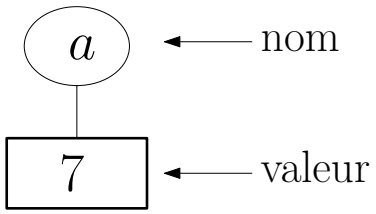
Variables

I Affectation des variables

Pour stocker et manipuler les informations élémentaires on utilise des **variables** qui associent à un nom une valeur : une variable est une boîte qui porte un nom.

Pour donner une valeur à une variable on utilise une instruction fondamentale : **l'affectation**. Sa syntaxe utilise le symbole = sous la forme :

nom_variable = expression




```
>>> a = 7
>>> a
7
```
1

Attention, ici l'utilisation du symbole d'égalité n'a rien à voir à celle des mathématiques où il énonce une relation [1].

En Python, le symbole = sert à décrire **une action** [2] : le membre de droite est évalué et la valeur obtenue est affectée à la variable dont le nom constitue le membre de gauche. Si elle n'existe pas déjà, celle-ci est alors créée. Sinon son ancienne valeur est écrasée par la nouvelle.

En conséquence, il ne se comporte pas de manière symétrique :

```
>>> 7 = a
SyntaxError: can't assign to literal
```
2

• Le **nom de la variable** peut comporter plus d'une lettre mais il faut respecter quelques règles :

— On peut utiliser les lettres minuscules ou majuscules qui sont différentiées ainsi que les 10 chiffres (0 → 9). Les autres caractères sont interdits sauf le tiret bas _ (le tiret usuel - est interdit car c'est le symbole de la soustraction).

— Le premier caractère est obligatoirement une lettre.

— Les mots réservés du language sont interdits :

```
  >>> else = 5
SyntaxError: invalid syntax
```
3

1. Par exemple : $\int_a^b f(t)\mathrm{d}t + \int_b^c f(t)\mathrm{d}t = \int_a^c f(t)\mathrm{d}t$.
2. En pseudo-code on écrira par exemple : a ← 5.

— Utiliser un nom de fonction ne provoquera pas d'erreur directe mais c'est absolument à éviter :

```
>>> print('Hello Word')
Hello Word
>>> print = 1789
>>> print('Hello word')
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    print('Hello word')
TypeError: 'int' object is not callable
```

4

Bonne pratique : en fonction des circonstances dans le code il est recommandé d'utiliser des noms de variables indiquant leurs rôles.

Définition (Expression)

Une **expression** est une combinaison de valeurs explicites (appelées littéraux), de variables, d'opérateurs et de fonctions qui est évaluée pour donner lieu à une valeur.

Lors de l'exécution d'une instruction d'affectation, il y a d'abord évaluation de l'expression puis la valeur obtenue est attribuée à la variable.

```
>>> a = 5
>>> a
5
>>> b = a*(a + 1)
>>> b
30
>>> a = max(1, a, 50)
>>> a
50
```

5

Exercice 1

Déterminer si les suites de symboles sont des expressions ou des instructions.

* x + 1/(x∗x +1)
* x = y + 1
* x < y + 1
* Flag = y >= x + 1

Entraînement 1

Déterminer si les suites de symboles sont des expressions ou des instructions.

* x == y + 1
* x = y + 1
* x <= y + 1
* estPremier = nbDiviseur(n) == 2

Des erreurs classiques :

```
>>> a - 1 = 5
SyntaxError: can't assign to operator
```

6

```
>>> u = v
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    u = v
NameError: name 'v' is not defined
```

7

L'incrémentation :

```
>>> a = 1789
>>> a
1789
>>> a = a + 1
>>> a
1790
```

8

Exercice 2

Quelle instruction provoquera la décrémentation de la variable a ? Son doublement ?

Python permet des raccourcis d'écriture :

```
>>> a = 10
>>> a += 4
>>> a
14
>>> a -= 5
>>> a
9
>>> a *= 2
>>> a
18
```

```
>>> a /= 3
>>> a
6.0
>>> a = 18
>>> a //=3
>>> a
6
>>> a %=5
>>> a
1
```

9

On a aussi :

```
>>> x = y = 10
>>> (x, y)
(10, 10)
>>> (x, y) = (1, 2)
```

```
>>> x
1
>>> y
2
```

10

Essayons un premier code pour permuter les valeurs de deux variables :

```
>>> x = 1
>>> y = 100
>>> y = x
>>> x = y
>>> x
1
>>> y
1
```

11

Expliquer :

La bonne manière de procéder est d'utiliser une variable auxiliaire :

```
>>> x = 1
>>> y = 100
>>> aux = x
>>> x = y
>>> y = aux
>>> x
100
>>> y
1
>>> (x, y)      # Pour afficher les valeurs de plusieurs variables
(100, 1)
```

12

Remarque : on peut aussi utiliser les couples (tuples à deux éléments) avec (x, y) = (y, x) mais c'est considéré comme une solution trop spécifique de Python.

Entraînement 2

En utilisant une seule variable auxiliaire (`aux`) et sans utiliser de tuples, effectuer la permutation circulaire des variables des `x`, `y`, `z` selon le schéma : `x` ⟶ `y` ⟶ `z` ⟲

Exercice 3

Prévoir le résultat des instructions :

```
>>> a = 10
>>> b = 7
>>> c = a + b
>>> a = 3
>>> c = a - b
>>> (a, b, c)
```

13

```
>>> a = 1
>>> b = a
>>> b = 10
>>> c = a
>>> a = b
>>> (a, b, c)
```

14

Définition (État courant d'un programme)

Lors de l'exécution d'un programme, l'ensemble des variables à un instant donné s'appelle l'état d'exécution ou l'état courant du programme.

Suivre l'évolution de l'état courant d'un programme permet souvent de le comprendre en profondeur et de le mettre au point. On peut utiliser un tableau comme ci-contre.

ligne	état courant après		
	a	b	c
1			

Le site pythontutor vous permet de visualiser l'état courant de vos programmes.

`http://www.pythontutor.com/`

Entraînement 3 

On suppose que les variables `x` et `y` contiennent respectivement des valeurs entières a et b. À l'aide d'un tableau d'état, déterminer ce que fait la suite des instructions suivantes :

```
>>> x = x + y
>>> y = x - y
>>> x = x - y
```

Exercice 4

On suppose que, dans l'état courant, les variables x, y, et z ont respectivement pour valeur 2, -1 et 5. Évaluer les expressions suivantes (indépendamment les unes des autres) :

(a) x + 1

(b) y ** 2 == x - 1

(c) x <= z

(d) x * y + 1

(e) x ** y

(f) x+2 * y

II Type

Définition (Type)

Chaque variable possède un type qui est déterminé au moment de l'affectation mais il peut changer (on parle de typage dynamique). Le type indique la nature des données qu'une variable contient.

Les types Python que nous utiliserons le plus souvent sont :

type Python	"traduction"	exemple
`int`	entier	1871
`float`	flottant (décimal)	3.1415
`bool`	booléen (`True` ou `False`)	`True`
`tuple`	n-uplet	`(2, 0, 1, 7)`
`str`	chaîne de caractères (string)	`'Python rocks'`
`list`	liste	`['toto', 5, 2.71, False]`
`function`	fonction	`max`
`complex`	nombre complexe	`1-3j` (i est écrit j)

On obtient le type d'une variable par la fonction **type** :

```
>>> x = 101
>>> type(x)
<class 'int'>
>>> x = 101.
>>> type(x)
<class 'float'>
>>> y = 101
>>> type(x == y)
<class 'bool'>
>>> x == y
True
>>> type(False)
<class 'bool'>
```

```
>>> a = (1, 3.14159)
>>> type(a)
<class 'tuple'>
>>> ch = 'Hello word'
>>> type(ch)
<class 'str'>
>>> L = [1, 'abcdefgh', 2.71]
>>> type(L)
<class 'list'>
>>> def carre(x):
        return x*x

>>> type(carre)
<class 'function'>
```

```
>>> z = 1 + 2*j
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    z = 1 + 2*j
NameError: name 'j' is not defined
>>> type(1+2j)
<class 'complex'>
```

Remarque : Python est un langage orienté objet. Tout y est objet. Les objets même les plus simples ont des classes ce qui explique l'apparition du terme **class** là où attendrait plutôt **type**. Le terme de type est plus général.

Noter l'utilisation du caractère ' pour délimiter les chaînes de caractères.

Exercice 5 Prévoir le type des expressions suivantes :

(a) (x, y) == (1, b)
(b) 'u == 3'
(c) 1+2.
(d) 3.14
(e) 3,14
(f) [3,14]

Quelques opérateurs usuels

On notera que certains opérateurs peuvent s'appliquer à des objets de types différents ; ils sont polymorphes.

Opérateur	Type	Action
+	int, float	Addition
-	int, float	Soustraction
*	int, float	Multiplication
**	int, float	Puissance
/	float	Division décimale
//	int	Quotient de la division euclidienne
%	int	Reste de la division euclidienne
+	str	Concaténation de deux chaines de caractères ou listes
not	bool	Négation (False ↔ True)
and	bool	Et logique
or	bool	Ou logique

Les règles de priorités usuelles s'appliquent. Si on a le moindre doute, le parenthèsage est de rigueur.

On rappelle que les fonctions mathématiques usuelles sont utilisables à condition de les importer :

```
>>> from math import tan, pi
>>> tan(pi/4)
0.9999999999999999
```
15

Pour **convertir** d'un type vers un autre on utilise une fonction dont le nom est le type vers lequel on veut convertir :

```
>>> str(1789)
'1789'
>>> int('1789')
1789
>>> float('3.14159')
3.14159
>>> bool(0)
False
```

```
>>> bool(5)
True
>>> list('2017')
['2', '0', '1', '7']
>>> tuple('(1, 5)')
('(', '1', ',', '5', ')')
# Attention !
# pas le tuple (1,5)
```
16

II.1 Les booléens

Ils ne peuvent avoir que deux valeurs : True ou False.

Les booléens sont souvent obtenus à l'aide des opérateurs de comparaison.

Opérateur	test effectué
==	égalité
!=	différent
<=	inférieur ou égal
>=	supérieur ou égal
<	inférieur strict
>	supérieur strict

Il est à noter que ces opérateurs sont extrêmement polymorphes ; Python évalue en booléen parfois là où l'on attendrait une erreur, ce qui peut rendre cette erreur difficile à détecter. Attention à bien faire la différence entre = et == :

x = y est une instruction qui est exécutée et x == y est une expression qui est évaluée.

On peut chainer les comparaisons :

```
>>> 4>2<8
True
>>> 1 < 5 < 2
False
```
17

Les opérateurs logiques permettent de construire des expressions complexes :

```
>>> a = 5
>>> not(1 < a < 4) or ( a < 0)
True
```
18

Exercice 6 Quelle est la valeur des expressions booléennes suivantes lorsqu'elles sont correctes ?

(a) 3*1.5 > 4

(b) 2.* 8 == 16

(c) 3 - 1 => 3

(d) 0 < 10**5 = 100000

(e) **not**(True **or** False)

(f) 'tric' + 'trac' == 'TricTrac'

Exercice 7 Écrire des expressions booléennes pour traduire les conditions suivantes :

1. Le point de coordonnées (x, y) est dans le cercle de centre (u, v) et de rayon r.
2. Les points (x, y) et (z, t) sont situés sur une même droite parallèle à l'un des axes du repère.
3. L'entier n est divisible par 6.
4. Les entiers a, b, et c sont deux à deux distincts.

Entraînement 4

Écrire des expressions booléennes pour traduire les conditions suivantes :

1. Les intervalles d'entiers $[\![a, b]\!]$ et $[\![c, d]\!]$ ne se rencontrent pas. (On suppose a<b et c<d).
2. Les entiers (supposés non nuls), m et n sont tels que l'un est le multiple de l'autre.
3. Le point de coordonnées (x, y) est dans le rectangle dont deux sommets opposés sont de coordonnées (a, b) et (c, d).

II.2 Les n-uplets

Les n-uplets (*tuple* en anglais) généralisent la notion de couple ou de triplet. Il corespondent au produit cartésien d'ensemble en math. Pour définir un tuple en Python, on utilise les parenthèses et la virgule.

```
>>> t = (2 , 3 , 5 , 7)
>>> t + (11, 13)  # Concaténation
(2, 3, 5, 7, 11, 13)
>>> t[0]   # Accès aux composantes
2
>>> t[6]
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    t[6]
IndexError: tuple index out of range
>>> 3 in t   # Test d'appartenance
True
>>> len(t)    #Longueur
4
>>> u = ('MPSI', 2.4, 1 == 0.5 * 2, (1, 2))
>>> u[2]
True
>>> u[3]    #  un tuple peut être dans un tuple
(1, 2)
>>> type ((1))
<class 'int'>
>>> type ((1,))  # pour avoir un tuple avec un élément
<class 'tuple'>
>>> type(())   # Il existe un tuple vide
<class 'tuple'>
```

19

Les tuples ne sont pas modifiables, on dit aussi qu'ils sont immuables.

```
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    u[0] = 'PCSI'
TypeError: 'tuple' object does not support item assignment
```

20

Pour déconstruire un tuple, on utilise une affectation :

```
>>> (x, y) = A
>>> x
1
>>> y
5.5
>>> (_, v) = A  # Si on ne veut qu'une composante
>>> v
5.5
```

21