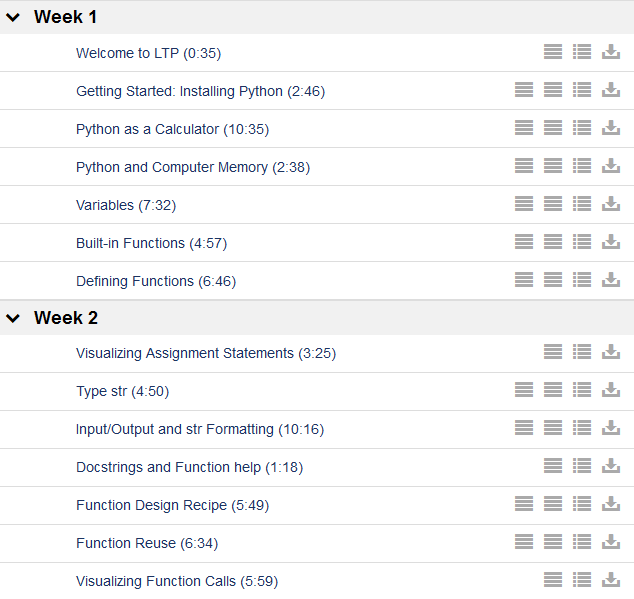
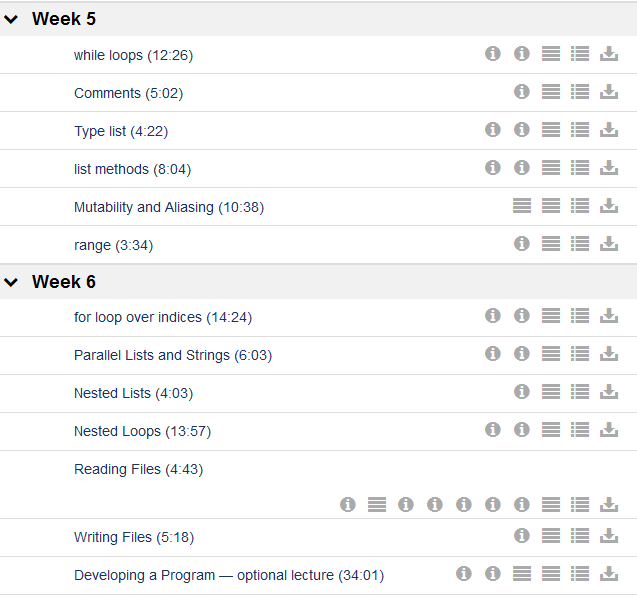
**Python Video Tutorials -- Coursera**

[**WWW.COURSERA.ORG**](http://WWW.COURSERA.ORG)



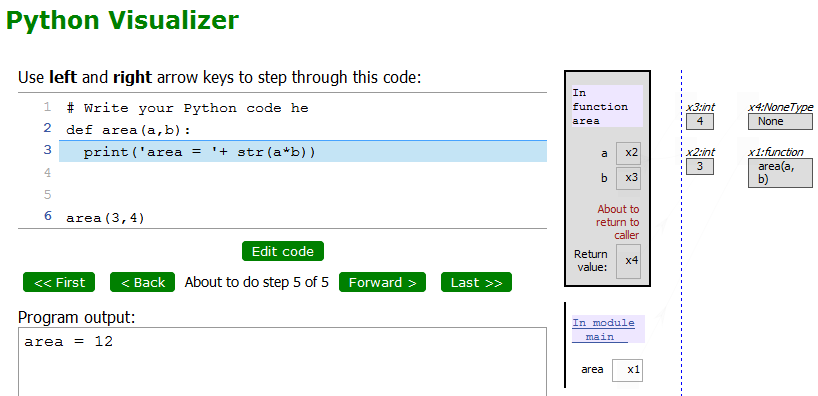






## Python Visualizer

<http://ec2-174-129-21-232.compute-1.amazonaws.com/crs-ltp/tutor.php#mode=edit>



======================================================

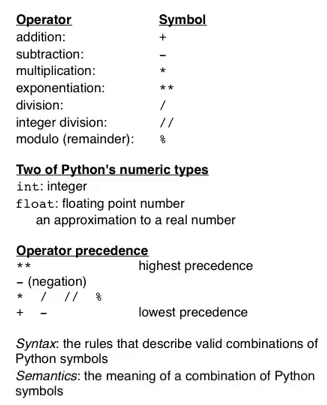
## Installing Python (1/43)

<http://www.python.org/download/>

[Python 3.3.0 Windows X86-64 MSI Installer](http://www.python.org/ftp/python/3.3.0/python-3.3.0.amd64.msi) (Windows AMD64 / Intel 64 / X86-64 binary

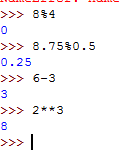


## Numbers and Math (2/43)

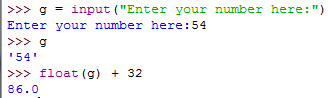


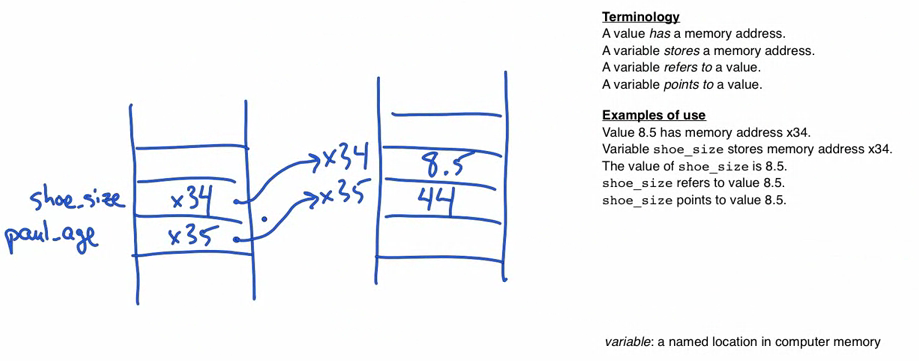
% - module -- gives a remainder from a division

\*\* -- equals to ^ sign. 2\*\*3 = 8 = 2^3



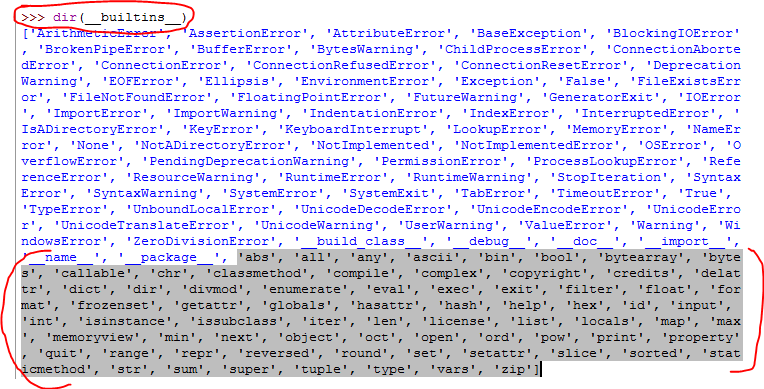
## Variables (3/43)





**Build In Functions**

To see the build in functions

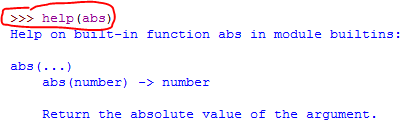


**>>> dir(\_\_builtins\_\_)**

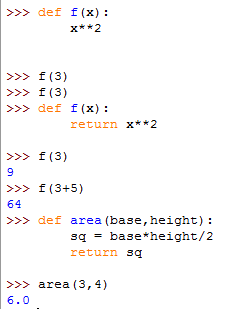
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '\_', '\_\_build\_class\_\_', '\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_name\_\_', '\_\_package\_\_', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

---

**Function help**

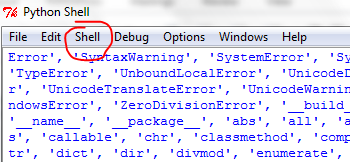


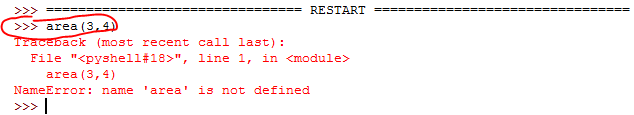
**Defined Functions**



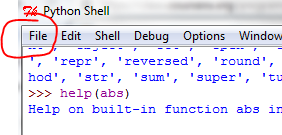
**Shell 🡪 Restart Shell**

This resets all the functions stored in the memory

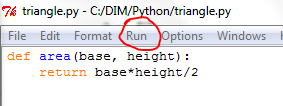




**To save the script:** File 🡪 New Window



**To run the script.** Open the script and in the script window: Run 🡪 Run Module (F5)

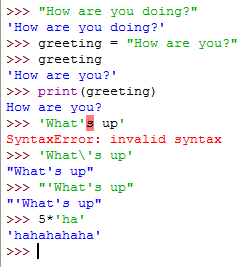


# Type str: Strings in Python

The escape sequence \' indicates that the second quote is simply a quote, not the end of the string:

>>> storm\_greeting = 'Wow, you\'re dripping wet.'

"Wow, you're dripping wet."



An alternative approach is to use a double-quoted string when including a a single-quote within it, or vice-versa. Single- and double-quoted strings are equivalent. For example, when we used double-quotes to indicate the beginning and end of the string, the single-quote in you're no longer causes an error:

>>> storm\_greeting = "Wow, you're dripping wet."

"Wow, you're dripping wet."

### String Operators

|  |  |  |  |
| --- | --- | --- | --- |
| Expression | Description | Example | Output |
| str1 + str2 | concatenate str1 and str1 | print('ab' + 'c') | abc |
| str1 \* int1 | concatenate int1 copies of str1 | print('a' \* 5) | aaaaa |
| int1 \* str1 | concatenate int1 copies of str1 | print(4 \* 'bc') | bcbcbcbc |

Note: *concatenate* means to join together

The \* and + operands obey by the standard precedence rules when used with strings.

# Input/Output and str Formatting

#-----------------------------------------

# Week2 -- Input/Output and str Formatting

#-----------------------------------------

print("Hello", "there!") # prints both words with a "space" in between

def square\_return(num):

return num\*\*2

def square\_print(num):

print("The square of num is", num\*\*2)

def sum(number1, number2):

return number1 + number2 # return the value and stops here.

# The "print" command will not be executed

print("hello")

def sum2(number1, number2):

print("hello")

return number1 + number2

#---------------------------------

# Get input from a user

#---------------------------------

name = input("What is your name? ")

location = input("What is your location? ")

print(name, "lives in ", location)

#---------------------------------------

print('3,\t4,\t5') # \t --> tabulation symbols

print(name, '''\n lives in \n''', location) # \n --> print on new line

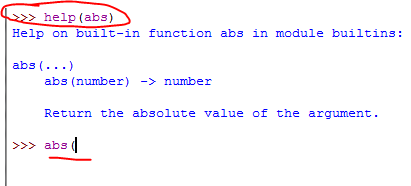
# must use ''' to use \n

print('You\'e very welcome!!!') # \ --> escape character

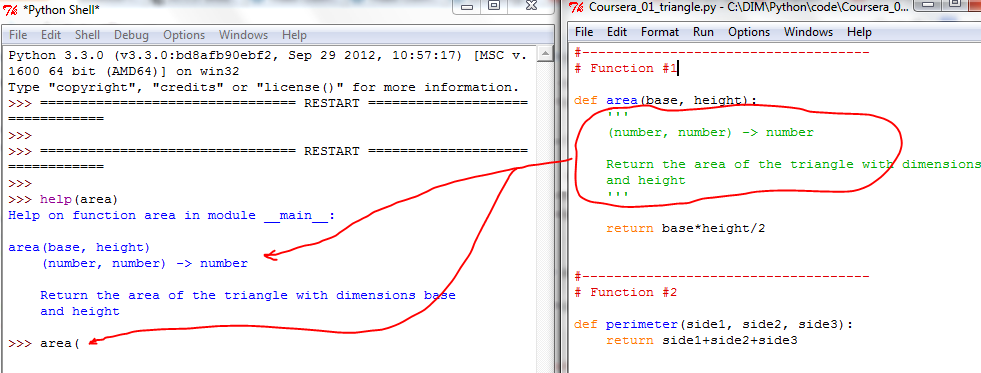
## Docstrings and Function Help

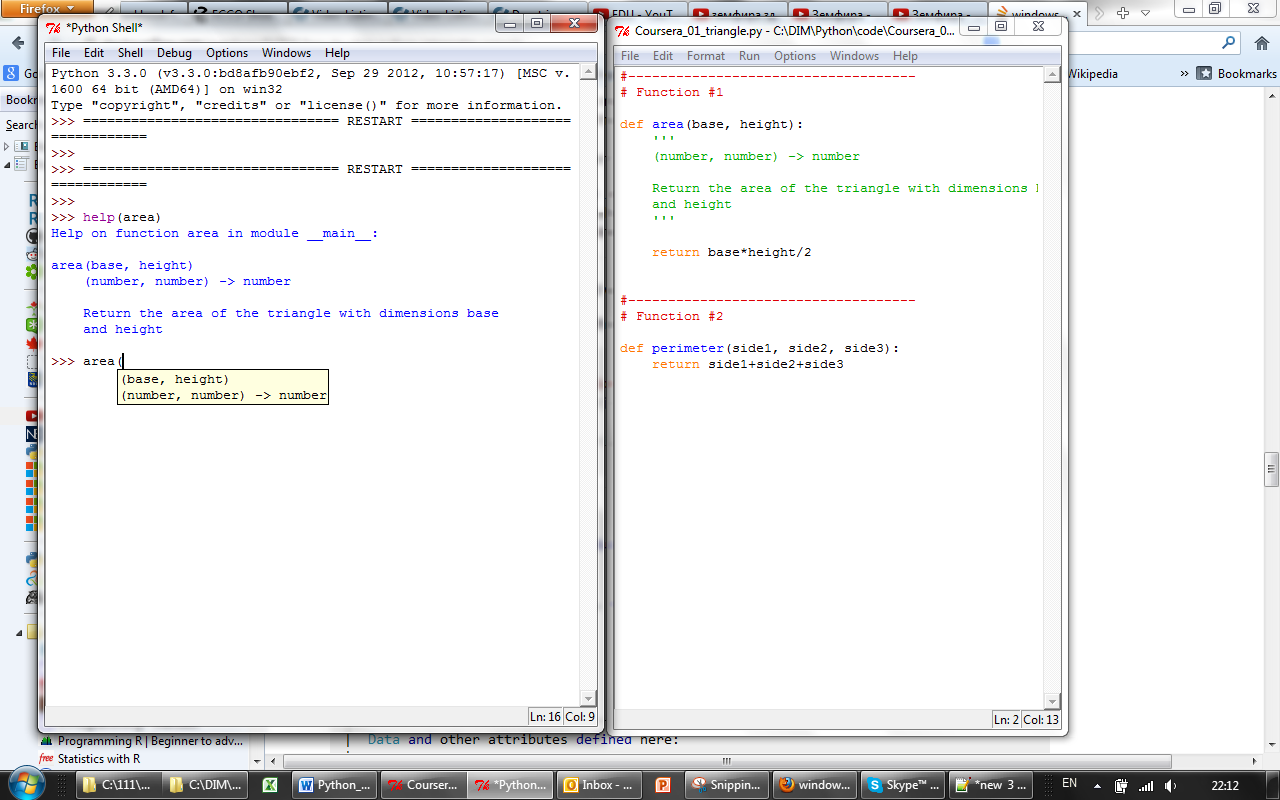
To access the function help we can print help(<function anme>)

Or start printing the function and the prompt will pop up…

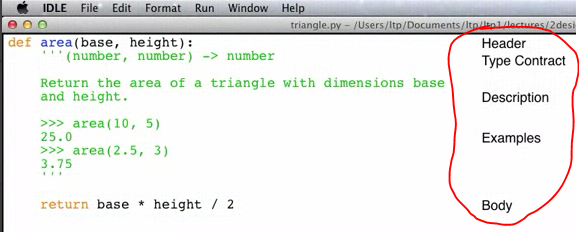


We can also add the pop up explanation to the functions that we created.

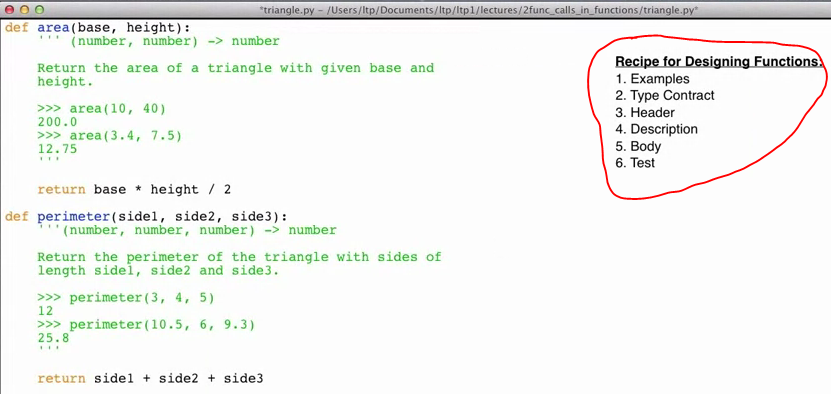


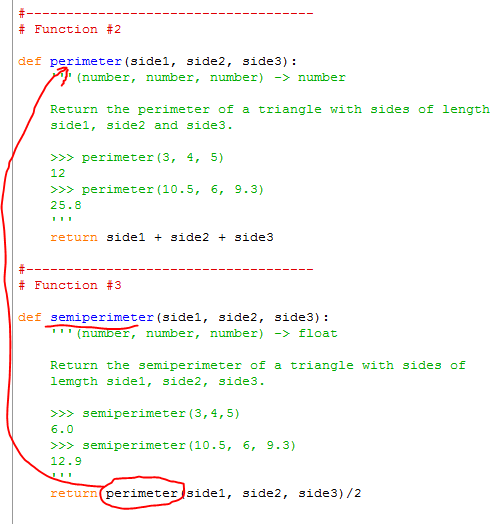


## Function Design



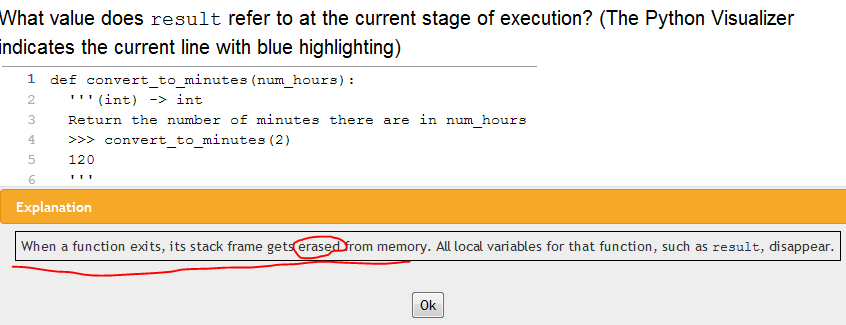
## Function Reuse (Week 2 -- Coursera)





#------------------------------------------------

Week 2 -- [Visualizing Function Calls (5:59)](https://class.coursera.org/programming1-2012-001/lecture/44)



## Type bool: Booleans in Python

### Boolean values

The Python type bool has two values: True and False.

### Comparison operators

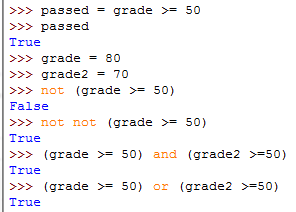
The comparison operators take two values and produce a Boolean value.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Operator | Example | Result of example |
| less than | < | 3 < 4 | True |
| greater than | > | 3 > 4 | False |
| equal to | == | 3 == 4 | False |
| greater than or equal to | >= | 3 >= 4 | False |
| less than or equal to | <= | 3 <= 4 | True |
| not equal to | != | 3 != 4 | True |

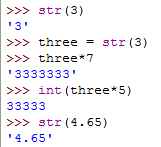
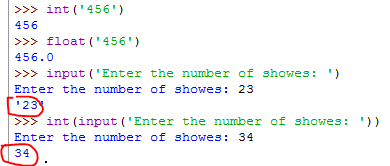
### Logical operators

There are also three logical operators that produce Boolean values: and, or, and not.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Operator | Example | Result of example |
| not | not | not (80 >= 50) | False |
| and | and | (80 >= 50) and (70 <= 50) | False |
| or | or | (80 >= 50) and (70 <= 50) | True |



## Converting between int, str, and float (COURSERA)

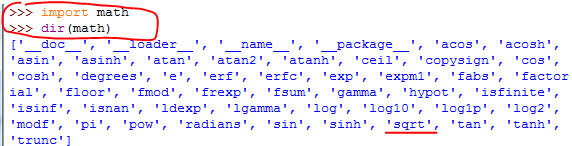


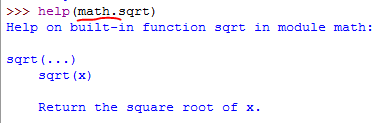
# Import: Using Non-Builtin Functions (COURSERA)

To import functions

Use 'import' to import a module

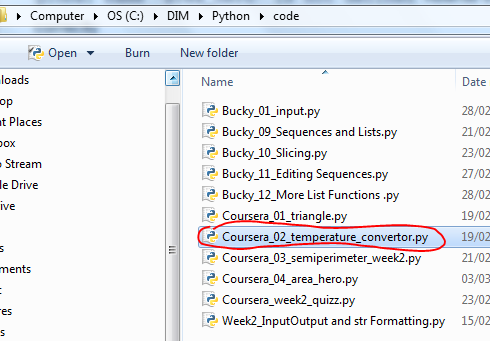
dir -- to view the functions available in the module





**We can also import modules that we wrote.**

This file has to be in the same directory as our script file.



The imported module should be in the same directory as the module that imports it.

#########------------------------------------

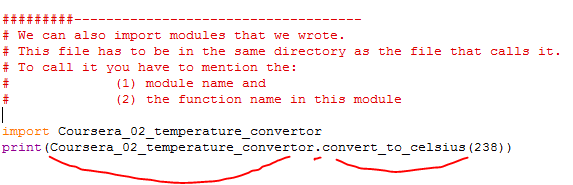
# We can also import modules that we wrote.

# This file has to be in the same directory as the file that calls it.

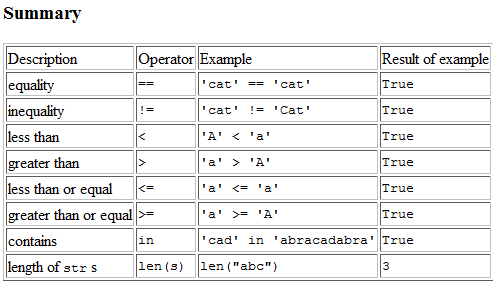
# To call it you have to mention the (1) module name and (2) the function name in this module

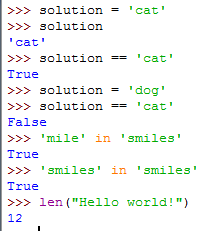
import Coursera\_02\_temperature\_convertor

print(Coursera\_02\_temperature\_convertor.convert\_to\_celsius(238))

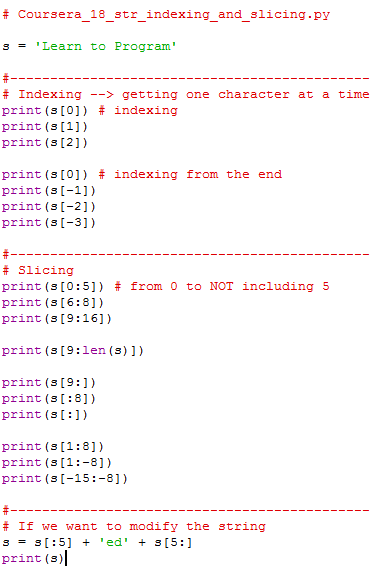


**17\_More str Operators.htm (Coursera)**





**18\_str indexing and slicing.htm (Coursera)**



**19\_str Methods: Functions Inside Objects (Coursera)**

### Methods

A method is a function inside of an object.

The general form of a method call is:

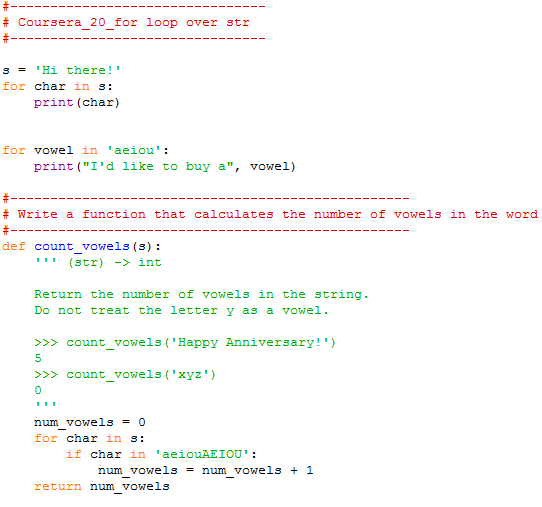
object.method(arguments)

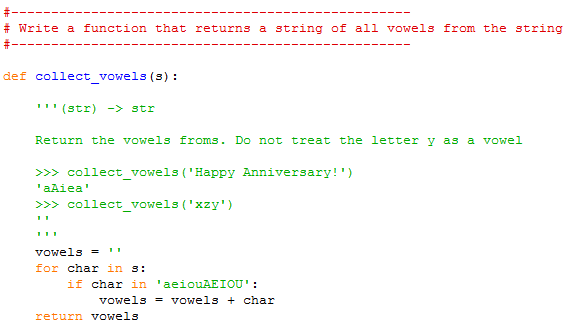
**>>> dir(str)**

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_',

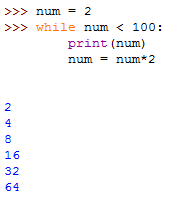
|  |  |  |  |
| --- | --- | --- | --- |
| capitalize | isalnum | join | rsplit |
| casefold | isalpha | ljust | **rstrip** |
| center | isdecimal | **lower** | split |
| count | isdigit | **lstrip** | splitlines |
| encode | isidentifier | maketrans | startswith |
| endswith | islower | partition | **strip** |
| expandtabs | isnumeric | replace | swapcase |
| **find** | isprintable | **rfind** | title |
| format | isspace | rindex | translate |
| format\_map | istitle | rjust | **upper** |
| **index** | isupper | rpartition | zfill |

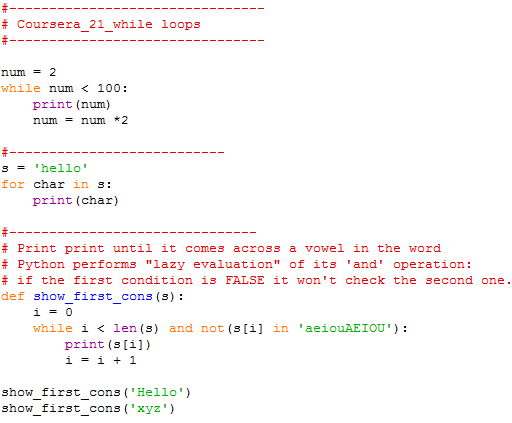
## Coursera\_20\_for loop over str

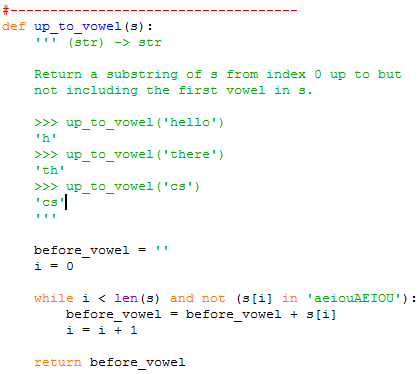




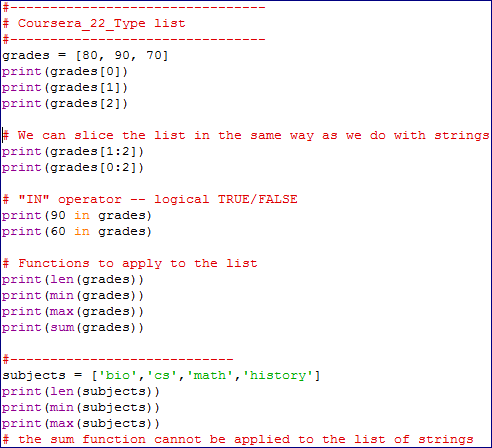
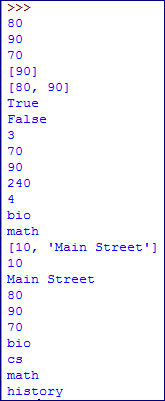
## 21\_while loops.htm

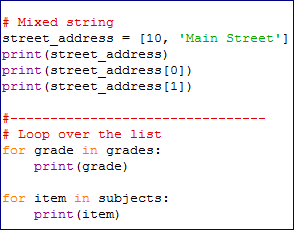






## Type list (Coursera, week 5)





## list Methods (Coursera, week 5)

### Methods

A method is a function inside an object. You can find out the methods in type list by typing dir(list).

### Modifying Lists

The table below contains methods that modify lists.

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| list.append(object) | Append object to the end of list. | >>> colours = ['yellow', 'blue']  >>> colours.append('red')  >>> print(colours)  ['yellow', 'blue', 'red'] |
| list.extend(list) | Append the items in the list parameter to the list. | >>> colours.extend(['pink', 'green'])  >>> print(colours)  ['yellow', 'blue', 'red', 'pink', 'green'] |
| list.pop([index]) | Remove the item at the end of the list; optional index to remove from anywhere. | >>> colours.pop()  'green'  >>> print(colours)  ['yellow', 'blue', 'red', 'pink']  >>> colours.pop(2)  'red'  >>> print(colours)  ['yellow', 'blue', 'pink'] |
| list.remove(object) | Remove the first occurrence of the object; error if not there. | >>> colours.remove('green')  Traceback (most recent call last):  File "", line 1, in  colours.remove('green')  ValueError: list.remove(x): x not in list  >>> colours.remove('pink')  >>> print(colours)  ['yellow', 'blue'] |
| list.reverse() | Reverse the list. | >>> grades = [95, 65, 75, 85]  >>> grades.reverse()  >>> print(grades)  [85, 75, 65, 95] |
| list.sort() | Sort the list from smallest to largest. | >>> grades.sort()  >>> print(grades)  [65, 75, 85, 95] |
| list.insert(int, object) | Insert object at the given index, moving items to make room. | >>> grades.insert(2, 80)  >>> print(grades)  [65, 75, 80, 85, 95] |

### Getting Information from Lists

The table below contains methods that return information about lists.

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| list.count(object) | Return the number of times object occurs in list. | >>> letters = ['a', 'a', 'b', 'c']  >>> letters.count('a')  2 |
| list.index(object) | Return the index of the first occurrence of object; error if not there. | >>> letters.index('a')  0  >>> letters.index('d')  Traceback (most recent call last):  File "", line 1, in  letters.index('d')  ValueError: 'd' is not in list |

# Mutability and Aliasing

### Mutability

We say that lists are *mutable*: they can be modified. All the other types we have seen so far (str, int, float and bool) are *immutable*: they cannot be modified.

Here are several examples of lists being modified:

>>> classes = ['chem', 'bio', 'cs', 'eng']

>>>

>>> # Elements can be added:

>>> classes.append('math')

>>> classes

['chem', 'bio', 'cs', 'eng', 'math']

>>>

>>> # Elements can be replaced:

>>> classes[1] = 'soc'

>>> classes

['chem', 'soc', 'cs', 'eng', 'math']

>>>

>>> # Elements can be removed:

>>> classes.pop()

'math'

>>> classes

['chem', 'soc', 'cs', 'eng']

### Aliasing

Consider the following code:

>>> lst1 = [11, 12, 13, 14, 15, 16, 17]

>>> lst2 = lst1

>>> lst1[-1] = 18

>>> lst2

11, 12, 13, 14, 15, 16, 18]

# Function range

### Overview

Sometimes it is helpful to be able to generate a range of numbers. Python provides a builtin function range that does just this. Here are the first several lines of help(range):

class range(object)

| range([start,] stop[, step]) -> range object

|

| Returns a virtual sequence of numbers from start to stop by step.

A call on range with a single int argument produces the numbers starting a 0 and going up to, but not including, that argument.

Function range is often used with a for loop.

>>> for i in range(10):

print(i)

0

1

2

3

4

5

6

7

8

9

Function len returns the length of a string, and so range(len(s)) can be used to generate all the indices of the characters in s:

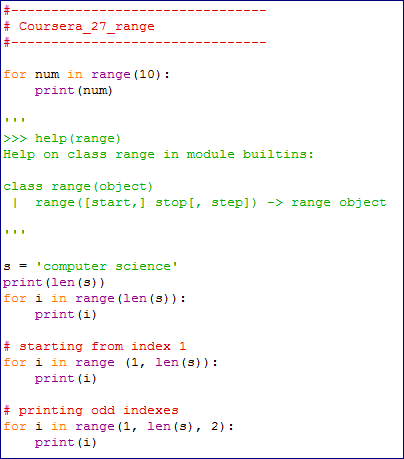
>>> s = 'computer science'

>>> len(s)

16

>>> for i in range(len(s)):

print(i)



# for loops over indices

### range

Here is the top part of the help for range:

class range(object)

| range([start,] stop[, step]) -> range object

|

| Returns a virtual sequence of numbers from start to stop by step.

# Parallel Lists and Strings

### Correspondings Elements

Two lists are parallel if they have the same length and the items at each index are somehow related. The items at the same index are said to be at corresponding positions.

Consider these two lists:

list1 = [1, 2, 3]

list2 = [2, 4, 2]

# 30. Nested Lists

Lists can contain items of any type, including other lists. These are called nested lists.

Here is an example.

>>> grades = [['Assignment 1', 80], ['Assignment 2', 90], ['Assignment 3', 70]]

>>> grades[0]

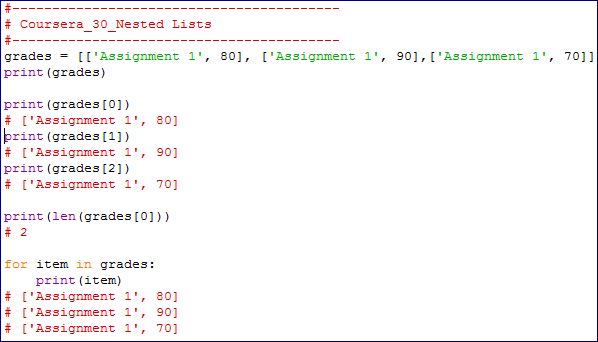
['Assignment 1', 80]

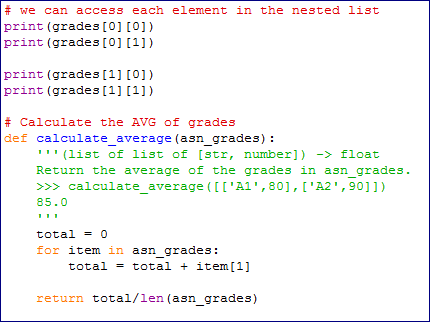
>>> grades[1]

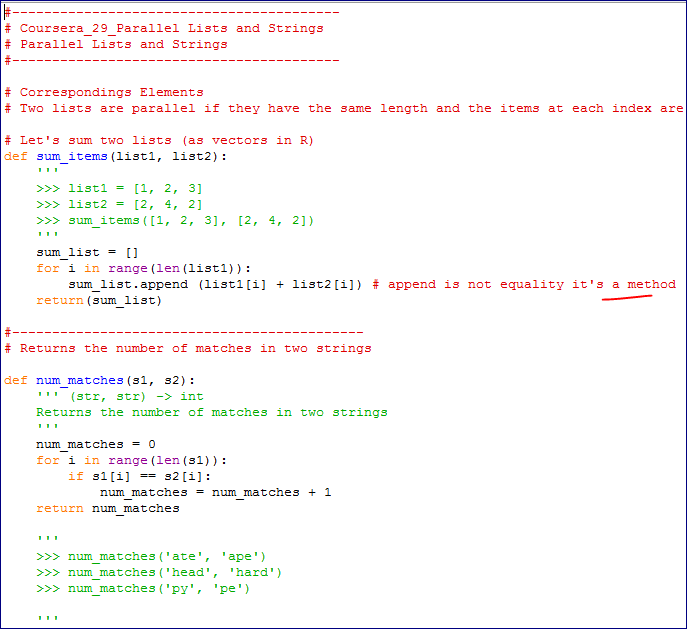
['Assignment 2', 90]

>>> grades[2]

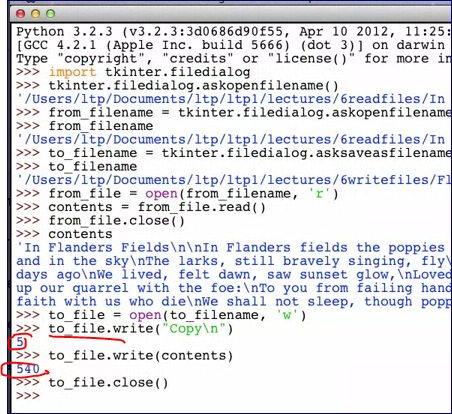
['Assignment 3', 70]







**Writing Files**



**The method returns the number of characters that were written to the file.**

**Developing a Program**

