Utiliser python comme une calculette

Lorsque vous démarrez l'interprète python, vous disposez en fait d'une calculette; par exemple vous pouvez taper

```
20 * 60
```

Les règles de "priorité" entre les opérateurs sont habituelles, les produits et divisions sont évalués en premier, ensuite les sommes et soustractions:

```
 2 * 30 + 10 * 5
```

De manière générale, il est recommandé de bien parenthéser ses expressions. De plus, les parenthèses facilitent la lecture d'expressions complexes. Par exemple, il vaut mieux écrire

```
(2 * 30) + (10 * 5)
```

Rappelez vous des opérateurs suivants qui sont très pratiques

signe	opération
`/`	quotient
`%`	modulo
`**`	puissance

```
48 / 5
```

```
48 % 5
```

```
2 ** 10
```

Vous pouvez facilement faire aussi des calculs sur les complexes; souvenez vous seulement que la constante complexe que nous notons en français `i` se note en anglais `j`

```
(2 + 3j) * 2.5j
```

Aussi pour entrer ce nombre complexe `j` il faut toujours le faire précéder d'un nombre, donc ne pas entrer simplement `j` (qui serait compris comme un nom de variable, nous allons voir ça tout de suite) mais plutôt `1j` ou encore `1.j`, comme ceci

```
1j * 1.j
```

Utiliser des variables

Il peut être utile de stocker un résultat qui sera utilisé plus tard, ou de définir une valeur constante; pour cela on utilise tout simplement une affectation comme ceci:

```
largeur = 5
```

Puis

```
largeur * 20
```

```
largeur * 10
```

Pour les symboles mathématiques, on peut utiliser la même technique

```
pi = 3.14159
2 * pi * 10
```

Ou encore plus simplement, utiliser les valeurs prédéfinies par la librairie mathématique de python. En anticipant un peu sur la notion d'importation que nous approfondirons plus tard, on peut écrire:

```
from math import e, pi
```

Et ainsi imprimer les racines troisièmes de l'unité par la formule

$r_n = e^{2i\pi \frac{n}{3}},$ pour $n\in {0,1,2}$

```
n = 0
print "n=", n, "racine = ", e**((2*pi*1j*n)/3)
n = 1
print "n=", n, "racine = ", e**((2*pi*1j*n)/3)
n = 2
print "n=", n, "racine = ", e**((2*pi*1j*n)/3)
```

Remarque: bien entendu il sera possible de faire ceci plus simplement lorsque nous aurons vu les boucles `for`.

Attention aux types

Ce qui change par rapport à une calculatrice standard est le fait que les valeurs sont typées (essentiellement, entier ou flottant). Il est important de bien suivre mentalement le type de l'opération.

```
100 / 6
```

Ici nous avions affaire à deux entiers, la division `/` est alors une division ***entière*** (c'est-à-dire un quotient), alors que si l'une des valeurs est flottante comme ici

```
100. / 6
```

alors python va faire une "vraie" division et retourner un résultat flottant.

Conversions

Comme on le voit, il peut être nécessaire de faire volontairement une conversion de type

```
a = 100
a / 6
```

Comme dans ce cas on ne peut pas ajouter '.' à l'expression `a` pour la transformer en flottant, on peut faire dans ce cas

```
float(a) / 6
```

De manière plus générale, pour convertir un objet en un entier, un flottant, ou un string, on peut simplement appeler une fonction built-in qui porte le même nom que le type cible:

Type	Fonction
Entier	`int`
Flottant	`float`
Complexe	`complex`
Chaîne	`str`

Ainsi dans l'exemple précédent, `float(a)` représente la conversion de `a` en flottant, soit `100.`

On a illustré cette même technique dans les exemples suivants:

```
# imaginons que l'utilisateur a entré un entier en tapant '100'
chaine = '100'

# on voudrait le mettre au carré, mais ceci ne fonctionne pas:
chaine**2


# pour pouvoir faire des calculs,
# il nous faut d'abord en faire un entier
entier = int(chaine)
print 'entier=', entier, 'de type', type(entier)
print "au carré", entier**2
```

```
# ou pour convertir la même chaine en un nombre complexe
complexe = complex(chaine)
print "converti en complexe", complexe, "de type", type(complexe)
```

Entiers et bases

Les calculettes scientifiques permettent habituellement d'entrer les entiers dans d'autres bases que la base 10.

En python, on peut aussi entrer un entier sous forme binaire comme ceci

```
deux_cents = 0b11001000 ; print deux_cents
```

Ou encore sous forme octale (en base 8) comme ceci

```
deux_cents = 0o310 ; print deux_cents
```

Ou enfin encore en hexadecimal (base 16) comme ceci

```
deux_cents = 0xc8 ; print deux_cents
```

Pour d'autres bases, on peut utiliser la fonction de conversion 'int' en lui passant un argument supplémentaire:

```
deux_cents = int('3020', 4) ; print deux_cents
```

Fonctions mathématiques

python fournit naturellement un ensemble très complet d'opérateurs mathématiques pour les fonctions exponentielles, trigonométriques et autres, mais leur utilisation ne nous est pas encore accessible à ce stade et nous les verrons ultérieurement.