EXPRESSIONS
TYPES ET OPERATIONS

I) Expressions et types simples

Expression

Une **expression** est une suite de caractères définissant une valeur.
Cette expression peut être une **constante** (12, 15.3), un **nom de variable** (valeurs littérales comme *x* ou *longueur*), une **expression mathématique** entre parenthèses (3 + 5), la **composée** de plusieurs expressions réunies à l'aide d'un **opérateur** (comme par exemple 5 + (2 * 6)**5), la **composée d'une fonction** appliquée à d'autres expressions (comme sin(2*3.14)).

La valeur est obtenue en évaluant l'expression. Cette valeur possède un certain **type** : la valeur peut être de type "entier" (comme 3), de type "flottant" (comme 3.1), de type "chaine de caractères" (comme 'Bonjour') pour ne citer que les plus simples.
L'expression, elle, n'a pas de type a priori : cela dépend de sa valeur, de l'état des variables au moment de l'évaluation. Ainsi dans les exemples suivants le type de a + 2 n'est pas le même lorsque a valait 2 et lorsque a valait 2.0 .
Par exemple avec Python 2.7, on a :

```
>>> a = 2
>>> type(a + 2)
<type 'int'>
>>> a = 2.0
>>> type(a + 2)
<type 'float'>
```

Avec Python 3.x, les réponses s'écrivent avec le terme "class" au lieu de "type"

Types simples

Entiers

Le cas usuel est le **type "int"** pour les entiers. Il permet de faire des calculs exacts sur ces valeurs. Pour le cas des entiers ayant plus de 11 chiffres, Python 2.7 utilise aussi le **type "long"** pour les entiers longs.. Il permet également de faire des calculs exacts mais utilise un peu plus de mémoire… Python 2.7 les repère en terminant l'écriture décimale de ces entiers par la lettre L. Avec Python 3.x, les deux types ont été regroupés dans le type 'int'.

```
>>> type(12345678912)
<type 'long'>
>>> 12345678912
12345678912L
```

Sur ces entiers, les opérations sont **l'addition** (symbole +), la **soustraction** (symbole -), la **multiplication** (symbole *), **l'exponentiation** (symbole **), l'**opposé** (symbole – en préfixe), le **quotient** dans la division euclidienne (symbole //), le **reste** dans la division euclidienne (symbole %). Attention : pour la division euclidienne par un entier négatif, la division euclidienne pour Python donne pour reste un entier entre m+1 et 0, alors qu'en maths, on donne un reste entre 0 et |m| - 1.
En l'absence de parenthèses, les règles de priorités des opérateurs (on parle de **précédence**) sont celles qui régissent les priorités des opérations mathématiques : priorité aux exponentiations, puis, à égalité, multiplications, divisions entières et modulos, puis enfin, encore à égalité, additions et soustractions.
En cas d'égalité, on utilise évalue les expressions à partir de la gauche… à l'exception notable de l'exponentiation

Flottants

Le type pour les nombres à virgule flottante (ou simplement flottants) est le **type "float"**. Attention : contrairement aux entiers, on ne peut pas avoir accès à tous les flottants imaginables. Ainsi, Python traite bien l'entier 2**2000 alors qu'il ne connait pas 2.**2000
Sur ces flottants, on retrouve les opérations +, *, -, **. Pour la **division** flottante, il s'agit du symbole /.
Remarque : lorsqu'une opération fait intervenir un entier et un flottant (ou une opération sur les flottants), Python convertit automatiquement l'entier en flottant.

On peut convertir un entier en flottant à l'aide de la fonction **float**, et on peut convertir un flottant en entier avec l'instruction **int**. Remarquons toutefois que int(x) n'est pas la partie entière du flottant x mais sa troncature.

Booléens

Le **type "bool"** est assez simple puisqu'il ne possède que 2 constantes booléennes : *True* et *False*. Ces deux valeurs représentent les résultats d'évaluation d'expressions logiques pouvant prendre soit la valeur vraie soit la valeur fausse.
Les opérateurs sur les booléens sont la négation (symbole **not**), la conjonction (symbole **and**) et la disjonction (symbole **or**).
Les règles d'évaluation de ces opérateurs sont résumées dans le tableau suivant :

b1	b2	**not** b1	b1 **and** b2	b1 **or** b2
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

Les opérateurs **and** et **or** sont dits "paresseux" : ils ne calculent que ce qui est nécessaire pour évaluer l'expression. Par exemple, dans l'expression b1 **and** b2, si b1 est fausse, il n'y a pas nécessité d'évaluer b2 pour savoir que b1 **and** b2 est fausse

```
>>> 2 < 1  and  1/0  < 2
False
>>> 1/0 < 2 and 2 < 1
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    1/0 < 2 and 2<1
ZeroDivisionError: division by zero
```

L'opérateur **not** a précédence sur **or** et **and**. L'opérateur **and** a précédence sur **or**.

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

L'apparition la plus fréquente des booléens se fait lors de comparaisons d'autres types.
On a la comparaison d'**égalité** (symbole ==), de **différence** (symbole !=), **d'inégalité stricte** (symboles < ou >), **d'inégalité large** (symboles <= ou >=).
On peut faire des comparaisons multiples du style "a < b < c" ce qui correspond à l'expression logique "a < b" **and** "b < c" .
C'est l'occasion de rappeler que le nombre flottant n'est pas le nombre décimal correspondant (puisque les écritures au standard IEEE 754 se font en base 2, base dans laquelle rares sont les nombres décimaux ayant un développement binaire fini. Ainsi les calculs sur les nombres flottants ne sont pas "exacts"…)

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
>>> 0.1 + 0.1 + 0.1 < 0.3
False
>>> 0.1 + 0.1 + 0.1 > 0.3
True
```

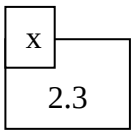
Alors que " 0.1 + 0.1 == 0.2 " est bien vraie

II) Variables

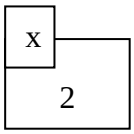
Notion de variable

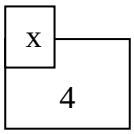
Dans un programme, une variable est une zone mémoire de l'ordinateur dans laquelle on a stocké une valeur, une fonction, un tableau …
Pour faire référence à cette zone mémoire, on utilise un nom de variable.
Un nom de variable est une chaine de caractères commençant par une lettre et contenant des lettres, des chiffres et le caractère spécial _ .

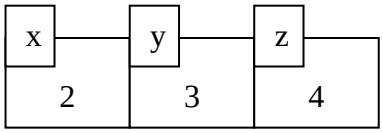
Pour la suite dans ce cours, on représentera une variable à l'aide d'un rectangle donnant la valeur de cette variable surmontée d'une petite étiquette indiquant le nom de la variable.
Par exemple, le diagramme suivant indique qu'une variable de valeur 2.3 a été stockée en mémoire et que le nom de cette variable est x.



L'ensemble des variables définies à un instant donné est appelé l'état. Cet état courant dépend des différentes expressions et instructions précédant l'étape courante. Lors de l'évaluation d'une expression, les noms des variables sont remplacés par les valeurs de ces variables.

Ainsi, l'expression x + 3 prendra la valeur 5 dans l'état 

et 7 dans l'état 

Dans l'état 

l'expression x + y * z s'évalue en 2 + 3*4

Si lors de l'évaluation d'une expression, un nom de variable est utilisé alors qu'il n'apparait pas dans l'état, l'évaluation est bloquée et Python renvoie un message d'erreur.

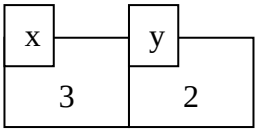
Déclaration, initialisation, affectation d'une variable

Déclarer une variable c'est l'ajouter à l'état. Dans certains logiciels, il y a des fonctions qui permettent à l'utilisateur de dire qu'il utilisera les variables de noms ….
Initialiser une variable c'est donner à une variable déclarée une première valeur.
En Python, la déclaration et l'initialisation s'effectuent avec la même instruction :

```
>>> nom_de_variable = expression
```

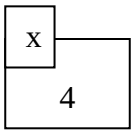
Remarque : cette instruction n'appelle pas de réponse par Python, mais elle est tout de même exécutée…

L'affectation s'écrit de la même façon. Elle permet de changer la valeur d'une variable.
Dans une telle affectation, seule la variable dont le nom est à gauche du symbole = est transformée. Toutes les variables apparaissant à droite du symbole = sont remplacées par leur valeur dans l'état courant.

Ainsi, lorsqu'à l'ouverture d'une session Python (c'est-à-dire lorsqu'initialement l'état est vide), nous effectuons les instructions suivantes : >>> x = 2 >>> y = x >>> x = 3
L'état final est alors 

Autre exemple : si on effectue les instructions : >>> x = 2 >>> x = x + 2

Dans la seconde instruction, le "x" n'a pas le même statut. Celui de gauche correspond à la nouvelle affectation de la variable de nom x, alors que dans celui de droite est transformé dans l'expression par 2. Au final, l'état final est indiqué ci-contre



Remarque : certaines opérations classiques en programmation ont un raccourci sur Python. Par exemple, le "x = x + 2" de vu précédemment peut-être écrit " x += 2". D'autres opérations-affectations existent : -=, *= sur les nombres par exemple.

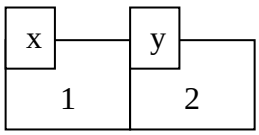
Il existe une expression particulière **input()** qui attend que l'utilisateur tape quelque chose au clavier et qui prendra pour valeur la chaine de caractères correspondante.
Ainsi l'instruction : >>> a = **input**() va mettre dans la variable de nom a la chaine de caractères qui sera tapée par l'utilisateur. Le souci est que Python n'affiche rien et on ne sait pas toujours qu'il attend quelque chose.
Aussi on peut mettre en argument de **input** une chaine de caractères précisant ce que l'on veut. Par exemple :

```
>>> a = input('donnez un entier')
donnez un entier48
>>> a
'48'
```

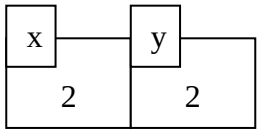
Notons plusieurs petites choses sur cet exemple :

- le premier 48 est entré par l'utilisateur après l'invitation faite par Python
- la phrase "donnez un entier" est entourée de ' ' (ou de "") : c'est la façon pour Python de reconnaître les chaines de caractères
- la valeur de la variable a n'est pas l'entier 48 mais la chaîne de caractères 48… cela signifie que l'on ne peut pas effectuer les opérations élémentaires directement avec a. Dans ce cas précis, il aurait fallu transformer la chaîne de caractères '48' en l'entier 48 en utilisant la fonction **int**

Pour terminer ce paragraphe, il convient de présenter un problème classique en informatique. On considère que 2 variables sont dans l'état et on veut échanger les valeurs de ces 2 variables. Pour fixer les idées on suppose qu'on est dans l'état



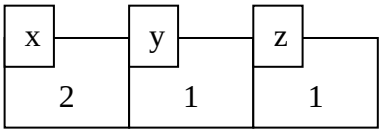
Une première approche : >>> x = y
>>> y = x
ne fonctionne pas car, alors, on est dans l'état



La méthode classique consiste à utiliser une variable intermédiaire (par exemple z), et de stocker dans cette variable la valeur d'une des variables de départ avant de la réaffecter.

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

nous conduit à l'état



III) Types composés

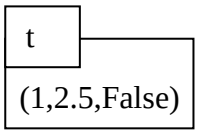
Il s'agit de valeurs formées de plusieurs valeurs de types plus simples

Les n-uplets

Le n-uplet, ***tuple*** en anglais, est la généralisation à n termes du concept de couple ou de triplet. Comme en mathématiques, le n-uplet est constitué de n valeurs séparées par des virgules et encadrées par des parenthèses. On peut donner un nom de variable à ce *tuple*.
Par exemple, à partir de l'état vide, l'instruction :

```
>>> t = (1, 2.5, False)
```

conduit à l'état

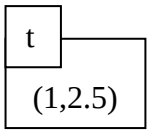


Remarque : dans l'affectation, on peut omettre les parenthèses…
Attention : Pour n = 0, on a le tuple vide noté (). **Pour n = 1**, le tuple constitué de la seule valeur v est **noté (v,)**. La virgule est essentielle pour distinguer le tuple (v) de l'expression (v) dont la valeur est v.

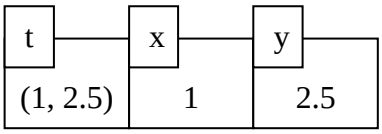
On a **accès aux** différentes **composantes** du *tuple* t en utilisant l'expression t[i] où i est l'indice de la composante. Attention cependant : Python numérote les indices à partir de 0. Ainsi, dans l'exemple précédent, t[0] vaut 1, t[1] le flottant 2.5 , et t[2] le booléen False

Par contre, le *tuple* est une variable immuable : il n'est pas possible d'affecter une nouvelle valeur à une des composantes (par contre on peut changer la variable complète…)

Il est possible de déconstruire un tuple en affectant simultanément ses composantes à différentes variables.
Ainsi en partant de l'état



l'instruction `>>> x,y = t` conduit à l'état



Remarque : on en déduit une méthode simple pour échanger les valeurs de deux variables sous Python. Il suffit simplement d'utiliser l'instruction : `>>> x,y = y,x`

On peut "coller" deux (ou plusieurs) tuples, pour en faire un autre plus grand. Il s'agit de la **concaténation** et on l'obtient en utilisant l'opérateur +

```
>>> (1,2) + (False,True) + (0,)
(1, 2, False, True, 0)
```

On peut tester si une valeur appartient à un tuple donné à l'aide de l'opérateur **in**

```
>>> t = (1,2) + (False,)
>>> False in t
True
```

On trouve la longueur d'un tuple à l'aide de la fonction **len**

```
>>> t = (1,2) + (False,)
>>> len(t)
3
```

Les chaînes de caractères

Le type des chaînes de caractères, *string* en anglais et dans Python, est celui permettant de représenter des textes. Une chaine de caractères est une suite finie de caractères consécutifs qu'on note entre apostrophes ou guillemets.
Remarque : Pour n = 0, on a la chaine de caractères vide ""

On a **accès aux** différents **caractères** d'une chaîne de caractères s en utilisant l'expression s[i] où i est l'indice de la composante. Attention cependant : Python numérote les indices à partir de 0.

```
>>> t = "Bonjour"
>>> t[0], t[2]
('B', 'n')
```

Comme pour le *tuple* , une chaine de caractères est une variable immuable : il n'est pas possible d'affecter une nouvelle valeur à une des composantes (par contre on peut changer la variable complète…)

On peut "coller" deux (ou plusieurs) chaines de caractères, pour en faire une autre plus grande. Il s'agit de la **concaténation** et on l'obtient en utilisant l'opérateur +

```
>>> "Bonjour" + " l'ami "
"Bonjour l'ami "
```

On peut tester si un caractère est dans une chaîne de caractères à l'aide de l'opérateur **in**

```
>>> a in " l'ami "
True
```

On trouve la longueur d'une chaîne de caractères à l'aide de la fonction **len**

Un ensemble de caractères consécutifs dans une chaîne donnée est une sous-chaîne. Si s est une chaîne, **s[i:j]** est la chaîne des caractères de s entre le caractère d'indice i et celui d'indice j-1. Le processus est également possible sur les tuples…

```
>>> mot = "Bonjour"
>>> mot[1:6]
'onjou'
```

On peut tester si une chaîne est une sous-chaîne d'une chaîne de caractères à l'aide de l'opérateur **in**

```
>>> "am" in " l'ami "
True
```

Lorsque c'est possible, on peut **convertir** une chaîne de caractères en un type différent.
La fonction **tuple** transforme une chaine de caractères en un n-uplet constitué des caractères de la chaîne.

```
>>> tuple(mot)
('B', 'o', 'n', 'j', 'o', 'u', 'r')
```

Les fonctions **int**, **float**, **bool**, transforment une chaîne de caractères transformable en l'entier, le flottant ou le booléen correspondant.

Les listes

Le type des listes sera étudié plus précisément dans un chapitre suivant. Une liste est un n-uplet entouré par des crochets []. La grande différence avec le tuple est que l'on a la possibilité de changer ces composantes sans changer toute la liste.

```
>>> L = [1,2,3.5]
>>> L[1] = 3
>>> L
[1, 3, 3.5]
```

Toutes les fonctions s'appliquant aux tuples s'appliquent aussi aux listes…

On peut parfois transformer une expression d'un type donnée en un autre.
Les principales fonctions le permettant sont **int**, **float**, **bool**, **list**, **str**, **tuple**…

IV) Exercices

1) Ecrire une expression permettant de déterminer les deux derniers chiffres de 20123^{25189}

2) Ecrire des expressions formées uniquement des opérateurs sur les entiers, de parenthèses et du chiffres 3 utilisé au plus 4 fois. On cherchera à obtenir le maximum de valeurs entières positives. Quel est le plus petit entier ne pouvant s'écrire ainsi ? Quel est le plus grand entier que l'on peut obtenir ?

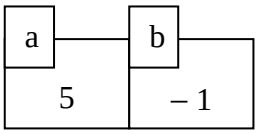
3) Que répondrait Python avec les instructions suivantes : .5 - .3 ? 4 / (9 – 3**2) ? float (7 // 2) ?

4) Quelle est la valeur des expressions booléennes suivantes : **not** (**not** True) ? 3 * 3.5 > 10 ? 3. * 7 == 21 ? 3 – 1 => 1 ? 0 < 10**-300 == 100**-150 ? **not** (2 – 1 == 1 == 4 + 3) ?

5) Ecrire des expressions booléennes traduisant les conditions suivantes. Les nombres sont des flottants.
- le point de coordonnées (x, y) est à l'intérieur du cercle de centre (z, t) et de rayon r.
- les points de coordonnées (x,y) et (z,t) sont des sommets opposés d'un carré de cotés parallèles aux axes
- Il existe un triangle de cotés a, b et c.

6) Ecrire des expressions booléennes traduisant les conditions suivantes. Les nombres sont des entiers
- l'entier n est un multiple de 5
- les entiers m et n sont tels que l'un est multiple de l'autre
- n est le plus petit multiple de 7 supérieur à 10^{100}

7) Déterminer la valeur des expressions suivantes dans l'état



- a + 2
- b – 1
- a + 2 * b
- a * a * a
- b <= 0 or a < 10
- a * b < 2

8) On considère un état dans lequel sont définies trois variables de noms x, y et z. Décrire une suite d'instructions permettant de placer les contenus de x, y et z dans respectivement z, x et y.

9) Partant de l'état initial vide, décrire l'évolution de l'état lors de l'exécution des instructions suivantes évaluées chronologiquement : • x = 3 • y = 2 • x = 1 + y * x • y = 1.2 + x • y = y

10) Ecrire une expression qui indique si les 6 voyelles de l'alphabet sont présentes dans une chaîne de caractères s

11) Ecrire une expression qui vérifie si la chaîne de caractères s commence par une majuscule et se termine par un point.