.4 Opération sur les nombres

$x + y$	Plus
$x - y$	Moins
$x * y$	Multiplier
x/y	Diviser
$x//y$	Résultat entier d'une division
$x\%y$	Modulo
$power(x, y)$	Puissance
$x * *y$	Puissance
$divmod(x, y)$	diviseur et reste
$x\|y$	ou logique
$x\&y$	et logique
$x\hat{}\ y$	ou exclusif logique
$x >> y$	décalage binaire à droite
$x << y$	décalage binaire à gauche

Exercice II.2

1. Essayez les différents opérateur et regarder le type de la valeur de retour.
2. Vérifier les priorité opératoire en python

II.5 Variables

Une variable est un objet qui possède un nom qui fait référence à un espace de la mémoire vive de l'ordinateur dans lequel est stockée une donnée : la valeur de la variable.

Les noms des variables en Python sont des séquences de lettres et de chiffres qui doivent commencer par une lettre et ne pas contenir de caractères accentués ou spéciaux. Les lettres minuscules et majuscules sont distinguées. Quelques mots réservés par le langage (comme print) ne doivent pas être utilisés comme nom de variables, afin de ne pas créer d'ambiguïté. Un programme est d'autant plus lisible que le nom des variables reflète leur sens dans le programme. Par exemple, si des variables doivent désigner la hauteur et la largeur d'un rectangle, on préférera utiliser les noms hauteur et largeur plutôt que x et y.

Exemple :

```
n=2
m=3
n+m
```

Il est possible d'affecter plusieurs variables à la fois :

```
x,y = 1,2
x
y
```

Voir même de permuter deux variables :

```
x,y = 1,2
x,y = y,x
x
y
```

Dans d'autres langage cette opération nécessiterait une troisième variable.

```
z=x
x=y
y=z
```

Exercice II.3 Expérimentez les deux façon d'affecter les variables.

Les mot suivant sont résévé par le language, vous ne pourrez pas vous en servir comme nom de variables :

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 yield

Il existe un raccourci pour écrire $a = a + b$ c'est $a+ = b$.

Exercice II.4 Expérimenter ce raccourci pour les opérateurs listé.

II.6 interactions

En python, il existe deux fonction spéciales qui permettent l'intéraction avec l'utilisateur :

— *print("un message", "deux messages") : affiche un message à l'écran.*

— input("un message") : demande une entrée clavier après avoir affiché un message.

```
x=input("Entrez quelque chose au clavier")
print("vous venez d'entrer : ",x)
```

Pour utiliser la fonction *input*, il vaut mieux écrire le programme dans un fichier.py et l'éxécuter avec python ou simplement utiliser un IDE.

Exercice II.5

1. Ecrire un programme qui converti des degrès en radian. ex $90° => 0.5$
2. Ecrire un programme qui converti des celsius en fahrenheit $(C\frac{9}{5}) + 32 = F$

II.7 Expressions booléennes

Les comparateurs :

<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal
==	égal
!=	différent
is	identité d'objet
is not	contraire de l'identité d'objet

Les Opérateur booléens :

x or y	si x est faux, alors y, sinon x
x and y	si x est faux, alors x, sinon y
not x	si x est faux, alors True, sinon False

Exercice II.6

1. Combinez différent éléments que vous connaissez. Peut-on comparer facilement deux element de type différent ?
2. Comprendre le programme suivant.

```
msg1 = "ha"
msg1 += "ha" # a+=b <=> a=a+b
msg2 = "haha"
msg3=msg1

print(msg1 == msg2)
print(msg1 is msg2)
print(msg1 is msg3)
```

II.8 Les conditions

Les programmes précédents sont des séquences d'instructions élémentaires qui sont toutes exécutées les unes à la suites des autres. Un calcul doit donner la possibilité de n'effectuer une opération que si telle donnée vérifie telle condition. On appelle instruction d'exécution conditionnelle les instructions permettant ce contrôle à l'exécution du programme. On utilise essentiellement deux instructions d'exécution conditionnelle :

```
...
if (condition):
  instruction1
...
et
...
if (condition):
  instruction1
else:
  instruction2
...
```

Dans le premier cas, l'instruction 1 n'est exécutée que si la condition est vérifiée ; dans le second cas, l'instruction 1 est exécutée ssi la condition est vérifiée et l'instruction 2 est exécutée ssi la condition n'est pas vérifiée. Une condition est une expression logique qui, lorsqu'elle est exécutée, est soit vraie, soit fausse. Les conditions élémentaires portant sur des expressions numériques sont construites à l'aide d'opérateurs booléens vu plus haut.

Exemple :

```
n=input('Entrez un nombre : ')
if (n\%2==0): # n est pair ssi le reste de la
  # division de n par 2 est nul
  print n,'est pair'
else:
  print n,'est impair'
```

Remarque importante : un bloc d'instructions est une suite d'instructions dont l'exécution est solidaire. Le programme précédent peut être défini comme un bloc de deux instructions dont la première est élémentaire et la seconde est composée. La partie else de la seconde instruction est un bloc composé d'une instruction composée. En Python, les instructions composant un même bloc doivent être indentées de la même manière, c'est-à-dire avec le même décalage. Lorsqu'on utilise un éditeur comme vim ou IDLE ou notepad++, configuré pour écrire des programmes en Python, les indentations correspondent aux tabulations et sont proposées automatiquement lors de l'écriture.

Il existe un version plus concise pour écrire des if then else imbriqué :

```
if condition:
  instruction1
elif condition2:
  instruction2
elif condition3:
  instruction3
else:
  instruction4
```

Exercice II.7

1. Comment serait écrite la version non consise à trois conditions ?
2. Reprenez le convertisseur celsius pour demander à l'utilisateur dans quel sens il veut faire la conversion.

3. Ecrivez un programme demandant à l'utilisateur de saisir une année, et affichant si cette année est bissextile ou non. Rappel : Une année est bissextile si l'année est divisible par 4 et non divisible par 100, ou bien s'il est divisible par 400 : 1900 n'est pas bissextile, 2000 l'est, 2001 ne l'est pas, 2004 l'est.
4. Créer un programme qui calcule les racines d'un polynome de second degrès. Après avoir demandé à l'utilisateur les coéfficients.

II.9 les boucles While

Les instructions précédentes ne permettent pas d'écrire un programme qui demande un nombre n à l'utilisateur et affiche les n premiers termes de la suite. Pour cela, il est nécessaire de pouvoir répéter l'exécution d'un bloc de programme un certain nombre de fois défini au cours de l'exécution du programme. L'instruction while permet une telle répétition. Sa syntaxe en Python est définie par

```
...
while (condition):
  bloc d'instructions
...
```

Tant que la condition est vérifiée, le bloc d'instructions dépendant de cette condition est exécutée. Soit $u_0 = 1$ et $\forall n > 0, u_{n+1} = 3u_n + 1$ Le programme suivant affiche les n premiers termes de la suite définie ci-dessus.

```
u=1
nbtermes=input('Combien de termes souhaitez-vous afficher : ')
compteur=1
while (compteur<=nbtermes):
  print u
  u=3*u+1
  compteur=compteur+1
```

Commentaires :
— les conditions ont la même structure que dans les instructions conditionnelles ;
— si la condition est fausse, le bloc qui en dépend n'est pas exécuté ;
— si la condition est vraie, le bloc est exécuté puis la condition est réévaluée ;
— si la condition ne devient jamais fausse, le bloc est réexécuté indéfiniment !

Exercice II.8

1. Ecrire un programme calculant la somme de la suite arithmétique $u_{n+1} = u_n + 3$.
2. Ecrire un programme calculant $n!$.
3. Ecrire un programme vérifie qu'un nombre est premier.
4. Ecrire un programme calculant le PGCD de deux nombre à l'aide de l'algorithme d'euclide.
5. Ecrire un programme calculant le n ieme element de la suite de fibonacci : $u_{n+2} = u_{n+1} + u_n$, $u_0 = u_1 = 1$.
6. (Bonus) Ecrire un convertisseur decimal binaire.

II.10 les listes

II.10.1 Définition d'une liste

Une liste est définie comme une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets :

```
jours = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi','samedi', 'dimanche']
ventes = [85, 82, 90, 83, 96, 100, "pas de vente le dimanche"]
premiers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Dans cet exemple, la variable jours désigne une liste composée de chaînes de caractères, la variable ventes désigne une lsite composée d'entiers et d'une chaîne de caractères et enfin la variable premiers désigne la liste composée des 10 premiers nombres premiers.

II.10.2 Accès aux éléments, extraction d'une sous-liste

Dans une liste les éléments sont ordonnés, on peut donc directement accéder à un élément à une position données, ou encore considérer tous les élements compris entre deux positions. **Attention !** En informatique, les numérotations des listes et des tableaux commencent en général à 0. C'est le cas en Python. Ainsi le i-ème élément de la liste est en position $i-1$.

```
>>> print jours[0]
lundi
>>> print jours[1]
mardi
>>> print jours[7]
IndexError: list index out of range
```

Il est parfois utile d'accéder au dernier élément d'une liste dont on ne connaît pas le nombre d'élément. En Python, le dernier élement d'une liste peut être désigné par l'indice -1, l'avant-dernier par l'indice -2, etc.

```
>>> print jours[-1]
dimanche
>>> print jours[-2]
samedi
>>> print jours[1:3]
['mardi', 'mercredi']
>>> print jours[:3]
['lundi', 'mardi', 'mercredi']
>>> print jours[3:]
['jeudi', 'vendredi', 'samedi', 'dimanche']
```

Attention lors de l'extraction d'une sous-liste aux bornes gauches inclues et aux bornes droites exclues...

II.10.3 Concaténation, répétition

On peut, comme on le fait en général pour les mots, concaténer ou répéter des listes. Les opérations correspondantes sont + et *. L'opération de répétition est par exemple utile pour créer une liste de grande taille ne contenant que des 0 :

```
>>> debut = jours[:3]
>>> fin = jours[3:]
>>> semaine = debut+fin
>>> print semaine
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi',
'dimanche']
>>> zeros = [0]*10
>>> print zeros
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> deux_uns = [1, 1]
>>> uns = deux_uns*5
```

```
>>> print uns
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

II.10.4 Opération de base

On a vu qu'on pouvait accéder directement à un élément d'une liste, on peut même le modifier directement !

```
>>> ventes[6] = 83
>>> print ventes
[85, 82, 90, 83, 96, 100, 83]
```

Longueur d'une liste. On dispose d'une fonction donnant la longueur d'une listes, c'est-à-dire le nombre d'éléments appartenant à cette liste. Cette fonction est len :

```
>>> n = len(ventes)
>>> print n
7
>>> print len(jours)
7
```

Suppression d'un élément. On peut également supprimer un élément d'une liste en indiquant la position de l'élément que l'on souhaite supprimer :

```
>>> print jours, len(jours)
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi','dimanche'] 7
>>> del jours[2]
>>> print jours, len(jours)
['lundi', 'mardi', 'jeudi', 'vendredi', 'samedi', 'dimanche'] 6
```

II.11 Boucle for

Une particularité essentielle des listes est que les éléments sont ordonnés au sein de cette liste. Ceci a permis aux concepteurs de Python de proposer un moyen de parcourir les listes de façon très simple pour l'utilisateur :

```
>>> for jour in jours:
    print jour,
lundi mardi jeudi vendredi samedi dimanche
>>> proverbe = ['La','raison','du','plus','fort','est','toujours','la','meilleure']
>>> for mot in proverbe:
    print mot
La raison du plus fort est toujours la meilleure
```

II.11.1 Range

On a souvent besoin de faire parcourir à une variable entière les entiers naturels compris entre 1 et n par exemple. Il existe en Python la fonction range qui permet la construction d'une liste contenant ces valeurs :

```
>>> liste = range(10)
>>> print liste
[0,1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> liste2 = range(5,13)
>>> print liste2
[5, 6, 7, 8, 9, 10, 11, 12]
```

On utilisera la fonction range avec un ou deux arguments. Il s'agit de bien faire attention au sens de la fonction range dans ces deux cas :

— avec l'appel range(n), la liste commence avec l'élément 0 et s'arrête à l'élément n - 1 (elle contient donc n éléments),

— avec l'appel range(n,m), la liste commence avec l'élément n et s'arrête à l'élément m - 1.

Important : L'utilisation de la fonction range combinée avec l'opérateur de parcours de liste for permet donc de réaliser la boucle "Pour" vue précédemment.

```
>>> print range(1,7)
[1, 2, 3, 4, 5, 6]
>>> for i in range(1,7):
  if (i\%2 == 0):
    print i, "est pair"
  else:
    print i, "est impair"
1 est impair
2 est pair
3 est impair
4 est pair
5 est impair
6 est pair
```

On peut également utiliser le constructeur range pour parcourir un ensemble d'indices :

```
>>> premiers = [2, 3, 5, 7, 11, 13]
>>> for i in range(len(premiers)):
    print "Le ", i+1, "eme nombre premier est ", premiers[i]
Le 1 eme nombre premier est 2
Le 2 eme nombre premier est 3
Le 3 eme nombre premier est 5
Le 4 eme nombre premier est 7
Le 5 eme nombre premier est 11
Le 6 eme nombre premier est 13
```

La mot clef *break* permet d'arrêter la boucle. La mot clef *continue* passe à l'étape suivante.

Example :

```
for i in range (-4,4):
  if i ==-1 or i==1:
    continue
  print(i)

L1=["a","b","c","d","e"]
for i in range(0,len(L1)):
  if L1[i] == "c":
    print("indice:",i)
    break
print("pas trouve")
```

Exercice II.9

1. Utilisez cette boucle pour calculer $n!$
2. Utilisez cette boucle pour calculer la somme de $\sum_{i=0}^{n} i$

Exercice II.10

On désigne sous le nom de crible d'Eratosthène (vers 276 av.J.-C - vers 194 av.J.-C), une méthode de recherche des nombres premiers plus petits qu'un entier naturel n donné. Pour ceci, on écrit la liste de tous les nombres jusqu'à n.

— On élimine 1.
— On souligne 2 et on élimine tous les multiples de 2.
— Puis on fait de même avec 3.
— On choisit alors le plus petit nombre non souligné et non éliminé ici 5, et on élimine tous ses multiples.
— On réitère le procédé jusqu'à la partie entière de la racine de n.

Faite un programme qui donne les n premiers nombres premiers.

II.11.2 Chaine de caractères

Un chaine de caractère est une liste aussi.

Exercice II.11

1. afficher chaque caractère sur une ligne
2. afficher aussi son code ASCII