Python 3 Quick Reference

COMMENTS

```
# single-line comment
```

KEYWORDS

and as assert break class continue def del elif else except False finally for from global if import in is lambda None nonlocal not or pass raise return True try while with yield

EXCEPTION HANDLING

```
try:
    # code to evaluate
except XXXError as e:
    # Handle XXXError
else:
    # executes if no exceptions
finally
    # always executes
```

IMPORTING MODULES

```
import module
from module import *
from module import name,...
from . import name
```

RELATIONAL and BOOLEAN OPERATORS

```
== != >  <  >= <=    and or not
```

ASSIGNMENT

normal assignment

=

self-assignment

```
+= -= *= /=
```

TYPE CONVERSIONS

str(x)	any type x to string
int(s) float(s)	string s to int or float; throws exception if fails to parse
bool(x)	any type x to Boolean
list(c)	collection c to list
tuple(L)	list L to tuple

CONTROL FLOW

```
if expr:      # code
elif expr:    # code
else:         # code
for var in iterable:    # code
while boolean-expression:   # code
```

STRINGS

literals

```
'string literal'
'''triple-delimited string'''
r'raw string'
```

" can be used instead of '

escapes

\n,\r	newline, carriage return
\f,\t	formfeed, tab
\ooo \x*hh*	ascii char with octal value *ooo*, hex value *hh*
\u*xxxx*	Unicode character with hex value *xxxx*
\N{*name*}	Unicode character with name *name*

methods

s.capitalize()	s capitalized
s.center(w) s.center(w,f)	s centered in string of width w, padded with f f is space unless specified
s.count(b) s.count(b,i) s.count(b,i,j)	# occurrences of b in s[i:j]
s.encode() s.encode(c)	s encoded to bytes using default codec, codec c
s.endswith(e,i,j)	T if s[i:j] ends with e
x.expandtabs() x.expandtabs(t)	s with tabs expanded to t spaces (default 8)
s.find(b) s.find(b,i) s.find(b,i,j)	index of 1st occurrence of b in s[i:j]
s.format(p,...)	S with placeholders filled in from parameters p...
s.format_map(m)	S with placeholders filled in from mapping m
s.index(b)	Index of b within s
s.isalnum()	T if s contains only letters and digits
s.isalpha()	T if s contains only letters
s.isdigit()	T if s contains only digits
s.isidentifier()	T if s is legal identifier
s.islower()	T if all letters in s are lowercase
s.isnumeric()	T if contains only numeric characters (digits + period and signs)
s.isprintable()	T if all characters are printable
s.isspace()	T if s contains only whitespace
s.istitle()	T if s contains only title cased words (like "Xxxx")
s.isupper()	T if all letters in s are uppercase
s.join(m)	Elements of sequence m joined with s as delimiter
s.ljust(w) s.ljust(w,c)	s left justified in string of width w, padded with c f is space unless specified
s.lower()	s converted to lowercase
s.lstrip() s.lstrip(c)	s with all characters in c stripped from beginning c defaults to whitespace
s.partition(b)	Split s into part-before-b, b, and part-after-b
s.replace(m,n) s.replace(m,n,c)	s with c occurrences of m replaced with n unlimited if c not specified
s.rfind(b) s.rfind(b,i,j) s.rindex(b) s.rindex(b,i,j)	Like find() and index(), except return last occurrence
s.rjust(w) s.rjust(w,c)	s right justified in string of width w, padded with c f is space unless specified
s.rsplit(b) s.rsplit(b,c)	list of tokens after splitting s on delimiter b if c specified, at most c splits are done **from the right**
.rstrip() s.rstrip(c)	s with all characters in c stripped from end c defaults to whitespace
s.split(b) s.split(b,c)	list of tokens after splitting s on delimiter b if c specified, at most c splits
s.capitalize()	s capitalized
	are done
s.splitlines() s.splitlines(k)	list of lines (split on \n) \n removed unless k is T
s.startswith(e,i,j)	T if s[i:j] starts with e
s.strip() s.strip(c)	s with all chars in c stripped from both ends c defaults to whitespace
s.swapcase()	s with case of all letters inverted
s.title()	s converted to title case
s.translate(t) s.translate(t,d)	s with chars in table t translated' chars in string d deleted if d specified t must be exactly 256 chars long
s.upper()	s converted to upper case
s.zfill(w)	s left-padded with zeros to width w

1. i,j default to 0,len(s)
2. isXXX() functions return F if len(s) == 0

FORMATTING

```
"format".format(p1,p2,...)
format contains fields:
{n} {n:t} {n:wt} {n:fwt}
n=param # w.m=min.max width
 t=type
```

type is one of:

d	decimal integer
o	octal integer
u	unsigned decimal integer
x	hex integer
e,E	scientific notation (lower, UPPER case)
f,F	floating point
c	character
r	string (using **repr()** method)
s	string (using **str()** method)
{{ }}	literal braces

flag is one of:

<	left justify (default)
>	Right justify
0	left-pad number with zeros
+	precede number with + or -
	(blank) precede positive number with blank, negative with -

INPUT AND OUTPUT

write to STDOUT

```
print(item,...,sep=' ',end='\n')
```

read from file

```
with open("filename") as f:
    for line in f:
        # do something with line
    m = f.read()
    m = f.readlines()
```

write to file

```
with open("filename","w") as f:
    f.write(s)
    f.writelines(m)
```

binary files

```
append 'b' to mode 'r' or 'w'
```

FUNCTIONS

defining

```
def name(arg, arg=default,
*optional-args,**keyword-args):
   # statements
   return value
```

lambda function

```
ref = lambda args, ...: expr
```

ALL COLLECTIONS

sequences, dictionaries, sets

functions and operators

x in c	True if x is equal to an item of c
len(c)	number of elements in s
min(c)	smallest item of s
max(c)	largest item of s

ALL SEQUENCE TYPES

lists,tuples,strings,Unicode strings

indexing and slicing

```
s[i:j:k]
```

all s[n] such that i <= n < j
i is incremented by k
default values:
i = 0 j=**len**(s) k = 1

methods and operators

s + t	concatenate s and t
s * n , n * s	n shallow copies of s concatenated
s.count(x)	count of elements whose value is x
s.index(x[, i[, j]])	index of first element whose value is xt

LISTS

declaring

```
L = []
L = [item1,item2,...]
L = list(iterable)
```

methods and operators

L.append(o)	Append object o
L.extend(s)	Append each object in s
L.insert(i, o)	insert o at offset i
L.sort()	Sort L in place
L.pop() L.pop(n)	Remove element n (default last)
L.remove(o)	Remove o fr

TUPLES

declaring

```
T = (item1,item2,...)
T = (item,)  tuple w/ 1 value
T = item1,item2,...
```

LIST COMPREHENSIONS

```
L = [ expr for v in seq ]
L = [ expr for v in seq if expr2 ]
```

GENERATOR EXPRESSIONS

```
G = ( expr for v in seq )
G = ( expr for v in seq if expr2 )
```

DICT COMPREHENSIONS

```
D = { kx: vx for k in seq }
D = { kx: vx for k in seq if expr }
```

kx/vx: key expression/value expression

ALL MAPPING TYPES
dictionaries,sets,frozensets

methods

m.clear()	remove all elements
m.copy()	offset where match ends
m.update(n)	add elements in n to m

DICTIONARIES

declaring

```
d = { key1:val1,k2:v2,...}
d = dict(K1=V1, K2=V2, ...)
```

indexing

```
d['key1']
```

methods

d.get(k) d.get(k,v)	d[k]if k in d, otherwise v (default None)
d.items()	Iterator of elements as key/value tuples for
d.keys()	Iterator of all keys
d.values()	Iterator of all values

SETS

declaring

```
s = {item1,item2,...}
s = set(iterable)
s = frozenset(iterable)
```

frozenset is immutable

methods and operators

s.add(o)	add o to s
s.remove(o)	remove o from s
s1 & s2	insersection of s1 and s2
s1 | s2	union of s1 and s2
s1 - s2	difference of s1 and s2
s1 ^ s2	symmetric difference of s1 and s2 (AKA xor)

REGULAR EXPRESSIONS

```
import re
r = re.compile(regex)
```

re object methods

r.search(s)	return match object (=True) if s contains RE compiled to r
r.findall(s)	return list of matches as strings
r.finditer(s)	iterable object – provides match objects
r.sub(s1,s2)	return s2 with s1 substituted for the RE *s1 can be a callback function*
r.subn(s1,s2)	same, but returns tuple with s2 and # replacements
r.split(s)	returns list of tokens after splitting s on RE

match object methods

m.start()	offset where match starts
m.end()	offset where match ends
m.group(n)	capture group n (default 0)
m.group(s)	capture group named s
m.groups()	list of all capture groups

basic RE metacharacters

one character matches

.	any character
[abc] [^abc]	any character in, not in abc
\w \d \s	1 word char, digit, space char
\W \D \S	complements of \w,\d,\s

quantifiers (repeat counts)

* + ?	0 or more, 1 or more, 0 or 1
{m} {m,} {m,n}	m repeats, >= m repeats, m-n repeats

anchors

^ $ \b	beg of str, end of str, beg/end of word

grouping and alternation

a|b	a or b
(*pat*)	group and capture
(?P<*name*>*pat*)	named capture

NUMBERS

literals

decimal `123 123.455 4.234e9`
hex `0xBEAD` *octal* `027 or 0o27`
binary `0b10111011`

methods and operators

x + y	sum of **x** and **y**
x – y	difference of **x** and **y**
x * y	product of **x** and **y**
x / y	quotient of **x** and **y** (always returns float)
x // y	quotient of **x** and **y** (rounded to next lower whole float)
x % y	remainder of **x / y**
-x	**x** negated
abs(x)	absolute value of **x**
int(x)	integer value of **x**
long(x)	**x** as long integer
float(x)	**x** as float
complex(r,i)	complex with real **r** and imaginary **i**
divmod(x, y)	the pair of values (**x // y**, **x % y**)
pow(x,y) x ** Y	**x** raised to power **y**

BITWISE OPERATORS

x & y	x ANDed with y
x | y	x Ored with y
x ^ y	x XORed with y
x >> s	x right-shifted s bits
x << s	x left-shifted s bits

s must be positive