Commonly Used Python Methods (Python 3.x) v2.2

Strings

Slicing strings in Python:

```
# index start through end-1
S[start:end]

# index start through remaining string
S[start:]

# index from the beginning [0] through end-1
S[:end]

# a complete copy of S
S[:]
```

str.capitalize()
Return a copy of the string with its first character capitalized and the rest lowercased.

str.count(sub[, start[, end]])
Return the number of non-overlapping occurrences of substring sub in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.

str.isalnum()
Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

str.isalpha()
Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.

str.islower()
Return true if all cased characters [4] in the string are lowercase and there is at least one cased character, false otherwise.

str.isnumeric()
Return true if all characters in the string are numeric characters, and there is at least one character, false otherwise.

str.isspace()
Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.

str.istitle()
Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

str.isupper()
Return true if all cased characters [4] in the string are uppercase and there is at least one cased character, false otherwise.

str.join(iterable)
Return a string which is the concatenation of the strings in iterable. A TypeError will be raised if there are any non-string values in iterable, including bytes objects.

str.lower()
Return a copy of the string with all the cased characters [4] converted to lowercase.

str.replace(old, new[, count])
Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

str.split(sep, maxsplit)
Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements). If maxsplit is not specified or -1, then there is no limit on the number of splits (all possible splits are made).

str.strip([chars])
Return a copy of the string with the leading and trailing characters removed. The chars argument is a string specifying the set of characters to be removed. If [chars] is omitted, all non-printing chars are removed (space, tab and newline).

str.title()
Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

str.upper()
Return a copy of the string with all the cased characters converted to uppercase.

Lists

list.append(x)
Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.insert(i, x)
Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x)
Remove the first item from the list whose value is x. It is an error if there is no such item.

list.pop([i])
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.

list.clear()
Remove all items from the list. Equivalent to del a[:].

list.index(x[, start[, end]])
Return zero-based index in the list of the first item whose value is x. Raises a ValueError if there is no such item.

list.count(x)
Return the number of times x appears in the list.

list.sort(key=None, reverse=False)
Sort the items of the list in place (the arguments can be used for sort customization, see sorted() for their explanation).

list.reverse()
Reverse the elements of the list in place.

list.copy()
Return a shallow copy of the list. Equivalent to a[:].

Sets

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary.

Examples:

```
>>> basket = {'apple', 'orange', 'apple', 'pear',
'orange', 'banana'}
>>> print(basket) # duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket # fast membership testing
True
>>> 'crabgrass' in basket
False
```

Demonstrate set operations

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a  # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a – b  # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b  # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b  # letters in both a and b
{'a', 'c'}
>>> a ^ b  # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Tuples

A tuple consists of a number of values separated by commas, for instance:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
# Tuples may be nested:
>>> u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
# Tuples are immutable:
t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment
# but they can contain mutable objects:
>>> v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

On output tuples are always enclosed in parentheses, so that nested tuples are interpreted correctly; they may be input with or without surrounding parentheses, although often parentheses are necessary anyway (if the tuple is part of a larger expression).

Though tuples may seem similar to lists, they are often used in different situations and for different purposes. Tuples are immutable, and usually contain a heterogeneous sequence of elements that are accessed via unpacking or indexing (or even by attribute in the case of *namedtuples*). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

A special problem is the construction of tuples containing 0 or 1 items: the syntax has some extra quirks to accommodate these. Empty tuples are constructed by an empty pair of parentheses; a tuple with one item is constructed by following a value with a comma (it is not sufficient to enclose a single value in parentheses).

Dictionaries:

Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

It is best to think of a dictionary as an unordered set of key:value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the

dictionary; this is also the way dictionaries are written on output.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, in arbitrary order (if you want it sorted, just use sorted(d.keys()) instead). To check whether a single key is in the dictionary, use the in keyword.

Files

open()

Returns a file object, and is most commonly used with two arguments: open(filename, mode).
>>> f = open('workfile', 'w')

The first argument is a string containing the filename. The second argument is another string containing a few characters describing the way in which the file will be used. mode can be 'r' when the file will only be read, 'w' for only writing (an existing file with the same name will be erased), and 'a' for appending; any data written to the file is automatically added to the end. 'r+' for both reading and writing. The mode argument is optional; 'r' will be assumed if it's omitted.

f.read(size)

Reads some quantity of data and returns it as a string (in text mode) or bytes object (in binary mode). size is an optional numeric argument. When size is omitted or negative, the entire contents of the file will be read and returned. If the end of the file has been reached, f.read() will return an empty string ('').

f.readline()

Reads a single line from the file; a newline character (\n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn't end in a newline. This makes the return value unambiguous; if f.readline() returns an empty string, the end of the file has been reached.

f.readlines() or list(f)

Use either of these if you want to read all the lines of a text file into a list.

f.write(string)

Writes the contents of string to the file, returning the number of characters written.

f.seek(offset, from_what)

The position is computed from adding offset to a reference point; the reference point is selected by the from_what argument. A from_what value of 0 measures from the beginning of the file, 1 uses the current file position, and 2 uses the end of the file as the reference point. from_what can be omitted and defaults to 0, using the beginning of the file as the reference point.

f.close()

Close the file and immediately free up any system resources used by it. If you don't explicitly close a file, Python's garbage collector will eventually destroy the object and close the open file for you, but the file may stay open for a while.