I11: Chapitre 1
Les bases du langage

Nicolas Méloni

Licence 1: 1er semestre
(2018/2019)

Données informatiques

- Les données sont stockés dans des **cases mémoires**
- Elles sont accéssibles via des **adresses**

Types de données standards

Python permet de manipuler toutes sortes de données :

- Entiers : 0, -12, 3569
- Flottants : 0.232, 3.1415, -12.0
- Booléens : `True, False` (vrai, faux)
- Chaînes de caractères : "Bonjour le monde"

On parle en programmation du **type** d'une donée. On peut connaitre le type d'une donnée en utilisant la fonction `type`. Pour afficher une donnée on utilise la fonction `print`.

```
>>> type(12)
 <class 'int'>
>>> print('Bonjour a tous')
 Bonjour a tous
```

Les entiers/flottants

Le type `int`

- Permet de représenter n'importe quel entier positif ou négatif
- On utilise l'écriture en base du 10 du nombre
- Il est possible d'utiliser les représentation binaire, octale ou hexadécimale

Exemples

```
>>> type(-1254)
 <class 'int'>
>>> 0b110 #ecriture binaire
 6
>>> 0o12 #ecriture octale
 10
>>> 0x5A #ecriture hexadecimale
 90
```

Le type `float`

- Permet de représenter une partie des nombres à virgule
- La sépartion entre partie entière et partie décimale se fait avec un point .
- On peut également utiliser l'écriture scientifique : 2e4 représente 2×10^4

Exemples

```
>>> type(3.14)
 <class 'float'>
>>> .95
 0.95
>>> 12.
 12.0
>>> 3.14e1 #ecriture scientifique
 31.4
```

Opérations sur les entiers/flottants

Opérations arithmétiques

- `-` : moins unaire
- `+,-,*,/,()` : opérations usuelles
- `//` : division entière
- `%` : reste de la division (modulo)
- `**`: puissance

Opérations sur les entiers/flottants

Exemples

```
>>> --3
 3
>>> (7-4)/2
 1.5
>>> 24//5
 4
>>> 20.5//2.5
 8.0
>>> 15 % 9
 6
>>> 7 % 2.4
 2.2
>>> 10**3
 1000
>>> 9**(1/2)
 3.0
```

Opérations sur les entiers/flottants

Opérations de comparaisons

- `==` : égalité
- `<`, `>`, `!=` : inégalités strictes
- `<=`, `>=` : inégaliés larges

Exemples

```
>>> 10*2-5 > 12
 True
>>> 20/2 < 10
 False
>>> 3.0 == 3
 True
>>> 5/2*3 != 5/(2*3)
 True
```

Les booléens

Le type `bool`

- Permet de représenter les valeurs logiques `VRAI` et `FAUX`
- Deux mots réservés en python : `True` et `False`
- N'importe qu'elle donnée en Python possède une valeur logique :
  - les valeurs non nulles valent `True`
  - les valeurs vides ou nulles valent `False`

Exemples

```
>>> type(True)
 <class 'bool'>
>>> type(False)
 <class 'bool'>
```

Opérations sur les booléens

Opérateurs booléens

- `not` : négation
- `and` : et
- `or` : ou inclusif

Exemples

```
>>> not True
 False
>>> True and False
 False
>>> True or False
 True
>>> (5>3) and (2.0==2)
 True
```

Priorités des opérateurs

Opérateurs rangés par priorité descendante

- ()
- - (unaire)
- ** !! associatif à droite !!
- *,/,//,%
- +,-
- <,>,<=,>=,==,!=
- not
- and
- or

Exemple

```
>>> not 16/2**2+10 != 15 % 8 * 2 and 0.5**-2/2*5 >= 10**2**(1/2)
 False
```

Opérations sur les chaînes de caractères

Le type `str`

- Permet de représenter les textes
- Toute suite de caractères entourée par des apostrophes (’), des guillemets (") ou bien des triples quotes (’’’ ou """)

Exemple

```
>>> type('mot')
 <class 'str'>
>>> 'Une chaine de caracteres'
 'Une chaine de caracteres'
>>> "Une autre"
 'Une autre'
>>> "L'exemple pertinant"
 'L\'exemple pertinant'
>>> '''Un exemple avec 'apostrophes' et "guillemets"'''
 'Un exemple avec \'apostrophes\' et "guillemets"'
```

Opérations sur les chaînes de caractères

Caractères spéciaux

- \ : le caractère d'échappement (transforme le caractère suivant)
- \n : saut de ligne
- \t : tabulation
- \b : retour arrière

Exemple

```
>>> print('Bonjour\na\ntous')
Bonjour
a
tous
>>> print('Bonjour\ta\ttous')
Bonjour	a	tous
>>> print('Bonjour\ba\btous')
Bonjoutous
```

Opérations sur les chaînes de caractères

Les opérateurs

- `+` : concaténation
- `*` : répétition
- `len` : fonction retournant la longueur d'une chaîne

Exemples

```
>>> "Bonjour" + "a" + "tous"
 'Bonjouratous'
>>> "cou"*2
 'coucou'
>>> len("cou"*2+' toi')
 10
```

Transtypage

Il est possible de transformer le type d'une donnée :

- Transformer en entier : fonction `int()`
- Transformer en floatant : fonction `float()`
- Transformer en entier : fonction `str()`

Cela ne fonctionne que si la donnée de départ est compatible avec le type d'arrivée.

Transtypage

Exemples

```
>>> int(2.3)
 2
>>> int("21")
 21
>>> int("2.3")
 ValueError: invalid literal for int() with base 10: '2.3'
>>> float(2)
 2.0
>>> float("2.3")
 2.3
>>> float("2,3")
 ValueError: could not convert string to float: '2,3'
>>> str(3.14)
 '3.14'
```

Variables et affectation

Variables

- Pour stocker une donnée en mémoire on utilise une **variable**
  - Le nom d'une variable est une suite de caractères ne contenant que des caractères alphanumériques ($0 \rightarrow 9$, a$\rightarrow$z , A$\rightarrow$A) ou le caractère sousligné (_)
  - Le nom d'une variable commence forcément par une lettre ou un caractère sousligné
  - Le nom d'une varaible ne peut pas être un mot réservé du langage : **if, elif, else, while, not, and, or** ...
  - La casse est prise en compte (majuscule $\neq$ minuscule)
- Stocker une donnée dans une variable se fait à l'aide de l'opérateur =
- Cette opération s'appelle une **affectation**

Variables et affectation

Exemples

```
>>> a = 3
>>> b = 5
>>> print(a,b)
3 5
>>> chaine = 'to'
>>> print(chaine*2)
toto
>>> _un_ = 1
>>> _2_ = 2
print(_un_ + _2_)
3
>>> 3eme = 3
 File "<stdin>", line 1
    3eme = 3
       ^
SyntaxError: invalid syntax
```

Variables et affectation

L'opérateur =

- Il n'est pas symétrique (`3 = a` ne marche pas)
- Il est associatif à droite ! (`a = b = c` veut dire `a = (b = c)`)
- Il peut servir pour des affectation parallèles (`a,b = 2,3`)

Exemples

```
>>> a,b,c = 1,2,3
>>> print(a,b,c)
1 2 3
>>> a = b = c
>>> print(a,b,c)
3 3 3
```

Afficher des données : la fonction `print`

- Affiche la valeur des données passées en paramètres
- S'il y a plusieurs données il faut les séparer par des virgules, elles séparées par un espace à l'affichage
- Interprète les caractères spéciaux (saut de ligne, tabulation, etc)
- Se termine par un saut de ligne ...
- .. qui peut être modifié à l'aide du paramètre optionnel `end`

Exemples

```
>>> print("Bonjour");print("Bonjour")
Bonjour
Bonjour
>>> print("Bonjour",end=" ");print("Bonjour")
Bonjour Bonjour
```

Saidir des données au clavier : la fonction `input`

- Stoppe l'exécution du programme jusqu'à ce que l'utilisateur aie saisi une donnée
- La saisie se termine toujours par l'appuie sur la touche `Entrée`
- Renvoie une donnée de type `str` !
- Un texte peut être ajouté avant l'invite de saisie en paramètre de la fonction

Exemples

```
>>> age = input("Quel est votre age?")
Quel est votre age?102
>>> age
 '102'
>>> print("Vous avez",age,"ans")
Vous avez 102 ans
```