Référence de la syntaxe Python 2.3

Types

Type	Catégorie	Forme littérale
None	Nulle	None
Bool	Booléen	True, False
Int	Nombres	25, -25
Long		25L, -25L
Float		25.0, 25.0E3
Complex		3+4j
String	Séquences	"abc", """..."""[1]
Unicode str.		u"abc"
Tuple		(1, 2)
List*		[], [1, 2, 3] [f(x) for x in seq]
Dict*	Mapping	{}, {"a":1; "b":2}
File	Fichiers	File("nom.txt")
Function	Exécutable	def func(): ...
Class		class objDef(): ...

* = mutable[2], (rien) = non mutable[3]

Caractères littéraux spéciaux

\	Continuation de ligne
\\	Backslash
\'	Guillemet simple
\"	Double guillemet
\n	Retour de ligne
\xhh	Valeur héxa (\x00 à \xff)
\uhhh	Valeur héxa Unicode

Convention de nommage des variables

Préfixe	Exemple	Type
s	sTexte	String
b	bValid	Booléen
i, il	iLen, ilLen	Int, Long
l	lColl	List
t	tCoord	Tuple
d	dTel	Dict

[1] les triples guillemets permettent d'écrire une chaîne de car. contenant plusieurs lignes de texte
[2] les modif. ont lieu sur l'objet
[3] les modif. ont lieu sur une copie de l'objet

Expressions

Affectation et affectation augmentée

Opération	Description
x = y	Affectation de y à x
del x	Suppression de l'objet x
x += y	x = x + y
x -= y	x = x - y (idem pour *=, /=, %=)

Opérations arithmétiques

Opération	Description
x + y	Addition
x - y	Soustraction
x * y	Multiplication
x / y	Division
x // y	Division arrondie à l'entier
x ** y	Puissance
x % y	Modulo (reste de x divisé par y)
-x	Moins unaire
+x	Plus unaire

Opérations de comparaison

Opération	Description
x < y	Plus petit que
x > y	Plus grand que
x != y	Différent de (identique à <>)
x >= y	Plus grand ou égal à
x <= y	Plus petit ou égal à
x == y	Egal à
x is y	Identique à[4] (même objet?)

Opérations logiques (booléennes)[5]

Opération	Description
x or y	si x est faux, retourne y; sinon x
x and y	si x est faux, retourne x; sinon y
not x	si x est faux, retourne 1; sinon 0

[4] Attention: souvent, x == y, mais x is not y.
[5] En Python, les expr. booléennes sont évaluées de gauche à droite et sont court-circuitées: dans x and y, y n'est évalué que si x est faux. De plus, toute valeur numérique non nulle, toute chaîne, liste, tuple ou dict. non vides sont évalués comme vrais.

Opérations sur les dictionnaires

Opération	Description
x = d[k]	Retourne l'élt de d qui a la clé k
d[k] = x	Affecte x à l'élt de d qui a la clé k
del d[k]	Efface un élément selon la clé k
len(d)	Nombre d'éléments du dict. d

Fonctions arithmétiques internes

Opération	Description
abs(x)	Valeur absolue du nombre x
divmod(x,y)	Retourne (int(x / y), x % y)
round(x,[n])	Arrondit le nombre x au multiple le plus proche de 10-n

Fonctions de conversion de type

Opération	Description
int(x[,base])	Convertit x en Integer
long(x [,base])	Convertit x en Long
float(x)	Convertit x en nb à virg. flottante
str(x)	Convertit x en sa représ. String
repr(x)	Convertit x en son expr. String
eval(str)	Evalue l'expr. str et retourne un objet
tuple(s)	Convertit la séquence s en un tuple
list(s)	Convertit la séquence s en une liste
chr(x)	Convertit l'entier x en un car.
unichr(x)	Idem, mais en un car. Unicode
ord(x)	Convertit le car. x en sa val. entière
hex(x)	Convertit l'entier x en chaîne héxa.
oct(x)	Convertit l'entier x en chaîne octale

Fonction de conversion en chaîne Unicode

Pour convertir une chaîne simple en une chaîne Unicode, utiliser la fonction interne suivante:

unicode(s [,encodage,[erreurs]])

Encodage	Description
'ascii'	ASCII 7-bits
'latin-1'	ISO 8859-1 Latin-1
'utf-8'	Unicode 8-bits à long. variable
'cp1252'	Windows Europe occidentale

Opérations sur les séquences

Opération	Description
s + r	Concatène s et r
s * n	Fait n copies de s
s % d	Met en forme d selon s[6] (voir ci-après)
s[i]	Retourne l'élt à la position i de s[7]
s[i:j]	*Slicing*; retourne les élts i à j[8] de s
s[:]	Retourne une copie[9] de s
x in s	Retourne True si x appartient à s
x not in s	Idem, si x n'appartient pas à s
for x in s: ...	Itération sur les élts de s
len(s)	Nombre d'éléments de s
min(s)	Elément le plus petit de s
max(s)	Elément le plus grand de s
s[i] = x	Affecte x à l'élt à la position i de s
s[i:j] = r	Affecte r au *slice* i:j de s
del s[i]	Supprime l'élt à la position i de s
del s[i:j]	Supprime les élts du *slice* i:j de s

Caractères de mise en forme des String

Caractères dans s utilisés avec l'opération s % d:

Caractère	Format de représentation
%d	Nombre entier
%5d	Idem sur 5 positions
%f	Nombre décimal
%.2f	Idem avec 2 décimales slt
%s	Chaîne de caractères
%25s	Idem sur 25 positions
%-25s	Idem, aligné à gauche
%c	Caractère simple
%%	Caractère %

[6] pour les types *String* seulement.
[7] le premier élément étant à la position zéro (0).
[8] élément j non compris: [1:5] contient donc 4 élts de s.
[9] copie superficielle (*shallow copy*); pour avoir une copie profonde (*deepcopy*), appeler copy.deepcopy(s)

Référence de la syntaxe Python 2.3

Expressions (suite)

Ordre d'évaluation[10]

De l'opér. la plus prioritaire à celle qui l'est le moins:

Opération	Description
(…), […], {…} `…`	Création de tuple, liste, dict. Conversion en chaîne
s[i], s[i:j] s.attr, f(…)	Indexation et slicing Attributs et appels de fonction
+x, -x, ~x	Opérateurs unaires
x ** y	Puissance
x * y, x / y, x % y	Multiplic., division, modulo
x + y, x - y	Addition, soustraction
x << y, x >> y	Glissement binaire
x & y	Et binaire
x ^ y	Ou exclusif binaire
x | y	Ou binaire
x < y, x <= y x > y, x >= y x == y, x != y x <> y x is y, x is not y x in s, x not in s	Comparaison, identité et test d'appartenance à une séquence
not x	Négation logique
x and y	Et logique
x or y	Ou logique
lamba *args*: *expr*	Fonction anonyme

Opérateur d'accès aux objets

L'opérateur **.** (point) permet d'accéder aux méthodes et attributs d'un objet.

Opération	Description
s.strip()	Appelle la méthode strip() de l'objet s
foo.x = 3	Affecte la valeur 3 à l'attribut x de l'objet foo

[10] lorsque Python est confronté à une expression contenant plusieurs opérations, il évalue les opérations dans un ordre très précis; cela s'appelle *l'ordre de précédence* des opérateurs.

Méthodes

Listes

Voir aussi *Opérations sur les séquences*, p. 1

Méthode	Description
list(s)	Convertit la séquence s en une liste
s.append(x)	Ajoute un nouvel élt x à la fin de s
s.extend(t)	Ajoute une nouv. liste t à la fin de s
s.count(x)	Compte les occurrences de x dans s
s.index(x)	Retourne le plus petit i où s[i] == x
s.insert(i,x)	Insère x à la position i
s.pop([i])	Retourne l'élt i et le retire de la liste; si i est omis, retourne le dernier.
s.remove(x)	Recherche le premier x et le retire de s
s.reverse()	Inverse la position des éléments de s
s.sort()	Trie les éléments de s

Chaînes de caractères (String)

Voir aussi *Opérations sur les séquences*, p. 1

Méthode	Description
s.center(i)	Centre s dans un champ de largeur i
s.count(sub [,start[,end]])	Compte les occurrences de la chaîne sub dans s
s.encode([enc [,errors]])	Encode s selon l'encodage enc; voir *Fonction de conversion en* chaîne Unicode (p. 1)
s.endswith(suf [,start,[end]])	Retourne True si le suffixe suf est présent à la fin de s
s.find(sub [,start[,end]])	Retourne la position de la première occurrence de sub dans s; sinon -1
s.index(sub [,start[,end]])	Idem, mais soulève une exception plutôt que la val. -1 si sub pas trouvé
s.isalnum()	Retourne True si tous les car. de s sont alphanumériques
s.isalpha()	Idem, mais s'ils sont alphabétiques
s.isdigit()	Idem, mais s'ils sont numériques
s.islower()	Idem, mais s'ils sont en minuscules
s.isspace()	Idem, mais si ce sont des espaces
s.isupper()	Idem, mais si ce sont des majuscules
s.join(t)	Joint toutes les chaînes de la liste t en utilisant le délimiteur s
s.lower()	Convertit s en minuscules
s.lstrip()	Supprime les premiers espaces de s

Chaînes de caractères (String), suite

Méthode	Description
s.replace(old ,new[,max])	Remplace les occurrences de old dans s par new
s.rfind(sub [,start[,end]])	Retourne la position de la dernière occurrence de sub dans s; sinon -1
s.rindex(sub [,start[,end]])	Idem, mais soulève une exception plutôt que la val. -1 si sub pas trouvé
s.rstrip()	Supprime les derniers espaces de s
s.split([sep [,maxsplit]])	Répartit s en une liste sur toutes les occurrences du délimiteur sep
s.splitlines([k])	Répartit les lignes de s en une liste; si k vaut 1, les retours sont conservés
s.startswith(pre [,start,[end]])	Retourne True si le préfixe pre est présent au début de s
s.strip()	Supprime les espaces au début et à la fin de s
s.swapcase()	Inverse la casse de s (min.↔maj.)
s.title()	Convertit la première lettre de chaque mot de s en maj., les autres en min.
s.translate(table)	Transcrit les caractères de s en utilisant une table de conversion
s.upper()	Convertit s en majuscules

Dictionnaires / Mapping types

Voir aussi *Opérations sur les dictionnaires*, p. 1

Méthode	Description
d.clear()	Supprime tous les élts de d
d.copy()	Retourne une copie de d
d.has_key(k)	Retourne True si d possède un élt avec la clé k; False sinon.
d.items()	Retourne une liste de tuples (k,v) avec toutes les clés et les valeurs de d
d.keys()	Retourne une liste avec les clés de d
d.update(e)	Ajoute tous les élts du dict. b à d
d.values()	Retourne une liste avec les valeurs de d
d.get(k[,v])	Retourne d[k] si trouvé; sinon v
d.popitem()	Supprime et retourne un[11] élt (k,v) de d
d.setdefault(k [,v])	Retourne d[k] si trouvé; sinon retourne v et définit d[k]=v

[11] un élément au hasard, on ne peut pas prédire lequel.

Facilités syntaxiques de Python[12]

Compactage/décomptage de tuples

Expression	Equivalent
(x,y,z) = (1,2,3)	x=1; y=2; z=3
(x,y) = divmod(25,3)	t = divmod(25,3) x = t[0]; y = t[1]

for (k,v) in dico.items():
  …
	for t in dico.items():
	  k = t[0]; v = t[1]
	  …

Opérateur lambda

voir *Programmation fonctionnelle*, p. 4

Expression	Equivalent approxim.
a = lambda x,y: x + y	def f(x,y): return x+y a = f

[12] *Syntaxic sugars* en anglais; tous les langages de progr. ont de petites facilités qui leur sont propres.

Référence de la syntaxe Python 2.3

Modèle de script

```
# -*- coding: cp1252 -*-
"""Description du programme"""

import sys

def main():
    """Programme principal"""
    instructions

if __name__ == "__main__": main()

# eof
```

Structure des lignes de code

Les instructions d'un prog. sont terminées par un retour de ligne (\n) ou par un point-virgule ;

Une longue instruction peut être répartie sur plusieurs lignes en utilisant le caractère de continuation de ligne \ :

```
a =   math.cos( 3 * ( x - n)) + \
      math.sin( 3 * ( y - n))
```

Le caractère de continuation de ligne n'est toutefois pas nécessaire à l'intérieur des parenthèses (…), crochets […], accolades { … } et triples guillemets """ … """.

Le caractère dièse # dénote un commentaire, qui s'étend jusqu'à la fin de la ligne.

L'indentation est utilisée pour dénoter les blocs de code, tels que le corps des fonctions, conditions, boucles et classes.

L'indentation doit être consistante au sein des instructions d'un bloc:

```
if expr :
    instr1    # consistant
    instr2
else :
    instr3
       instr4# pas consistant
```

Contrôle du flux d'exécution

Conditions

```
if expr:
    instructions
elif expr:
    instructions
…
else:
    instructions
```

Boucles

```
while expr:
    instructions
```

Exécute les instructions tant que *expr* est vraie.

```
for i in s:
    instructions
```

Exécute les instructions pour tous les éléments de la séquence s, en affectant chacun d'eux à i.
Si tous les éléments de s sont des tuples qui ont tous le même nombre de termes, on peut écrire:

```
for x,y,z in s:
    instructions
```

La clause else: peut être ajoutée après les boucles. Elle est exécutée seulement si la boucle arrive au terme de son exécution, soit qu'elle a effectué sa dernière itération, soit qu'elle ne s'est pas exécutée du tout; la clause n'est pas exécutée si la boucle est abandonnée à l'aide de l'instr. break:

```
while i < 10:
    faire quelque chose
    i = i + 1
else:
    print "Fait"

for i in s:
    if i == "abc":
        break
else:
    print "Pas trouvé 'abc'!"
```

Exceptions

```
try:
    faire quelque chose
except IOError, e:
    gérer l'exception e
except TypeError, e:
    gérer l'exception e
```

La clause else peut être ajoutée après la dernière clause except. Elle est exécutée si et seulement si le bloc try n'a soulevé aucune exception:

```
try:
    f = open( "fich.txt", "r")
except IOError, e:
    print "Pas pu ouvrir le fichier"
else:
    data = f.read()
    f.close()
```

La clause finally quant à elle permet de définir des instructions qui doivent être exécutées dans tous les cas, qu'une exception ait été soulevée ou non dans le bloc try:

```
f = open( "fich.txt", "r")
try:
    faire quelque chose avec le fichier f
finally:
    f.close()
    print "Le fichier a été fermé dans tous les cas."
```

Enfin, l'instruction raise peut être utilisée pour soulever volontairement une exception:

```
if( i < 10) or( i > 20):
    raise ValueError, \
        "i doit être compris entre 10 et 20."
```

Instruction pass

L'instruction pass n'effectue aucune opération. C'est une instruction «nulle».

```
def funcBidon(): pass
```

Assertions et __debug__

```
assert test [, data]
```

test est une expression qui doit retourner une valeur booléenne (True ou False). Si l'expression est évaluée à la valeur False, l'instruction assert soulève une exception AssertionError, avec le paramètre optionnel *data*.

En interne, l'instruction assert est traduite par les instructions suivantes:

```
if __debug__:
    if not( test):
        raise AssertionError, data
```

__debug__ est une variable interne de Python qui vaut toujours True, à moins que l'interpréteur n'ait été lance en mode optimisé (option -O), dans quel cas elle vaut False.

L'instruction assert peut être utilisée pour vérifier lors du développement[13] si une variable a bien la valeur supposée.

[13] Attention: lors du développement seulement! Cette instruction ne doit pas être utilisée pour exécuter du code qui doit être exécuté pour que le programme fonctionne correctement, car elle n'est pas exécutée si l'interpréteur Python est exécuté en mode optimisé. En particulier, c'est une erreur de l'utiliser pour vérifier les données entrées par l'utilisateur.

Référence de la syntaxe Python 2.3

Espaces de noms: local et global

Lorsque l'interpréteur Python rencontre un nom[14] dans une expression, il le recherche d'abord dans l'espace de noms *local*.

S'il ne l'y trouve pas, il le recherche ensuite dans l'espace de noms *global*.

S'il ne l'y trouve pas non plus, il recherche encore de l'espace de noms des fonctions internes de Python. Et s'il ne l'y trouve toujours pas, il soulève une exception NameError.

Chaque fois qu'une fonction est invoquée, un nouvel espace de nom *local* est créé. Cet espace contient les noms des paramètres de la fonction, ainsi que les noms des variables qui sont créées lors de l'exécution de la fonction.

L'espace de noms *global* d'une fonction est toujours le module dans lequel elle a été créée.

Mode d'exécution

Contexte	Mode	Exemple
Expression	Immédiat	a == 2
Instructions		print "hello"
Invoc. fonc.		func(10)
Invoc. méthode		string.upper()
Déf. fonction	Différé	def func(n): …
Déf. classe		class obj(): …
Importation		from string …
		import string

Fonctions

```
# Définition de fonction (différé)
def funcName( param1, param2):
    """Description de la fonction"""
    instructions qui utilisent param1 et param2
    return expr


# Invocation (immédiat)
result = funcName( val1, val2)
```

Programmation fonctionnelle

List comprehensions

```
[expr for elt1 in seq1
      for elt2 in seq2
      …
      for eltN in seqN
      if condition]
```

…constructeur très spécial; pour plus tard…

Opérateur lambda

```
lambda args: expr
```

(très spécial) retourne une fonction anonyme construite sur l'expression *expr*; *args* est liste de paramètres séparés par des virgules et *expr* une expression dans laquelle ces param. sont impliqués.

Fonctions map(), zip(), reduce() et filter()

…fonctions de manipulation de séquences très pratiques; pour plus tard…

Classes et objets

```
class className:
    # Variables d'instance
    …
    # Méthodes
    def funcName():
        …
```

…à étudier…

Modules

```
import string
from string import replace
dir( string)
dir( string.replace)
```

Aide

```
print funcName.__doc__
help( string)
help( string.replace)
```

[14] c'est-à-dire tout identificateur qui représente un module, une fonction ou une variable.