Cheatography

Python Cheat Sheet

by Gaston via cheatography.com/24027/cs/5463/

Identifier classes

__*__	System-defined names
_*	Module private name, not imported via import *
__*	Class-private name

Tuple

(obj1, obj2, ...)	tuple literal (parenthesis are optional, at least one comma is required)
(obj1 ,)	singleton, tuple with only one object (parenthesis are optional)
( )	empty tuple

A tuple is an immutable sequence of objects

List

[obj1, obj2, ...]	list literal
[obj1]	single item list
[ ]	empty list

A list is a mutable sequence of objects

Slices

s[i]	Item at index i of sequence s (not a slice)
s[i:j]	Items i to j-1 of s
s[i:j:k]	every k^{th} item starting at index i and not including index j or higher
s[:]	"s[:]" is a copy of seqquence s, while "s" is a reference to the same sequence

A Slice is a part of a sequence

Indexes start at 0 for the first item

Negative indexes start at the end of the sequence (eg. -1 is last element, -2 second last,...)

Dictionaries

{ key1: itm1, key2: itm2, ... }	Dictionary of items referenced by key
d[key]	return item referened by given key
d[key] = x	Add (or replaces) given key with attached item x
del d[key]	Remove d[key] from d. Raises KeyError if key not present in d
key in d	returns *True* if key is in dictionary
key not in d	returns *False* if key is in dictionary
d.items()	returns a list of key,value pairs

Dictionaries (cont)

d.keys()	returns list of keys in dictionary
d.values()	returns a list of the dictionaries values
d.copy()	shallow copy of the dictionary (only top level objects are copies, any object remains is a reference to the "old" object)
d.deepcopy()	deep copy of the dictionary
d.get(key[, default])	returns d[key] if th ekey is present, or default if not. If default is not passed, None is returned
d.pop(key[, default])	same as get() except that the key,value pair is also removed from the dictionary
d.popitem()	pops an arbitrary (key, value) pair from the dictionary.
d.setdefault(key[, default])	returns d[*key*] if it exists, otherwhise does *d[key]=default* and returns *default* as well. *default* defaults to None
viewitems() viewkeys() viewvalues()	returns dictionary view object contining either (key,value) pairs, only keys or only values

1) Keys are unique

2) If the key is a string it needs to be quoted
3) for loop "for k in dict" returns the keys, not the item

(Im)mutable sequences

x in s	*True* if an item of s is equal to x
x not in s	*False* if an item of s is equal to x
len(s)	number of items in s
min(s)	smallest item of s
max(s)	largest item if s
s.index(x)	first index of item equal to x in s
s.count(x)	number of occurences of x in s

These operations apply to mutable and immutable sequences

Sequence indexes start at 0 (first element), negative indexes start at the end (inex -1 is last element, -2 second last,...)
s: sequence
x : object/item

Cheatography

Python Cheat Sheet

by Gaston via cheatography.com/24027/cs/5463/

Mutable sequences

s[i] = x	replace item i of sequence s by x
s[i:j] = x	replace items i to j-1 (slice) by x
s[i:j:k] = x	replace each k^{th} item of s starting at i and not including j or higher by x
del <slice>	delete the specified slice from the sequence
s.append(x)	appends obect x to the end of the sequence s
s.extend(s2)	appends items of sequence s2 to s
s.insert(i, x)	inserts object x before the current item at index i
s.pop(i)	Removes item i from the sequence and returns it. If i is omitted i=-1 is ssumed
s.remove(x)	same as del s[s.index(x)]
s.reverse()	reverse the order of the items in s in place
s.sort([cmp [, key [, reverse]]])	sort elements in place (1)

(1) cmp is optional comparison function with 2 arguments (the items to be compared) (i.e. func(a,b)) and returning -1 if a<b, 0 if a==b and 1 if a>b
(2) key is an optional function taking one argument (item from the sequence) and used to extract the actual information from that item to be compared
(3) reverse, if True, causes the sort result to have reverse order
(4) in general specifying key and reverse (if possible) is much faster than the equivalent cmp function

Classes

```
class <classname>:(1)
  <static-var>=<value> (2)
  def __init__(self,<args>...): (3)
    <class constructor>
    self.<attribute>=<value> (4)
  def method(self,<args>,...): (5)
    <method body>
  def __privateMethod(self, <args>...): (6)
    <method body>
```

(1) Start definition of class
(2) Static variable, same value for all instances and referenced via <classname>.<varname> rather than via self
(3) Class constructor called via <calssname>() with optional arguments
(4) Attribute local to the instance
(5) Class method definition. Each and every method in a class has at least one argument, "self" referring the class instance
(6) private methods are not intended to be used from outside the class and are marked by two leading underscores

Special class methods

__init__(self[, ...])	Called for any new instance creation in order to initialize it. Derived classes must call their base class __init__ explicitely via <baseclass>.__init__(self,...) where self needs to be explicitely passed
__del__(self)	Called when the instance is to be destroyed
__repr__(self)	Called by the repr() function to return a string containing the classes representation. Usually used for debugging. If thsi method is defined and __str()__ is not, this method is also used for string conversions
__str__(self)	called by the str() function and by "print"
__lt__(self,other) __le__(self,other) __eq__(self,other) __ne__(self,other) __gt__(self,other) __ge__(self,other)	Rich comparison methods returning True if self<other, self<=other, self==other, self!=other. There is no defined relationship, i.e. x==y True doe not imply that x!=y is False