Chapitre 2: Premiers pas en python

Site web du cours pour télécharger les cours et exercices:
http://moodle2.unil.ch/course/view.php?id=826

2.1 Alphabet et vocabulaire

- **Alphabet**
  - Caractères graphiques

connus { `a .. z, A .. Z, 0 .. 9`
`, ; ? ! : ...`

moins connus:

`/`	slash
`@`	arobase, à commercial
`[ ]`	crochets gauche et droite
`{ }`	accolades gauche et droite
...	

ANNEXE

Quelques caractères spéciaux

a) Caractères graphiques

		Mac	PC
§	paragraphe		
"	guillemet		
%	symbole pour cent		
&	et (commercial), perluette		
/	slash		
\	backslash	<alt> /	<alt> <
|	barre verticale	<alt> 7	<alt> 7
@	à commercial, "queue de singe", arobase	<alt> g	<alt> 2
#	dièse	<alt> 3	<alt> 3
$	dollar		
£	pound, "livre sterling"		
~	tilde	<alt> n	<alt> ^
[	crochet gauche	<alt> 5	<alt> ü
]	crochet droite	<alt> 6	<alt>
{	accolade gauche	<alt> 8	<alt> ä
}	accolade droite	<alt> 9	<alt> $

b) Caractères non graphiques

↵	retour de chariot
→|	tabulateur
←	backspace, touche effacer
<esc>	escape - sortir, s'échapper
<ctrl>	control - contrôler
<alt>	touche alt (alternative)

Programmation *PYTHON (Résumé du langage)*

- Caractères non-graphiques
  - <DEL> <*effacer*>
  - <CR> <*retour de chariot*>
  - <TAB> <*tabulation*>
  - …

ANNEXE

Quelques caractères spéciaux

a) Caractères graphiques

		Mac	PC
§	paragraphe		
"	guillemet		
%	symbole pour cent		
&	et (commercial), perluette		
/	slash		
\	backslash	<alt> /	<alt> <
|	barre verticale	<alt> 7	<alt> 7
@	à commercial, "queue de singe", arobase	<alt> g	<alt> 2
#	dièse	<alt> 3	<alt> 3
$	dollar		
£	pound, "livre sterling"		
~	tilde	<alt> n	<alt> ^
[	crochet gauche	<alt> 5	<alt> ü
]	crochet droite	<alt> 6	<alt>
{	accolade gauche	<alt> 8	<alt> ä
}	accolade droite	<alt> 9	<alt> $

b) Caractères non graphiques

↵	retour de chariot
→|	tabulateur
←	backspace, touche effacer
<esc>	escape - sortir, s'échapper
<ctrl>	control - contrôler
<alt>	touche alt (alternative)

- **Vocabulaire**
  a) Mots réservés.

  Une trentaine de mots réservés à une utilisation bien précise dans le langage python. Ils ne peuvent pas être utilisés comme identificateurs.

  b) Symboles de ponctuation.

  → c.f. polycopié pages 1 et 2

1. SYNTAXE

1.1 Vocabulaire

a) Les mots réservés

Les mots suivants sont réservés à une utilisation bien précise dans le langage Python. Ils ne peuvent pas être utilisés comme identificateurs (voir 1.3 a).

and	et. Conjonction logique
as	traitement d'une exception, peut apparaître avec **with**
assert	permet d'introduire du code de débogage
break	rupture de l'exécution d'une boucle
class	permet de définir une classe
continue	dans le corps d'une boucle, met fin à l'itération en cours
def	abréviation de *define*. Permet de définir une fonction
del	abréviation de delete. Pour supprimer une référence à un objet
elif	abréviation de *else if*
else	sinon
except	introduit un gestionnaire d'exception
exec	abréviation de *execute*. Pour exécuter du code Python
finally	traitement d'une exception, introduit un gestionnaire de nettoyage
for	instruction de répétition i.e. introduit une boucle
from	pour définir un module d'importation
global	change la portée d'une déclaration
if	instruction si
import	importer
in	opérateur logique pour tester l'appartenance
is	opérateur pour comparer deux objets
lambda	pour introduire une lambda-expression i.e. une fonction anonyme
not	non. Négation logique
or	ou. Disjonction logique
pass	instruction vide
print	impression i.e. sortie
raise	lève une exception
return	retour de fonction
try	traitement d'une exception
while	instruction tant que
with	traitement d'une exception, une autre forme de **try**/**finally**
yield	pour introduire un générateur dans une fonction

b) «Symboles de ponctuation»

1) délimiteurs:	(	)	{	}	[	]
	,	.	:	;	`	=
	+=	-=	*=	/=	//=	%=
	&=	|=	^=	>>=	<<=	**=

2) opérateurs:	+	-	*	/	%	**
	//	<<	>>	&	|	^
	~	<	<=	>	>=	<>
	!=	==				

1.2 Grammaire

a) Définitions syntaxiques

Les éléments du vocabulaire sont entre guillemets.

::=	se récrit
( )	rôle habituel: permet de regrouper des éléments
une étoile (*)	zéro ou plusieurs fois l'élément qui précède
un plus (+)	une ou plusieurs fois l'élément qui précède
|	utilisé pour séparer des alternatives
[ ]	zéro ou une occurrence

Exemple 1-1: définitions syntaxiques de expression_list et exponent

```
expression_list ::=
    expression ("," expression)* [","]

exponent ::=
    ("e" | "E") ["+" | "-"] digit+
```

2.2 Grammaire

A) Forme de Backus-Naur (BNF)

Les grammaires sous forme BNF (Backus-Normal-Form, Backus-Naur-Form) permettent de décrire les règles syntaxiques des langages de programmation

On a les symboles suivants:

::=	signifie devient (se récrit)
()	sert à regrouper les éléments
|	signifie ou (choix)
<>	sert à délimiter des objets (ou concepts)

Exemple:

<phrase> ::= <sujet> <verbe> <complément>
<sujet> ::= (le chat | la souris)
<verbe> ::= (mange | ignore)
<complément> ::= (<sujet> | le repas)

le chat mange la souris
le chat ignore le chat
la souris mange le repas

B) Forme de Backus-Naur étendue (EBNF)

Les grammaires sous forme EBNF (Extended Backus-Normal-Form) permettent de décrire des règles grammaticales plus sophistiquées.

On a les symboles suivants:

`::=`	signifie *devient (se récrit)*
|	signifie *ou (choix)*
< >	sert à délimiter des objets (ou concepts)
[]	répétition de 0 ou 1 fois l'élément qui est entre crochet
*	répétition de 0 ou plusieurs fois l'élément précédent
+	répétition de 1 ou plusieurs fois l'élément précédent

Exemple:

<digit> ::= (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

<exponent> ::= ("e" | "E") ["+" | "-"]<digit>+

C) Diagrammes syntaxiques

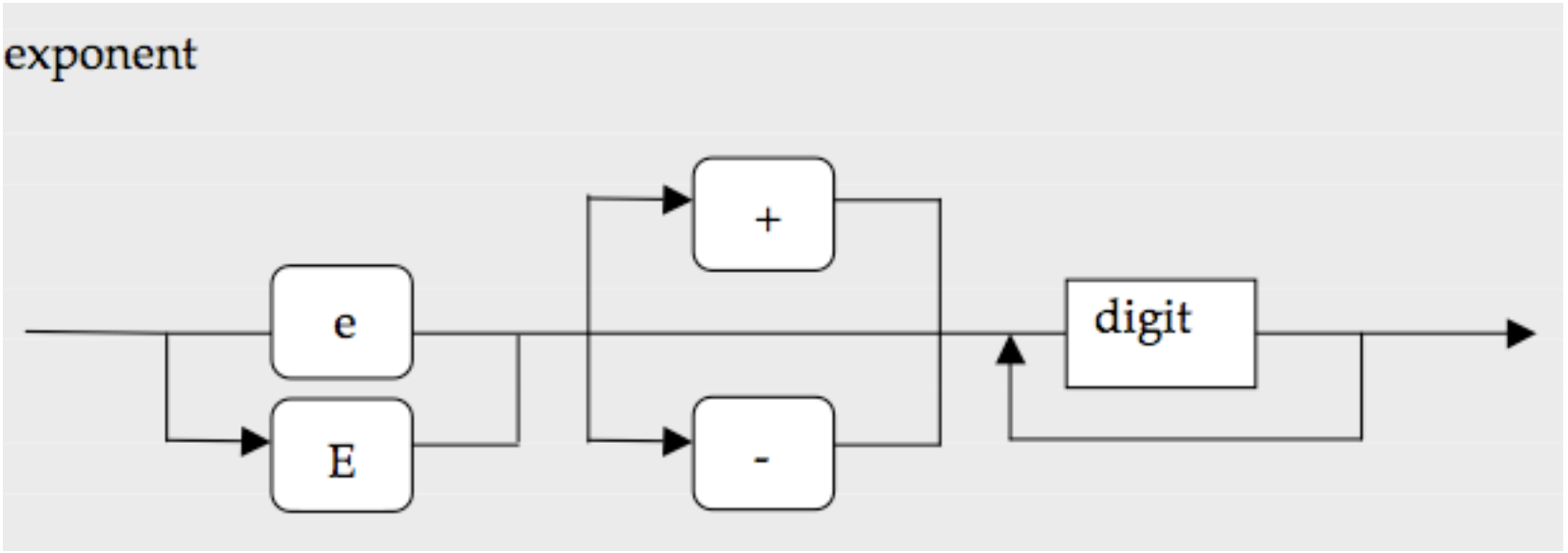
On peut représenter ces règles grammaticales sous forme de diagrammes syntaxiques.

Élément du vocabulaire

Sous-règle

Comment suivre la règle

Exemple:



2.3 Unités syntaxiques

Nous allons présenter les concepts suivants:

a) **Identificateurs**
b) **Littéraux numériques**
   - **Nombres entiers**
   - **Nombres flottants**
c) **Littéraux chaînes (de caractères)**
d) **Commentaires**
e) **Expressions arithmétiques**
   - **Unaires**
   - **Binaires**

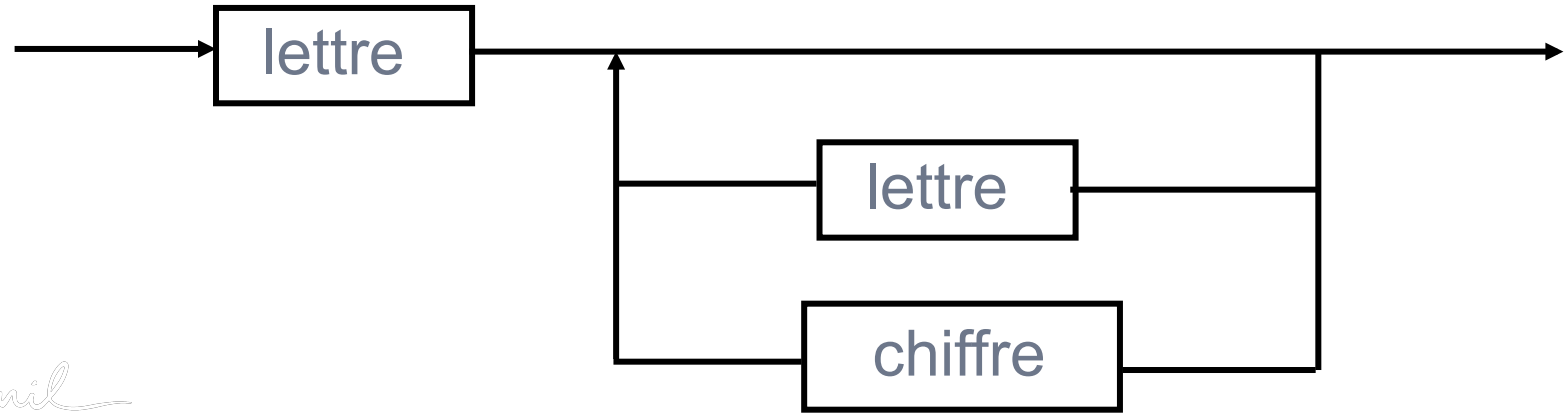
A) Identificateurs

- Un identificateur sert à identifier une variable, une fonction, une classe, un module ou tout autre objet.
- Syntaxiquement, un identificateur se constitue d'une suite de lettres et de chiffres, le premier caractère devant être une lettre.
- Les mots réservés du langage ne peuvent pas être utilisés comme identificateurs.

- Syntaxe d'un identificateur exprimée sous forme EBNF:

  <identificateurs> ::= <lettre> (<lettre> | <chiffre>)*

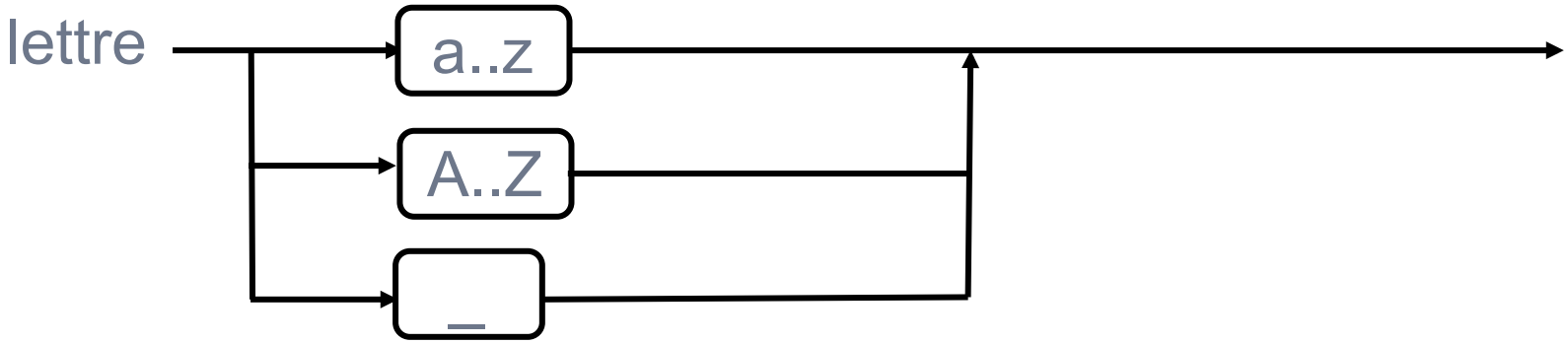
- Syntaxe d'un identificateur exprimée sous forme de diagramme syntaxique:



n'existe pas dans
la notation EBNF

« lettre » sous forme EBNF et schéma syntaxique:

<lettre> ::= "a" .. "z" | "A" .. "Z" | "_"



« chiffre » sous forme EBNF et schéma syntaxique:

<chiffre> ::= "0" .. "9"

Exemple: les expressions suivantes peuvent être utilisées comme identificateurs

```
ceciEstUnIdentificateur
create_polygon
_secret
Random
Boeing707
__init__
```

Exemple: processus d'affectation (i.e. associer un identificateur à un objet donné)

```
i = 55
x = 3.08
ch = "bla bla bla"
```

Ici, `i`, `x`, `ch` sont des identificateurs

Lorsqu'on entre ces expressions dans python, on obtient (faire petite démo):

```
>>> i=55
>>> i
55
>>> x=3.08
>>> x
3.0800000000000001
>>> ch="bla bla bla"
>>> ch
'bla bla bla'
```

B) Littéraux numériques

Les littéraux numériques sont les représentations des nombres. Ils peuvent être de deux sortes:

- **Entiers** (nombre entiers, i.e. sans virgule)
- **Flottants** (nombres à virgules)

- **Entiers:**
  - Un entier est un nombre « sans virgule », positif ou négatif.
  - « entier » sous forme EBNF:

    <entierDecimal> ::= <chiffreDifferentZero><chiffre>* | "0"
  - Attention: la mémoire d'un ordinateur étant finie, il n'est possible de représenter qu'un nombre fini d'entiers!
  - Ainsi les « entiers de l'ordinateur » sont différents des « entiers mathématiques Z ».

- Les entiers sont généralement représentés en base 10 (système décimal).
- En python, on peut aussi représenter les entier en base 8 (système octal) ou en base 16 (système exadécimal)
  - base 10 ➪ 10 signes: 0, 1, 2, …,9
  - base 8 ➪ 8 signes: 0, 1, 2, …,7
  - base 16 ➪ 16 signes: 0, 1, 2, …,9, a, b, c, d, e, f

Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	…	24	25	26	27
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	…	18	19	1A	1B

Let's Decode the Hex Number F2A4C= 993,868

Powers of 16	65,536's	4096's	256's	16's	Ones
	F	2	A	4	C

To "Decode" this number, the **NERD BRAIN**
(Or Calculator More Likely) **consciously** thinks . . .

65,536 x F = 983,040
4,096 x 2 = 8,192
256 x A = 2,560
16 x 4 = 64
1 x C = 12
+ ———
993,868

Ainsi, les entiers sont constitués d'une suite de chiffres, éventuellement précédée par une marque de base (o pour octal et ox ou oX pour héxadécimal).

Exemple: les expressions suivantes sont des entiers:

```
2514, 1998, 0777, 0xDadaCafe.
```

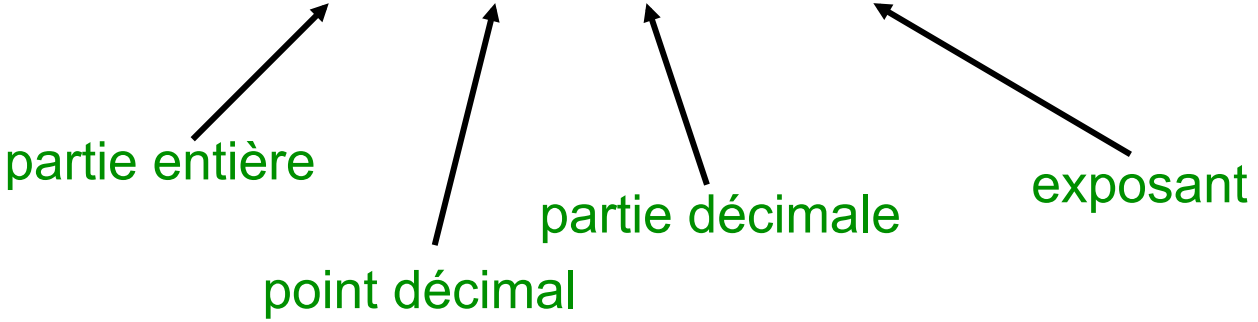
Si on entre ces entiers dans python, on obtient (faire petite démo):

```
>>> 2514
2514
>>> 1998
1998
>>> 0777
511
>>> 0xDadaCafe
3671771902L
```

- **Flottants:**
  - Les flottants sont les nombre à virgule.
  - Les flottants sont constitués des parties suivantes:
    - une partie entière;
    - un point décimal;
    - une partie décimale;
    - éventuellement un exposant (indiqué par la lettre e ou E suivi d'un entier avec ou sans signe)
  - xey ou xEy signifie $x*10^y$

- « flottant » sous forme EBNF:

```
entier ::= chiffre+
flottant ::= [entier] ["."] [entier] ["e"|"E"["+"|"-"] <entier>]
```



Remarque: en fait, les choses sont un peu plus complexes… Ici, la chaîne vide est possible. Voire la syntaxe de Python.

- Attention: la mémoire d'un ordinateur étant finie, il n'est possible de représenter qu'un nombre fini de flottants!
- Ainsi les « flottants de l'ordinateur » sont différents des « réels mathématiques R » (en effet contrairement aux réels, les flottants sont en nombre fini et ils sont tous de précision finie)

Exemple: les expressions suivantes sont des flottants:

3.14159
2.0
6.02213e+23
1.0E-9
1e10
.0
0.

C) Littéraux chaînes

Les chaînes de caractères sont des suites de symboles.

- Une chaîne de caractère peut être de deux formes différentes:
  - **chaîne courte**
  - **chaîne longue**

- **chaîne courte**: une chaîne de zéro ou plusieurs caractères entourée de simple *apostrophes* (') ou de *guillemets* ("), devant tenir sur une seule ligne.
- **chaîne longue**: une chaîne de zéro ou plusieurs caractères entourée de *trois apostrophes* (' ' ') ou de *trois guillemets* (" " ") pouvant s'étendre sur plusieurs lignes.

Remarque:
Les choses sont un peu plus complexes… si une chaîne contient des apostrophes, elle commencera et se terminera par des guillemets; si une chaîne contient des guillemets, elle commencera et se terminera par des apostrophes.

Exemple: Les expressions suivantes sont des chaînes de caractères:

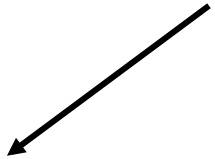
```
"Ceci est une chaine courte"
"L'arbre majestueux"
'"bien", répondit-il'
""
"""Le but de ce texte est de
decrire la maniere dont fonctionne
le programme ci-dessous"""
```

D) Commentaires

Le terme de commentaires désigne le texte qui sera tout simplement ignoré par l'ordinateur.

- Le texte qui commence par # est ignoré jusqu'à la fin de la ligne.
- Indispensable pour commenter son code (afin de pouvoir se relire).

```
# puis tout est ignoré jusqu'à la fin de la ligne
```



code Python # bla bla bla ……..

Remarque: dans les commentaires et les chaînes, il est possible d'utiliser des lettres accentuées.

Exemple: voici un exemple de code commenté

```
42              # littéral entier
3.14159         # littéral flottant
""              # chaîne vide
'hello'         # littéral chaîne
```

E) Expressions arithmétiques

Les expressions arithmétiques sont des expressions bien formées (selon des règles syntaxiques précises) impliquant:

- variables, *i.e.* identificateurs
- littéraux numériques
- opérateurs unaires
- opérateurs binaires
- (et)
- Fonctions arithmétiques

- Opérateurs unaires:
  - Les expressions sont de la forme EBNF suivante:

    <opérateur> <opérand>

  - Opérateurs unaires + et – se comportent normalement (donnent le signe d'une expression).

Exemple:

+1

-2

-35

-pi

- Opérateurs binaires:
  - Les expressions sont de la forme EBNF suivante:

    <opérand> <opérateur> <opérand>

    Exemple:

    1+1

    7-2

    2*35

    x**3

    7/2

- « + » et « – » sont les opérateurs binaires d'addition et de soustraction.

  Exemple:

  2+3 1234+b 7-12 a-b

- « * » est l'opérateur binaire de multiplication.

  Attention: le * est obligatoire pour exprimer la multiplication: a*2 signifie « a fois 2 » alors que a2 est un identificateur.

  Exemple:

  2*3 1234*b a*b

- « / » est l'opérateur binaire de division.
  - Attention: dans le cas de la division de deux entiers <entier> / <entier>, l'opérateur retourne la division entière (i.e. arrondi à l'entier inférieur).
  - Dans tous les autres cas, l'opérateur procède à une division normale (i.e. donne un résultat flottant).

Exemple:

7 / 2 ⇒ 3 (reste 1)
7 / -2 ⇒ -4
7.0 / 2 ⇒ 3.5
7 / 2.0 ⇒ 3.5
7.0 / 2.0 ⇒ 3.5

- « % » est l'opérateur binaire de modulo.

L'opération n % m retourne le reste de la division entière de n par m, i.e.

$$n \,\%\, m \Rightarrow n - (n/m)*m$$

Exemple:

7 % 2 ⇒ 1 (car 7/2 = 3 reste 1)

2 % 7 ⇒ 2 (car 2/7 = 0 reste 2)

- « ** » est l'opérateur binaire d'exponentiation, i.e. a**b ⇒ a^b

Exemple:

2 ** 2 ⇒ 4

9 ** 0.5 ⇒ 3.0

Priorités: ce sont les mêmes qu'en mathématique. En abrégé:

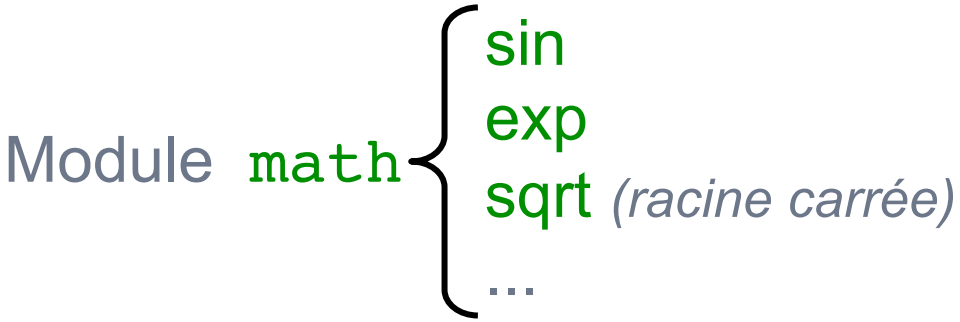
Parenthèses,

exponentiation,

opérateurs multiplicatifs,

opérateurs additifs.

- Fonctions mathématiques standards:



- Pour accéder à ces fonctions (on y reviendra)

```
from math import*
```

http://docs.python.org/lib/module-math.html

Exemple:

```
>>> from math import *
>>> a = sqrt(sin(pi/2)+3)
>>> print a
2.0
>>>
```

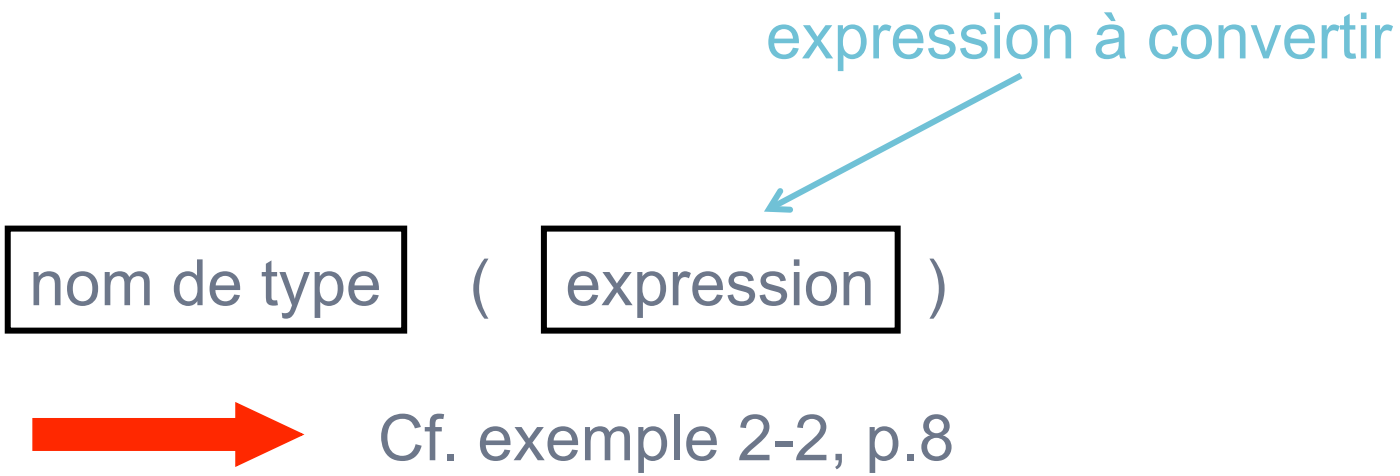
- **Conclusion**: On dispose d'une machine à calculer qui permet de faire n'importe quel calcul.

2.4 Conversion de types (transtypage)

- A chaque variable est associée un type.
- Ce type détermine l'ensemble des valeurs possibles que peut prendre la variable ainsi que les opérations possibles sur cette variable.
- A ce stade, on connaît 3 types:

int	entier (*integer*)
float	flottant (*floating number*)
str	chaîne de caractères (*string*)

- Pour convertir un type dans un autre, on utilise l'instruction suivante:



Exemple: conversions de types

```
type(2)       # retourne int
int(4.9)      # retourne 4: tronque la partie décimale
round(4.5)  # a noter l'existence de la fc round.
              # Ici: retourne 5.0
float(6)      # retourne 6.0
str(128)      # retourne la chaine '128'
int('245')    # retourne l'entier 245
float("3.4159") # retourne le flottant 3.4159000000
float("2.0e4")   # retourne le flottant 20000.0
```

2.5 Affectation directe

- L'affectation directe consiste a assigner un certain *nom de variable* à un certaine *valeur littérale* (numérique, textuelle ou booléenne).
- La syntaxe de cette instruction d'affectation est la suivante:

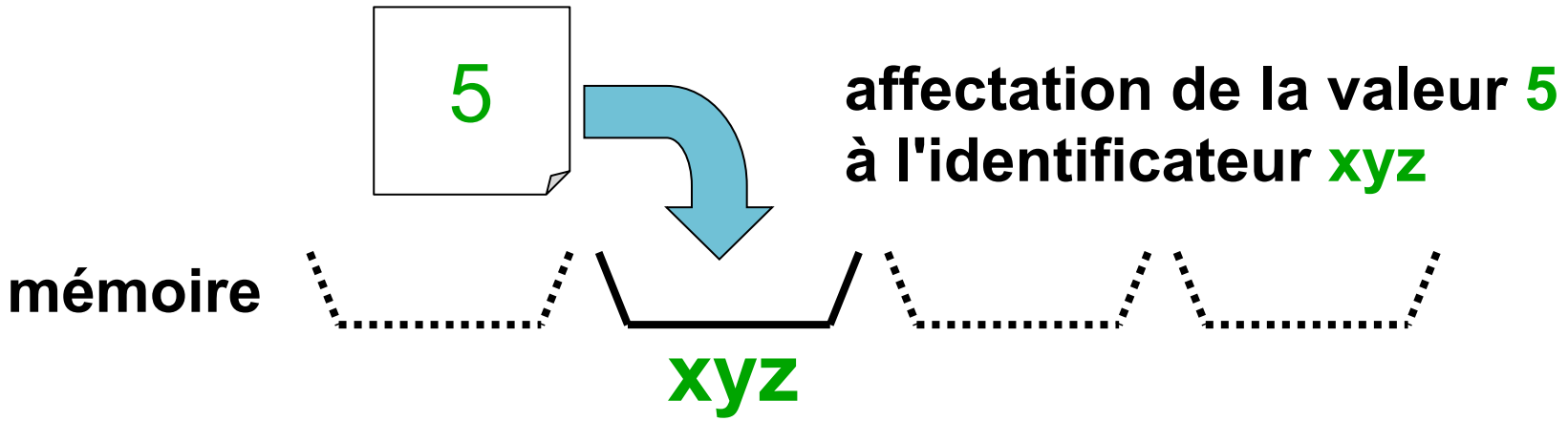
<variable> = <littéral>

Le signe = n'a aucune signification arithmétique!

Le sens de l'affectation: toujours de droite à gauche

Exemple:

```
>>> xyz = 5
```



2.6 Affectation simple

- L'affectation simple consiste a assigner un certain *nom de variable* à un certaine *expression*.
- La syntaxe de cette instruction d'affectation est la suivante:

<variable> = <expression>

Le signe = n'a aucune signification arithmétique!

Le sens de l'affectation: toujours de droite à gauche

- L'**expression** à laquelle on affecte un nom de variable peut être une **opération** (avec des opérateurs et des opérands) ou une **relation** (avec des relateurs et des opérands).

Exemple:

```
>>> xyz = 5
>>> m = xyz ** 2
```

expression

2.7 Opérateur sur les chaînes de caractères

- L'opérateur binaire « + » représente la concaténation.
- On a la syntaxe suivante:

  <chaîne> + <chaîne>

Le signe + n'a aucune signification arithmétique!

La concaténation ne peut s'effectuer qu'entre chaînes

Exemple: (faire démo)

```
>>> date = "15 septembre 2012"
>>> prix_diesel = 1.95  # francs/l
>>> message = date+": le prix du diesel vaut "+ str(prix_diesel)+" Sfr/l"
>>> message
'15 septembre 2012: le prix du diesel vaut 1.95 Sfr/l'
>>>
```