Short Python function/method descriptions:

__builtins__:

```
input([prompt]) -> str
  Read a string from standard input. The trailing newline is stripped. The prompt string,
  if given, is printed without a trailing newline before reading.
len(x) -> int
  Return the length of the list, tuple, dict, or string x.
open(name[, mode]) -> file open for reading, writing, or appending
  Open a file.  Legal modes are 'r' (read), 'w' (write), and 'a' (append).
print(value) -> None
  Prints the value.
range([start], stop, [step]) -> list-like-object of int
  Return the integers starting with start and ending with stop - 1 with step specifying
  the amount to increment (or decrement).
  If start is not specified, the list starts at 0.  If step is not specified,
  the values are incremented by 1.
```

importing:

```
from file_name import function_name
  Import function with name function_name from file with name file_name
```

functions:

```
def function_name(args):
   """docstring describing function here"""
   #function body here
```

for loops:

```
for obj in list:
   #for loop with obj here
  Loop over each object obj in list (list can be a range of ints)
```

while loops:

```
while condition:
   #while loop body here
  Loop until condition is not met
```

if/else:

```
if condition1:
   #do if condition1 is met
elif condition2
   #do if condition1 is not met, but condition2 is met
else
   #do if none of the above conditions are met
```

doctest:

```
import doctest

def func_to_test(val):
  """docstring describing function
  >>> func_to_test(testval_1)
  expected output
  >>> func_to_test(testval_2)
  expected output
  ...
  """
  #function body here

doctest.testmod()

Tests that func_to_test returns the expected outputs for values testval_1 and testval_2
```

unit test:

```
import unittest

class SomeTestCase(unittest.TestCase):
  def test_some_value(self):
     self.assertTrue(function_to_test(val_to_test))

 unittest.main()

Tests function_to_test using value val_to_test, and asserts that this will return True.
Common assertions include:
  assertEqual
  assertNotEqual
  assertTrue
  assertFalse
```

dict:

```
D[k] --> object
  Produce the value associated with the key k in D.
del D[k]
  Remove D[k] from D.
k in d --> bool
  Produce True if k is a key in D and False otherwise.
D.get(k) -> object
  Return D[k] if k in D, otherwise return None.
D.keys() -> list-like-object of object
  Return the keys of D.
D.values() -> list-like-object of object
  Return the values associated with the keys of D.
D.items() -> list-like-object of tuple of (object, object)
  Return the (key, value) pairs of D, as 2-tuples.
```

safely open a file:

```
with open('filename.txt') as F:
  #Do something with F
 Ensures exit method close() will be called on F even if any error happens in the block
```

file open for reading:

```
for line in F:
   #Do something with line
  Loop through each line of the file
```

file open for writing:

```
F.write(x) -> int
  Write the string x to file F and return the number of characters written.
```

list:

```
x in L --> bool
  Produce True if x is in L and False otherwise.
L.append(x) -> None
  Append x to the end of the list L.
L.index(value) -> int
  Return the lowest index of value in L.
L.insert(index, x) -> None
  Insert x at position index.
L.remove(value) -> None
  Remove the first occurrence of value from L.
L.sort() -> None
  Sort the list in ascending order *IN PLACE*.
L[start:end] -> list
  Slices the list starting at the start index, until (excluding) the end index
```

str:

```
x in s --> bool
  Produce True if and only if x is in s.
str(x) -> str
  Convert an object into its string representation, if possible.
S.find(sub[, i]) -> int
  Return the lowest index in S (starting at S[i], if i is given) where the
  string sub is found or -1 if sub does not occur in S.
S.index(sub) -> int
  Like find but raises an exception if sub does not occur in S.

S.split([sep]) -> list of str
  Return a list of the words in S, using string sep as the separator and
  any whitespace string if sep is not specified.
S.strip([chars]) -> str
  Return a copy of S with leading and trailing whitespace removed.
  If chars is given and not None, remove characters in chars instead.
```

main block:

```
if __name__ == '__main__':
   #main block code here
```