
ij2



J2 : Objectifs

- Visiter le zoo et nous adresser à chaque animal
 - avec Jupyter
 - Une bonne manière de prendre des notes
- Littéraux, opérateurs
- Entiers, Booléens, Flottants, Complexes, Chaînes
- Priorité des opérateurs et parenthèses
- Représentations
- L'affectation
- Fonctions
- Typage
- Opérateurs de comparaison
- Expressions
- Exercices

Entiers

- integer, int, <class 'int'>
- Littéraux : -1, 0, 12847
- Les littéraux sont évalués et affichés (« REPL »)
- Opérateurs : + - * / (opérations sur les nombres entiers)
 - Noter que +- désigne le signe des nombres entiers relatifs (opérateurs unaires) mais aussi les opérations d'addition et de soustraction (opérateurs binaires).
 - // division entière
 - % modulo (reste dans la division entière)
 - ** élévation à la puissance
 - Priorité des opérateurs - essayer 1+2*3
 - Les produits et les divisions sont calculés avant les additions et les soustractions
 - */ sont prioritaires sur +-
 - L'élévation à la puissance est prioritaire sur * et /
 - Quand on rencontre des opérateurs de même poids, c'est l'ordre donné par la lecture de gauche à droite qui s'applique.
 - Pour contrarier ces comportements de base, utiliser des parenthèses
 - Ce qui est entre parenthèses est évalué avant ce qui est autour.
- Ecriture binaire (en base 2), octale (en base 8), hexadécimale (en base 16)
 - 0b1010
 - 0o12
 - 0xA

Classeur Jupyter Entiers.ipynb

Représentation des nombres entiers

- La représentation polynomiale des nombres
- La représentation naturelle des entiers naturels
- L'algorithme de l'addition
- La représentation naturelle des entiers relatifs
- Inconvénients
- La représentation en complément à 2

Classeur Jupyter Entiers.ipynb

Opérations sur les chaînes de bits

- & et binaire
- | ou binaire
- ^ ou exclusif binaire
- ~ non binaire, complémentation
- << décalage à gauche, ajout de 0 à droite
- >> décalage à droite,

Il n'y a pas de rotation à droite/à gauche

Exercice : Que valent ?

`1&2`

`1|2`

`1^2`

`2^2`

`~2`

`2>>1, 2>>2`

`2<<1, 2<<2`

Expliquer !

Classeur Jupyter Bits.ipynb

Booléens

- Boolean, bool, <class 'bool'>
- Littéraux ou constantes : True, False
- Observer que Python3 ne connaît pas true et false : la casse (distinction majuscules/minuscules) est importante
- Opérateurs : not (opérateur unaire) and, or (opérateur binaire)
- Construire les tables de vérité de ces opérateurs
- Priorité des opérateurs : not est prioritaire sur and qui est prioritaire sur or.

Classeur Jupyter Booléens.ipynb

Flottants

- On parle de nombres réels, mais les ordinateurs représentent des nombres rationnels uniquement
 - mantisse exposant
 - Par exemple +3.14 est représenté par + 0.314 E+1
- Nombres flottants, float, <class 'float'>
- C'est le . qui sert de marque décimale (et non la virgule)
- Opérateurs : +, -, *, /, **
- Observer le résultat de $2.3 * 10$

Classeur Jupyter Flottants.ipynb

Représentation des nombres flottants

- Mantisse
 - Signe + ou -
 - Valeur absolue 0.xxxxxx
- Exposant
 - Signe+ -
 - Valeur absolue xxx

Classeur Jupyter Flottants.ipynb

Complexes

- Nombres imaginaires $a+b*j$ où a est la partie réelle et b , la partie imaginaire
 - $j^2 = -1$
- `complex`, `<class 'complex'>`
- Evaluer `3*(2+4j)`

Classeur Jupyter Complexes.ipynb

Chaînes de caractères

- String, <class 'str'>
- 'toto' ou « toto », utile pour "J'ai dit", 'Il a dit "Merci"', ou encore en échappant l'apostrophe 'J\'ai dit'
- Retour à la ligne " Il était \n une fois "
- Tabulation "Prix \t 3,50 Euros"
- Une chaîne sur plusieurs lignes

```
"""Maître corbeau sur un arbre perché
    Tenait en son bec un fromage
    Maître renard par l'odeur alléché
    Lui tint à peu près ce langage
    """
```
- Opérateur : + concaténation * répétition
 - "Salut"+"les amis"
 - "*"*10

Chaînes d'octets

- *bytes*
- *b'kjhkjhkjhkjh'*
- 8 bits
- octet = 1 valeur entre 0 et 255 écrite en hexa, par ex 0xff ou en 0o377

Séquences d'échappement

<code>\n</code>	fin de ligne
<code>\\</code>	backslash
<code>\', \"</code>	quotes 1x et 2x
<code>\n</code>	Line feed
<code>\r</code>	Carriage Return
<code>...</code>	<code>...</code>
<code>\t</code>	Tab
<code>\ooo</code>	octal
<code>\xhh</code>	hexa
<code>\uxxxx</code>	unicode 16 bits – 4 hex
<code>\Uxxxxxxxx</code>	unicode 32 bits – 8 hex
<code>\N{name}</code>	nom du point de code

Unicode

- Pour représenter les caractères, les octets (*bytes*) sont privilégiés
 - Caractères de la machine à écrire (moins de 100) + 32 caractères non imprimables utilisés pour les transmissions (accessibles au clavier avec la touche CTL...)
 - ASCII (America Standard Coding for Information Interchange) et d'autres (passés sous silence)
 - représentés sur 7 bits
 - Compléter l'emploi des 256 positions pour mettre les caractères accentués de chaque langue
 - ISO 8859-1 ou ISO Latin-1 (ajout de 96 symboles supplémentaires)
 - Windows CP1252 (ajout de 27 symboles supplémentaires)
- Les octets sont donc saturés.
 - Mis bout à bout, les alphabets internationaux demandent ensemble beaucoup plus que 255 positions.
 - Solution : invention d'Unicode, capacité de 1,1Millions de *code points*.
 - 10% sont utilisés pour le moment.
 - Unicode se confond avec l'ASCII pour les premières positions et l'étend.

UTF-8

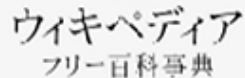
- Pour représenter un point de code par des octets, il faut un codage
 - *encoding*
- UTF-8 est un codage Unicode
 - Pour les points de code <128 , UTF-8 est identique au code ASCII
 - Pour les points de code ≥ 128 , UTF-8 utilise une séquence d'octets (2, 3 ou 4)
 - D'autres codages existent
 - UTF-16, ...
- Avantages
 - Tous les points de code sont représentés
 - Pas de place perdue,
 - Pas de perturbation des transmissions avec 0 dans les chaînes de caractères qui bloqueraient les strcpy()
 - Compatible avec ASCII
 - Resynchronisation efficace après un caractère erroné

UTF-8 et Python

- Python 3 est nativement UTF-8 partout
 - dans le source
 - dans la représentation du type str
- Python2 avait un type distinct pour les chaînes Unicode
 - un type str qui confondait la notion de caractère avec celle d'octet
 - des fonctions de conversions et des exceptions perturbantes pour le programmeur
- En Python 3, str et bytes sont des classes (types) différents

Recommandations

1. Dans votre code privilégier UTF-8
2. Dans les interfaces avec l'extérieur, identifier le codage utilisé (le mettre en doute...) et faire la conversion vers UTF-8 le plus tôt possible (et dans l'autre sens, le plus tard)
3. Dans vos tests, inclure des caractères Unicode exotiques pour éviter le *mojibake*

[illegible][illegible] $\frac{1}{\sqrt{2}}$

Ž

6-263

ç. "é"†

а±¥æ'è¡ç¤°



æ-þǫ-ǫœ-ǣ



ă,â,éă,ăfşăfţă,éă,ăă şăă @æ-ță—ăœ-ăă 'ăă «ăă ăă ,ăă |ăă -ăă Help:ç% 'æ@šæ-ță—
ă,ăă *è|șăă ăă ăă •ăă ,ăă,

[illegible]

- ä¼¼¼šäœ—âœ—âœ 'äœ äœ œäœ äœœÄ!äœ"äœ¡Ä¥Ääœ"Ä¥Ä!äœ"Ä£Ä äœ"äœ äœ "è¡çºäœ •ä,œä,äœ "äœ ©äœ,

[illegible]

āf «āf·ā,āf™āffāf·ā½ç™ēēēāā «āā Šāā „āā āā ēē²ā½āā ™ā,çç™ēēāā Œāāāoeāā —āā āā āā ēāā Ÿāā āā āā ē,‰
 āē āē—Ÿāoeēēēāā ©'Mojibake'āā āā āā ēēēē%

ă Ą Œǎ Ĭ āl @āļ %āō %é€šçªāō ™ă,ă^āō †āl «āl ºāl £āl Yă€,â†#Mojibake

© 2007 Pearson Education, Inc. All rights reserved.

1 ä, »ãl ºãŽŸã,

- 1.1 $\text{èi} \bar{\text{ç}}^{\circ} \text{æ}^{\text{TM}}, \text{ã} \text{ } \text{õã}, \text{ãf}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ã}, \text{ã} \text{ } \text{õæ} \text{ç}^{\circ} \text{ã} \text{õ} \text{ã} \text{ } \text{«é-çã}^{\text{TM}}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{«}$
1.2 $\text{æ} \text{ } \text{è}^{\circ} \text{ã}^{\circ} \text{ãf}^{\circ} \text{ã}, \text{õãf}^{\circ} \text{ãf}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ã} \text{ } \text{õé} \text{ } \text{ã} \text{ } \text{ã} \text{ } \text{«ã}, \text{ã}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{«}$
1.3 $\text{æ-çã}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ã}, \text{ã} \text{ } \text{õã}^{\circ} \text{ã}^{\circ} \text{æ} \text{ } \text{ã} \text{ } \text{«é-çã}^{\text{TM}}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{«}$
1.4 $\text{ãf}^{\circ} \text{ãf}^{\circ} \text{ã}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ } \text{õæ} \text{ } \text{æ} \text{æ} \text{ } \text{è}^{\circ} \text{ã}^{\circ} \text{ã}^{\circ} \text{ã}^{\circ} \text{ã}^{\circ} \text{ } \text{õç}^{\circ} \text{ã} \text{ } \text{ã} \text{ } \text{«ã}, \text{ã}, \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{«}$
1.5 $\text{é} \text{é} \text{ã} \text{ } \text{ç}^{\circ} \text{ã} \text{ } \text{ç}^{\circ} \text{ã} \text{ } \text{õæ} \text{ } \text{ã} \text{ } \text{ã} \text{ } \text{õãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{ãf}^{\circ} \text{«}$

©ăf-ăf«

names

- Identificateurs – pour Python ce sont des noms
 - Commencent par une lettre Unicode ou par un _
 - Continuent par des lettres, des _ ou des chiffres
- Utilisés à chaque fois qu'il nous faut baptiser
 - une variable, une fonction, une constante
- Eviter les collisions avec les mots-réservés du langage Python3
- Signalées par les outils
- Pas de confusion possible avec les littéraux

Variable

- Exemple: x
- Constitue la *référence* d'un objet qui stocke la valeur de x
- Python3 distingue les objets modifiables (*mutable*) et non modifiables (*immutable*). En Français, on peut aussi employer : muable et immuable.
- Dans le cas des objets non modifiables, dire que la variable x référence *la valeur* de x est acceptable.
- Dans le cas d'objets modifiables, la variable peut faire référence à *un objet* (toujours le même) mais la valeur de cet objet a pu changer.

Variable `_`

- dans l'interpréteur, `_` désigne la valeur de la dernière expression évaluée
- dans le code, `_` désigne une variable que l'on souhaite ignorer ou une variable que l'on ne se donne pas la peine de baptiser

Affectation

- Le signe égal ne signifie pas l'égalité mais se lit plutôt comme "prend la valeur de" ou bien "reçoit" (*assignment*)
 - Dans certains langages, on l'écrit comme une flèche ←
- L'expression à droite du signe = est évaluée et le résultat est conservé dans un objet en mémoire
- L'expression à gauche du signe égal est, la plupart du temps, le nom qui sert d'étiquette pour l'objet
- L'affectation crée un lien entre un nom et un objet qui a une valeur

```
x = 1
s = 3.14 * 12**2
```

Exercice : Echange (Swap)

- On donne les valeurs de a et de b.
- Quelles instructions Python écrire
 - et dans quel ordre –pour échanger les valeurs de a et b ?
- Par exemple, si on a donné...
- ... après l'échange, on doit avoir
 - Cela doit fonctionner pour toutes les valeurs de a et de b
 - (et pas seulement pour cet exemple!)

```
a = ...  
b = ...
```

```
a = 25  
b = 12
```

```
a == 12  
b == 25
```

Affectation augmentée

- *Augmented assignment*
- x prend l'ancienne valeur de x à laquelle on ajoute 1
- Autres exemples avec des opérateurs binaires

```
x += 1
```

```
x = x + 1
```

```
x -= 2  
x *= 2  
x += " Bla"
```



Affectation chaînée

- Affectation chaînée
chained assignment

```
x = y = 0
```

- Fait la même chose que :

```
temp = 0  
x = temp  
y = temp
```

Affectation parallèle

```
x, y, z = 0.157 , True, "A"
```

- Fait la même (?) chose que :
 - presque

```
x= 0.157  
y = True  
z=  "A"
```

- Application au swap de deux variables

```
a,b = b,a
```



Opérateur :=

- *walrus operator*
- Le morse
- Alors que l'affectation n'a pas de valeur...
- ...l'opérateur morse réalise une affectation ET donne à l'expression entre parenthèses la valeur de la partie droite

```
>>> a = 3  
>>>
```

```
>>> (a := 3)  
>>> 3
```

(Ne pas oublier les parenthèses)

- A partir de la version 3.8

Merci !

- Restons en contact :
 - Georges Georgoulis – ggeorgoulis@alteractifs.org – 06 12 68 40 06



Coopérative d'activité et d'entrepreneurs www.alteractifs.org