

Variables et expressions

Alexandre Benoit

BCPST

| Types simples

- Précision arbitraire ;

- Précision arbitraire ;
- 6 opérateurs :

- Précision arbitraire ;
- 6 opérateurs :
 - L'addition ;

- Précision arbitraire ;
- 6 opérateurs :
 - L'addition ;
 - La soustraction ;

- Précision arbitraire ;
- 6 opérateurs :
 - L'addition ;
 - La soustraction ;
 - La multiplication : * ;

- Précision arbitraire ;
- 6 opérateurs :
 - L'addition ;
 - La soustraction ;
 - La multiplication : * ;
 - Le quotient (//) et le reste (%) d'une division euclidienne (exemple :
17//5
donne 3 et
17% 5
donne 2) ;

- Précision arbitraire ;
- 6 opérateurs :
 - L'addition ;
 - La soustraction ;
 - La multiplication : * ;
 - Le quotient (//) et le reste (%) d'une division euclidienne (exemple :
17//5
donne 3 et
17% 5
donne 2) ;
 - L'exponentiation (exemple
2**3
donne 8).

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);
- Nombre du type :
3.2 ou 3.0;

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);
- Nombre du type :
3.2 ou 3.0;
- Les opérateurs +, -, *, ** sont également définis avec quelques différences : exemple
2.0**100.0
et
2** 100 ;

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);
- Nombre du type :
3.2 ou 3.0;
- Les opérateurs `+`, `-`, `*`, `**` sont également définis avec quelques différences : exemple
2.0**100.0
et
2** 100 ;
- Opérateur de division entre flottants : `/` (différent de l'opérateur sur les entiers).

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);
- Nombre du type :
3.2 ou 3.0;
- Les opérateurs +, -, *, ** sont également définis avec quelques différences : exemple
2.0**100.0
et
2** 100 ;
- Opérateur de division entre flottants : / (différent de l'opérateur sur les entiers).
- Changement possible d'un entier en flottant : float(7) donne 7.0.

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);
- Nombre du type :
3.2 ou 3.0;
- Les opérateurs `+`, `-`, `*`, `**` sont également définis avec quelques différences : exemple
2.0**100.0
et
2** 100 ;
- Opérateur de division entre flottants : `/` (différent de l'opérateur sur les entiers).
- Changement possible d'un entier en flottant : `float(7)` donne 7.0.
- ou d'un flottant en entier `int(3.9)` donne 3 et `int(-3.9)` donne -3

Les nombres à virgules flottants

- Écriture scientifique d'un nombre à virgule (en informatique, on met des .);
- Nombre du type :
3.2 ou 3.0;
- Les opérateurs `+`, `-`, `*`, `**` sont également définis avec quelques différences : exemple
2.0**100.0
et
2** 100 ;
- Opérateur de division entre flottants : `/` (différent de l'opérateur sur les entiers).
- Changement possible d'un entier en flottant : `float(7)` donne 7.0.
- ou d'un flottant en entier `int(3.9)` donne 3 et `int(-3.9)` donne -3
- On peut aussi avoir des conversions automatiques : `3*2.0` donne 6.0

- Seulement 2 booléens : True et False

- Seulement 2 booléens : True et False
- 3 Opérateurs :

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;
 - La disjonction : **or**.

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;
 - La disjonction : **or**.
- Opérateurs paresseux. Exemple
False and 1/0;

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;
 - La disjonction : **or**.
- Opérateurs paresseux. Exemple
False and 1/0;
- Opérateurs de comparaisons

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;
 - La disjonction : **or**.
- Opérateurs paresseux. Exemple
False and 1/0;
- Opérateurs de comparaisons
 - Test d'égalité : **==** ;

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;
 - La disjonction : **or**.
- Opérateurs paresseux. Exemple
False and 1/0;
- Opérateurs de comparaisons
 - Test d'égalité : **==** ;
 - Test de non égalité : **!=** ;

- Seulement 2 booléens : True et False
- 3 Opérateurs :
 - La négation : **not** ;
 - La conjonction : **and** ;
 - La disjonction : **or**.
- Opérateurs paresseux. Exemple
False and 1/0;
- Opérateurs de comparaisons
 - Test d'égalité : `==` ;
 - Test de non égalité : `!=` ;
 - Test d'inégalité : `<` ; `>` ; `<=` ; `>=`.

II Variables

Dans un programme, une **variable** sert à désigner une zone mémoire de l'ordinateur. On peut y stocker une valeur, accéder à cette dernière et la changer.

Dans un programme, une **variable** sert à désigner une zone mémoire de l'ordinateur. On peut y stocker une valeur, accéder à cette dernière et la changer.

Pour faire référence à une zone mémoire on utilise un **nom de variable**.

- En Python, on déclare une variable avec la commande
`nom_de_variable = expression`

- En Python, on déclare une variable avec la commande
`nom_de_variable = expression`
- Par exemple ; on déclare x comme
`x=2`

- En Python, on déclare une variable avec la commande
`nom_de_variable = expression`
- Par exemple ; on déclare `x` comme
`x=2`
- On peut afficher ou effectuer des calculs avec ces variables :
`x`
`x+1`

- Pour changer la valeur d'une variable, on utilise la même instruction que pour la déclaration.

```
x=3
```

```
x
```

```
x=x+1
```

```
x
```

- Pour changer la valeur d'une variable, on utilise la même instruction que pour la déclaration.

```
x=3
```

```
x
```

```
x=x+1
```

```
x
```

- Il existe aussi une expression particulière, **input()**, qui attend qu'on tape quelque chose au clavier :

```
a=input()
```

- Pour changer la valeur d'une variable, on utilise la même instruction que pour la déclaration.

```
x=3
```

```
x
```

```
x=x+1
```

```
x
```

- Il existe aussi une expression particulière, **input()**, qui attend qu'on tape quelque chose au clavier :

```
a=input()
```

- Si on tape 3.5, la sortie est '3.5' : Ce qui n'est ni un entier ni un flottant.

- Pour changer la valeur d'une variable, on utilise la même instruction que pour la déclaration.

```
x=3
```

```
x
```

```
x=x+1
```

```
x
```

- Il existe aussi une expression particulière, **input()**, qui attend qu'on tape quelque chose au clavier :

```
a=input()
```

- Si on tape 3.5, la sortie est '3.5' : Ce qui n'est ni un entier ni un flottant.

- Pour obtenir un entier, on tape :

```
a=int(input())
```

Pour un flottant, on tape :

```
a=float(input())
```

III Types composés

Un n -uplet, *tuple* en anglais, est une généralisation du concept de couple ou de triplet.

- Voici un couple composé d'un entier et d'un flottant :
(1, 2.2)

Un n -uplet, *tuple* en anglais, est une généralisation du concept de couple ou de triplet.

- Voici un couple composé d'un entier et d'un flottant :

(1, 2.2)

- On peut stocker un n -uplet dans une variable

t=(1, 2.2, 3, 4)

Un n -uplet, *tuple* en anglais, est une généralisation du concept de couple ou de triplet.

- Voici un couple composé d'un entier et d'un flottant :

(1, 2.2)

- On peut stocker un n -uplet dans une variable

`t=(1, 2.2, 3, 4)`

`t[0]` permet d'évaluer la première composante.

Un n -uplet, *tuple* en anglais, est une généralisation du concept de couple ou de triplet.

- Voici un couple composé d'un entier et d'un flottant :

$(1, 2.2)$

- On peut stocker un n -uplet dans une variable

$t = (1, 2.2, 3, 4)$

$t[0]$ permet d'évaluer la première composante.

- On peut aussi affecter simultanément différentes variables à partir d'un n -uplet :

$x, y = (1, 2)$

- Il est possible de coller un n -uplet et un p -uplet pour obtenir un $(n + p)$ -uplet :
 $(1+2)+(3,4,5)$

Opération sur les n -uplets

- Il est possible de coller un n -uplet et un p -uplet pour obtenir un $(n + p)$ -uplet :
 $(1+2)+(3,4,5)$
- Il est possible de tester si une valeur appartient à un n -uplet :
 $3 \text{ in } (1,2,3)$

Opération sur les n -uplets

- Il est possible de coller un n -uplet et un p -uplet pour obtenir un $(n + p)$ -uplet :
 $(1+2)+(3,4,5)$
- Il est possible de tester si une valeur appartient à un n -uplet :
 $3 \text{ in } (1,2,3)$
- Il est possible de connaître la longueur d'un n -uplet :
 $\text{len}((1,2,3))$

Opération sur les n -uplets

- Il est possible de coller un n -uplet et un p -uplet pour obtenir un $(n + p)$ -uplet :
 $(1+2)+(3,4,5)$
- Il est possible de tester si une valeur appartient à un n -uplet :
 $3 \text{ in } (1,2,3)$
- Il est possible de connaître la longueur d'un n -uplet :
 $\text{len}((1,2,3))$
- Il n'est pas possible de modifier un élément directement :
 $t = (1,2)$
 $t[0]=2$

- Une chaîne caractères est une suite finie de caractères consécutifs, qu'on note entre apostrophe ou guillemets :
'Ceci est une phrase'
"Ceci est une phrase"

- Une chaîne caractères est une suite finie de caractères consécutifs, qu'on note entre apostrophe ou guillemets :
`'Ceci est une phrase'`
`"Ceci est une phrase"`
- Comme pour les n -uplets, on peut accéder directement à un caractère
`s = 'bonjour'`
`s[2]`

- Une chaîne caractères est une suite finie de caractères consécutifs, qu'on note entre apostrophe ou guillemets :
`'Ceci est une phrase'`
`"Ceci est une phrase"`
- Comme pour les n -uplets, on peut accéder directement à un caractère
`s = 'bonjour'`
`s[2]`
- On peut aussi concaténer, calculer la longueur, ou effectuer un test d'appartenance.

- On peut convertir une valeur d'un type simple vers une chaîne de caractère :

```
str(1.2)
```

- On peut convertir une valeur d'un type simple vers une chaîne de caractère :

```
str(1.2)
```

- Ou effectuer l'opération inverse

```
int('123')
```

```
float('1.2')
```

```
bool('true')
```

Une liste est un n -uplet dont on peut changer la valeur des composantes.

- On remplace les parenthèses par des crochets :

[1,2,3]

Une liste est un n -uplet dont on peut changer la valeur des composantes.

- On remplace les parenthèses par des crochets :

[1,2,3]

- Toutes les opérations vues pour les n -uplets sont définies (Concaténation, longueur, test d'appartenance)

Une liste est un n -uplet dont on peut changer la valeur des composantes.

- On remplace les parenthèses par des crochets :

[1,2,3]

- Toutes les opérations vues pour les n -uplets sont définies (Concaténation, longueur, test d'appartenance)

- On peut changer directement la valeur d'un élément :

L = [1,2,3]

L[1]=4

Une liste est un n -uplet dont on peut changer la valeur des composantes.

- On remplace les parenthèses par des crochets :

```
[1,2,3]
```

- Toutes les opérations vues pour les n -uplets sont définies (Concaténation, longueur, test d'appartenance)

- On peut changer directement la valeur d'un élément :

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

```
L[1]=4
```

- On peut convertir les n -uplets en listes :

```
list( (1,2,3) )
```

```
tuple( [4,5,6] )
```

```
list('bonjour')
```

Un dictionnaire est un objet pouvant en contenir d'autres. Un dictionnaire associe chaque objet à une clé.

Un dictionnaire est un objet pouvant en contenir d'autres. Un dictionnaire associe chaque objet à une clé.

- On déclare le dictionnaire comme :

```
Paul = { 'age' : 18, 'Lycee' : 'Hoche',  
        'classe' : 'BCPST1A' }
```

Un dictionnaire est un objet pouvant en contenir d'autres. Un dictionnaire associe chaque objet à une clé.

- On déclare le dictionnaire comme :

```
Paul = { 'age' : 18, 'Lycee' : 'Hoche',  
        'classe' : 'BCPST1A' }
```

- Pour avoir l'âge de Paul, on écrit :

```
Paul[ 'age' ]
```

Un dictionnaire est un objet pouvant en contenir d'autres. Un dictionnaire associe chaque objet à une clé.

- On déclare le dictionnaire comme :

```
Paul = { 'age' : 18, 'Lycee' : 'Hoche',  
        'classe' : 'BCPST1A' }
```

- Pour avoir l'âge de Paul, on écrit :

```
Paul [ 'age ' ]
```

- Pour ajouter sa moyenne en maths, on écrit :

```
Paul [ 'maths ' ] = 14
```

- ce qu'est une variable, la déclarer et la modifier en Python

- ce qu'est une variable, la déclarer et la modifier en Python
- Connaître et reconnaître des types de variables (int, float, string, bool, etc)

- ce qu'est une variable, la déclarer et la modifier en Python
- Connaître et reconnaître des types de variables (int, float, string, bool, etc)
- Connaître les opérateurs (+, **not**, != etc)

- ce qu'est une variable, la déclarer et la modifier en Python
- Connaître et reconnaître des types de variables (int, float, string, bool, etc)
- Connaître les opérateurs (+, **not**, != etc)
- Faire les 6 premiers exercices du TD.

Ce cours est issu en grande partie du livre *Informatique pour tous* de Wack et autres aux éditions **Eyrolles**