Basic Python

Mads M. Pedersen, mmpe@dtu.dk
David Robert Verelst

Python 3 compatibility

Use this header in Python 2 to make python 3 compatible code:

```
from __future__ import division, print_function, absolute_import, unicode_literals
range = xrange; xrange = None
```

Print-function

```
print ("hello")
print (4+5)
print ("Hello: " + name)
print (4, 5)  # commas separate values by space
print ("hello: " + str(4+5))
```

Comments

```
#one line comment

"""
Multi
line
comment
"""
```

Boolean

```
True, False
```

Operator	name	example	output
and	Logical conjunction	`True and False`	`False`
or	Logical disjunction	`True or False`	`True`
not	Negation	`not True`	`False`

Comparison

<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!= or <>	not equal
is	object identity
is not	negated object identity

Numeric types

Integer: `1, int(1.), int('1')`
Long: `1L` but don't care. Python does the job
Float: `1., float(1), float('1')`

command	name	example	output
+	Addition	`4+5`	`9`
-	Subtraction	`8-5`	`3`
*	Multiplication	`4*5`	`20`
/	Division	`19/3` `5./10`	`6.333*` `.5`
%	Remainder	`19%3`	`5`
**	Exponent	`2**4`	`16`

*Without "`from __future__ import division`" the result will be 6 in python 2

Sequences (String, list, tuple)

String: `"This is a string", 'this is a string', str(1)`
List: `[1,"a",(),[]]`
Tuple: `(1,"a",(),[])` Note: (1) is the number 'one' while (1,) is a tuple
Tuples are similar to lists but cannot be modified after creation

Command	Description	Example	Result
x in s	True if an item of s is equal to x, else False	1 in [1,2,3]	True
x not in s	False if an item of s is equal to x, else True	'a' in 'bcde'	True
s + t	the concatenation of s and t	(1,)+(2,)	(1,2)
s.append(x)	Append element x to list s	[1,2].append(3)	[1,2,3]
s.extend(t)	Extend list s with list t	[1].extend((2,3))	[1,2,3]
s.pop()			
s.remove(x)			
s * n	n copies of s concatenated	'a'*3	'aaa'
s[i]	i^{th} item of s. First item is s[0]	'abc'[1]	b
S[-1]	i^{th} item of s from the end. Last item is s[-1]	'abc'[-2]	b
s[i:j]	slice of s from i to j	[1,2,3,4][1:3]	[2,3]
s[i:j:k]	slice of s from i to j with step k	[1,2,3,4,5][1:-1:2] [1,2,3,4,5][::-1]	[2,4] [5,4,3,2,1]
len(s)	length of s	Len([1,2,3])	3
min(s)	smallest item of s	min("abc")	'a'
max(s)	largest item of s	Max((1,2,3))	3
s.index(i)	index of the first occurence of i in s	[1,2,3].index(2)	1
s.count(i)	total number of occurences of i in s	'this is it'.count('i')	3
range(x)	List from 0 to x	range(3)	[0,1,2]
range(x,y)	List form x to y	range(1,3)	[1,2]
range(x,y,z)	List from x to y step z	range(2,10,3)	[2,5,8]

String methods

Command	Description	Example	Result
str.upper()	Convert to upper case	'abc'.upper()	'ABC'
str.lower()	Convert to lower	'ABC'.upper()	'abc'
str.replace(x,y)	Replace	'abc'.replace('b','d')	'adc'
str.split(x)	Split to list of strings	'ab.cd.ef'.split('.')	['ab', 'cd', 'ef']
str.join(list)	Joins list of strings	','.join(['ab', 'cd', 'ef'])	'ab,cd,ef'

String formatting:

Include variable	'The value is: %s" % value
Include variables	'The values are: %s and %s" % (value1, value2)
Special characters	' \'quote\' '
New line	'\n'
Tab	'\t'

Formatting number variables

format_spec	[sign][#][0][width][.precision][type]		Example	Result
sign	"+":	sign always shown	"%+d"%3 "%+d"%-3	'+3' '-3'
	"-":	only negative sign shown (default)	"%-d"%3 "%-d"%-3	'3' '-3'
	" ":	add space for positive numbers	"% d"%3 "% d"%-3	' 3' '-3'
#	Prefixes oct and hex values with '0o' or '0x		"%#x"%3	'0x3'
0	Zero padding instead of space padding		"%02d"%3	'03'
width	Number of pads before decimal separator		"%2d"%3	' 3'
precision	Number of space or 0 after decimal separator		"%.2f"%3	'3.00'
type	"d"	decimal	"%d"%3	'3'
	"e"	exponent notation	"%.2e"%3	'3.00e+00'
	"E"	exponent notation	"%.2E"%3	'3.00E+00'
	"f", "F"	fixed point	"%.2f"%3	'3.00'
	"g", "G"	round/exponent notation if required	"%1.2g"%3.4 "%1.2g"%.456 "%1.2g"%234	'3.4' '0.46' '2.3e+02'
	"o"	octal	"%#o"%9	'011'
	"x", "X"	hex	"%#x"%9	'0x9'

Formatting string variables

format_spec	[align][width][type]		Example	Result
align	"+": "-":	right align(default) left align	"%+3s"%2 "%-3s"%2	' 2' '2 '
width	String length		"%6s"%2	' 2'
type	"c"	Unicode character	"%c"%2	'\x02'
	"s"	String	"%s"%234	'234'

Dictionary

```
a = dict(one=1, two=2, three=3)
b = {'one': 1, 'two': 2, 'three': 3}
c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
d = dict([('two', 2), ('one', 1), ('three', 3)])
```

Command	Example	Description
Assign	d[key] = value	Assign value to key (replace if key exists)
Contains	Key in d	Return True if d contains key else False
Loop up	d[key]	Return value assigned to key. Throws error if key not exists
Get	d.get(key, default=None)	Return value assigned to key. Return default if key not exists
Delete	del d[key]	Delete key from d. If key not exists an error is thrown
Key list	d.keys()	Returns a list with the keys (in arbitrary order)
Value list	d.values()	Returns a list with the values (in arbitrary order)
Item list	d.items()	Returns a list with the (key, value) tuples (in arbitrary order)

When to use: Use dict instead of list if you need to look up values, as the running time of look up is proportional to the length of the list while expected constant for dictionaries.

Branching

```
if x==0:
    x=1
elif x==1:
    x=2
else:
    x=3
```

Unbound loop

```
while x<10:
    pass
```

The `pass` statement does nothing except maintaining the correct indent

Bound loop

```
for x in [0,1,2]:
    print x
```

```
for x in range(3):
    print x
```

Functions

```
def x_n(x, n=2):
    return x*n
```

`x_n(2)`	4
`x_n(2,3)`	6

`x_n(x=2,n=3)#ok`
`x_n(x,n=3)  #ok`
`x_n(x=2,3)  #SyntaxError: non-keyword arg after keyword arg`	

`s = (2,3)` `x_n(*s) # * unpacks sequence s`	6
`d = {'x':2,'n':3}` `x_n(**d) # ** unpacks dict d`	6
`x_n('hi')`	'hihi'
`x_n`	<function __main__.x2>