

GEOG503A Lab 1

Use Jupyter Notebook to perform the following tasks:

Tasks:

1. Follow Exercise 1 (Learning Python language fundamentals) and type all the Python scripts as instructed. (10 pt.)
2. Write a program to prompt the user for hours and rate per hour to compute gross pay. (15 pt.)
3. Write a program which prompts the user for a Fahrenheit temperature, convert the temperature to Celsius, and print out the converted temperature. (15 pt.)

What to submit:

1. The Jupyter Notebook that contains solutions to the above tasks. File name convention for assignment submissions:
lastname_firstname_lab1.ipynb

Exercise 1

Learning Python language fundamentals

Work with numbers

Python can be used as a powerful calculator. Practicing math calculations in Python will help you not only perform these tasks, but also show you how Python works with different types of numbers.

- 1 Start ArcMap. On the Standard toolbar, click the Python button.**
- 2 In the Python window, type the following code and press ENTER:**

```
>>> 12 + 17
```

This code should give you the result 29. All basic calculator functions work just as you would expect, with one exception, which follows.

- 3 Run the following code:**

```
>>> 10 / 3
```

And the result is 3? What went wrong? The inputs to the calculation (10 and 3) are both integers, and therefore the result is, by default, also an integer. This results in rounding. If you want ordinary division, the solution is to use real numbers or floats—that is, numbers with decimals. If either one of the inputs in a division is a float, the result will also be a float.

Note: Do not type the prompt (>>>), since it is already provided by the Interactive Window.

Whenever sample code in this exercise is preceded by the prompt, it means the code should be entered in the Interactive Window.

4 Run the following code:

```
>>> 10.0 / 3.0
```

The result is 3.3333333333333335—notice that the very last number does not make sense because it has reached the limit of the number of decimal places Python uses.

Note: The exact number of decimal places depends on the version of Python you are using.

Is there a similar upper limit to the size of integers? Yes, ordinary integers cannot be larger than 2147483647 or smaller than -2147483647. However, if you want larger values, you can use a long integer, commonly referred to as "long."

5 Run the following code:

```
>>> 12345678901
```

The result is 12345678901L, with the letter L indicating that Python converted the input value to a long integer.

Basic arithmetic operations such as addition, subtraction, multiplication, and division are relatively straightforward. Many more operations are possible, but take a look at just one more for now: the exponentiation, or power, operator (*).

6 Run the following code:

```
>>> 2 ** 5
```

And the result is 32.

Although you are not very likely to use Python directly as a calculator, the examples here show you how Python handles numbers, which will be useful as you start writing scripts.

Work with strings

Now you can take a look at strings. You have already seen a very simple example, which follows.

1 Run the following code:

```
>>> print "Hello World"
```

The code prints `Hello World` to the next line. This is called a *string*, as in a string of characters. Strings are values, just as numbers are.

Python considers single and double quotation marks the same, making it possible to use quotation marks within a string.

2 Run the following code:

```
>>> print 'Let's go!'
```

The code results in a syntax error because Python does not know how to distinguish the quotation marks that mark the beginning and end of the string from the quotation marks that are part of the string—in this case, in the word “Let’s.” The solution is to mix the type of quotation marks used, with single and double quotation marks.

3 Run the following code:

```
>>> print "Let's go!"
```

The code prints `Let's go!` to the next line.

Strings are often used in geoprocessing scripts to indicate path and file names, so you will see more examples of working with strings throughout the exercise.

Strings can be manipulated in a number of ways, as you will see next.

4 Run the following code:

```
>>> z = "Alphabet Soup"
>>> print z[7]
```

The code returns the letter *t*, the seventh letter in the string where the letter *A* is located at index number 0. This system of numbering a sequence of characters in a string is called *indexing* and can be used to fetch any element within the string. The index number of the first element is 0.

5 Run the following code:

```
>>> print z[0]
```

The code returns the letter *A*. The index number of the last element depends on the length of the string itself. Instead of determining the length, negative index numbers can be used to count from the end backward.

***Note:** Quotation marks in Python are “straight up,” and there is no difference between opening quotation marks and closing quotation marks, as is common in word processors. When you type quotation marks directly in Python, they are automatically formatted properly, but be careful when copying and pasting from other documents. Quotation marks in Python have to look like this (' ') or this (" "), not like this (‘ ’) or this (“ ”).*

6 Run the following code:

```
>>> print z[-1]
```

The code returns the letter *p*.

To fetch more than one element, you can use multiple index numbers. This is known as *slicing*.

7 Run the following code:

```
>>> print z[0:8]
```

The reference `z[0:8]` returns the characters with index numbers from 0 up to, but not including, 8, and therefore the result is Alphabet.

As you have seen, you can use an index to fetch an element. You can also search for an element to obtain its index.

8 Run the following code:

```
>>> name = "Geographic Information Systems"  
>>> name.find ("Info")
```

The result is 11, the index of the letter I. In this example, `find` is a method that you can use on any string. Methods are explored later in this exercise.

Work with variables

All scripting and programming languages work with variables. A variable is basically a name that represents or refers to a value. Variables store temporary information that can be manipulated and changed throughout a script. Many programming languages require that variables be declared before they can be used. Declaring means that you first create a variable and specify what type of variable it is—and only then can you actually assign a value to that variable. In Python, you immediately assign a value to a variable (without declaring it), and from this value, Python then determines the nature of the variable. This typically saves a lot of code and is one reason why Python scripts are often much shorter than code in other programming languages.

Next, you can try a simple example using a numeric value.

1 Run the following code:

```
>>> x = 12
>>> print x
```

The value of 12 is now printed to the next line. The code line `x = 12` is called an *assignment*. The value of 12 is assigned to the variable `x`. Another way of putting this is to say that the variable `x` is bound to the value of 12. Implicitly, this particular line of code results in variable `x` being an integer, but there is no need to explicitly state this with extra code.

Once a value is assigned to a variable, you can use the variable in expressions, which you'll do next.

2 Run the following code:

```
>>> x = 12
>>> y = x / 4
>>> print y
```

The result is 3.

Variables can store many different types of data, including numbers (integers, longs, and floats), strings, lists, tuples, dictionaries, files, and many more. So far, you have seen only integers. Next, you can continue with strings.

3 Run the following code:

```
>>> k = 'This is a string'
>>> print k
```

Note: Variable names can consist of letters, digits, and underscores (_). However, a variable name cannot begin with a digit.

Work with lists

Lists are a versatile Python data type used to store a sequence of values. The values themselves can be numbers or strings.

1 Run the following code:

```
>>> w = ["Apple", "Banana", "Cantaloupe", "Durian", "Elderberry"]
>>> print w
```

This prints the contents of the list.

Lists can be manipulated using indexing and slicing techniques, very much like strings.

2 Run the following code:

```
>>> print w[0]
```

This returns Apple because the index number of the first element in the list is 0. You can use negative numbers for index positions on the right side of the list.

3 Run the following code:

```
>>> print w[-1]
```

This returns Elderberry.

Slicing methods using two index numbers can also be applied to lists, which you'll try next.

4 Run the following code:

```
>>> print w[2:-1]
```

The reference `w[2:-1]` returns the elements from index number 2 up to, but not including, -1, and therefore the result is `['Cantaloupe', 'Durian']`.

Notice the difference here between indexing and slicing. Indexing returns the value of the element, and slicing returns a new list. This is a subtle but important difference.

Use functions

A function is like a little program you can use to perform a specific action. Although you can create your own functions, Python has functions already built in, referred to as *standard functions*.

1 Run the following code:

```
>>> d = pow (2, 3)
>>> print d
```

So instead of using the exponentiation operator (`**`), you can use a power function called `pow`. Using a function this way is referred to as

calling the function. You supply the function with *parameters*, or *arguments* (in this case, 2 and 3), and it *returns* a value.

Numerous standard functions are available in Python. You can view the complete list by using the `dir(__builtins__)` statement.

Note: There are two underscores on either side of the word "builtins," not just one.

2 Run the following code:

```
>>> print dir(__builtins__)
```



```
Python
>>> print dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BufferError', 'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis',
'EnvironmentError', 'Exception', 'False', 'FloatingPointError', 'FutureWarning',
'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError',
'IndexError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
'NameError', 'None', 'NotImplemented', 'NotImplementedError', 'OSError',
'OverflowError', 'PendingDeprecationWarning', 'ReferenceError', 'RuntimeError',
'RuntimeWarning', 'StandardError', 'StopIteration', 'SyntaxError',
'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError',
'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError',
'Warning', 'WindowsError', 'ZeroDivisionError', '__debug__', '__doc__',
'__import__', '__name__', '__package__', 'abs', 'all', 'any', 'apply',
'basestring', 'bin', 'bool', 'buffer', 'bytearray', 'bytes', 'callable', 'chr',
'classmethod', 'cmp', 'coerce', 'compile', 'complex', 'copyright', 'credits',
'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'execfile', 'exit',
'file', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals',
'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'intern', 'isinstance',
'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'long', 'map', 'max',
'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print',
'property', 'quit', 'range', 'raw_input', 'reduce', 'reload', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str',
'sum', 'super', 'tuple', 'type', 'unichr', 'unicode', 'vars', 'xrange', 'zip']
>>> |
```

It may not be immediately intuitive as to what many of these functions are used for, although some are straightforward. For example, `abs` returns the absolute value of the numeric value.

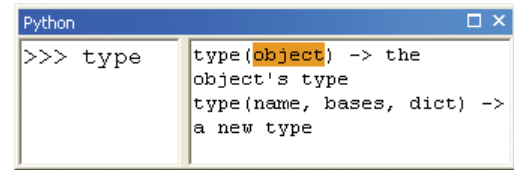
3 Run the following code:

```
>>> e = abs(-12.729)
>>> print e
```

This returns the value of 12.729.

Since it may not be immediately clear how many of these functions work and what the parameters are, it should be helpful to take a look at the Help function next.

- 4 **First, clean up the Python window by removing all the code so far. Right-click in the Python section of the window and click Clear All.**
- 5 **Make sure the Help and syntax panel is visible by dragging the divider in place.**
- 6 **Type the function `type` and don't press ENTER yet. Notice that the syntax appears in the adjacent panel. ➔**



You can find similar descriptions in the Python manuals, but having it right where you are coding in the Python window is convenient. Notice that a section of the syntax is highlighted. It specifies the parameters of the function. When you call the function, you need to supply these parameters for the function to work, although some parameters are optional.

Next, you can try out this function.

- 7 **Run the following code:**

```
>>> type(123)
```

The result is `<type 'int'>`—that is, the input value is an integer.

- 8 **Run the following code:**

```
>>> type(1.23)
```

The result is `<type 'float'>`—that is, the input value is a float, or floating point.

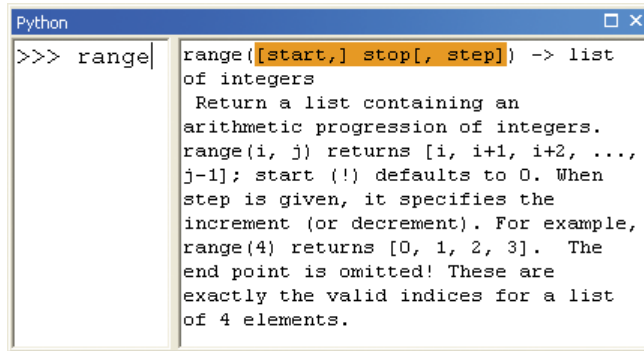
- 9 **Run the following code:**

```
>>> type("GIS")
```

The result is `<type 'str'>`—that is, the input value is a string.

Multiple parameters are separated by commas. Optional parameters are shown between square brackets `[]`. For example, take a look at the function `range`.

- 10** In the Python window, type the function `range` and notice that the syntax Help appears in the adjacent panel.



Notice the syntax: `range([start,] stop[, step])`. The function `range` has three parameters: `start`, `stop`, and `step`.

- 11** Run the following code:

```
>>> range(10, 21, 2)
```

The result is `[10, 12, 14, 16, 18, 20]`.

The function returns a list of integers from 10 to 20, with an increment of 2. However, the only required parameter is the endpoint (`stop`).

- 12** Run the following code:

```
>>> range(5)
```

The result is `[0, 1, 2, 3, 4]`.

The function returns a list of integers using the default values of 0 for the start parameter and 1 for the increment (`step`) parameter.

Use methods

Methods are similar to functions. A method is a function that is closely coupled with an object—for example, a number, a string, or a list. In general, a method is called as follows:

```
<object>.<method>(<arguments>)
```

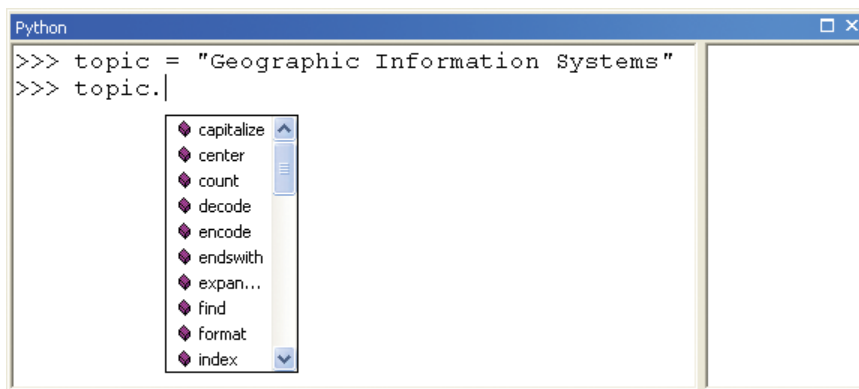
Calling a method looks just like calling a function, but now the object is placed before the method, with a dot (.) separating them. Next, take a look at a simple example.

1 Run the following code:

```
>>> topic = "Geographic Information Systems"
>>> topic.count("i")
```

The code returns the value of 2 because that is how often the letter *i* occurs in the input string.

A number of different methods are available for strings. Notice that when you start calling methods by typing a dot after the variable, a list of methods is provided for you to choose from. The syntax Help is also context sensitive.



Next, try this out by using the `split` method.

2 Run the following code:

```
>>> topic.split(" ")
```

The result is a list of the individual words in the string:

```
['Geographic', 'Information', 'Systems']
```

Next, you can see how to apply the `split` method to work with paths. Say, for example, the path to a shapefile is `c:\data\part1\final`. How would you obtain just the last part of the path?

3 Run the following code:

```
>>> path = "c:/data/part1/final"
>>> pathlist = path.split("/")
>>> lastpath = pathlist[-1]
>>> print lastpath
```

The result is `final`.

So what happened exactly? In the first line of code, the path is assigned as a string to the variable `path`. In the second line of code, the string is split into four strings, which are assigned to the list variable `pathlist`. And in the third line of code, the last string in the list with index `-1` is assigned to the string variable `lastpath`.

Methods are also available for other objects, such as lists.

4 Run the following code:

```
>>> mylist = ["A", "B", "C"]
>>> mylist.append("D")
>>> print mylist
```

The result is `['A', 'B', 'C', 'D']`.

Very few built-in methods are available for numbers, so in general, you can use the built-in functions of Python or import the `math` module (see next section) to work with numeric variables.

Use modules

Hundreds of additional functions are stored in modules. Before you can use a function, you have to import its module using the `import` function. The functions you used in the preceding sections are part of Python's built-in functions and don't need to be imported. One of the most common modules to import is the `math` module, so you'll start with that one.

1 Run the following code:

```
>>> import math
>>> h = math.floor (7.89)
>>> print h
```

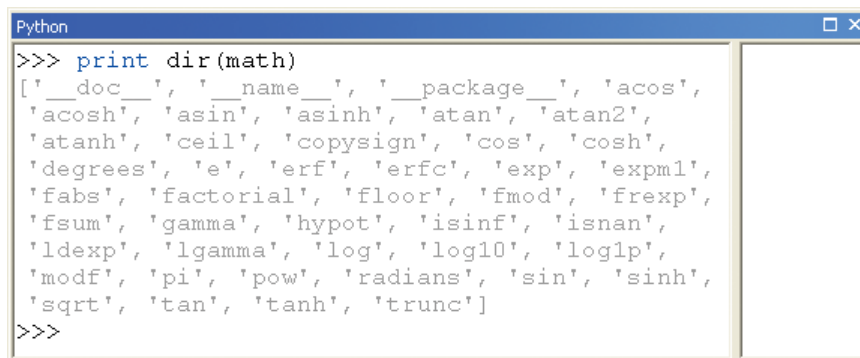
The result is 7.0.

Notice how the math module works: you import a module using `import`, and then use the functions from that module by writing `<module>.<function>`. Hence, you use `math.floor`. The `math.floor` function always rounds down, whereas the built-in `round` function rounds to the nearest integer.

You can obtain a list of all the functions in the math module using the `dir` statement.

2 Run the following code:

```
>>> print dir(math)
```

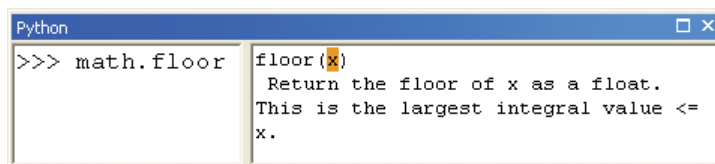


```
Python
>>> print dir(math)
['__doc__', '__name__', '__package__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',
'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'hypot', 'isinf', 'isnan',
'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
>>>
```

You can learn about each function in the Python manuals, but remember that you can also see the syntax in the Python window's Help and syntax panel.

3 Type the following code and do not press ENTER:

```
>>> math.floor
```



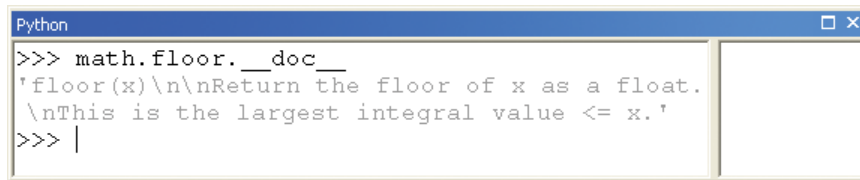
```
Python
>>> math.floor
floor(x)
Return the floor of x as a float.
This is the largest integral value <=
x.
```

Another way to see the documentation is to use the `__doc__` statement directly in Python.

4 Run the following code:

```
>>> print math.floor.__doc__
```

The result is a printout of the same syntax Help but now directly within the interactive Python interpreter.

A screenshot of a Python interpreter window titled "Python". The window has a blue title bar with standard window controls. The main area is a text editor with a light gray background. It shows the command `>>> math.floor.__doc__` followed by the output: `'floor(x)\n\nReturn the floor of x as a float.\n\nThis is the largest integral value <= x.'`. The cursor is at the end of the command line.

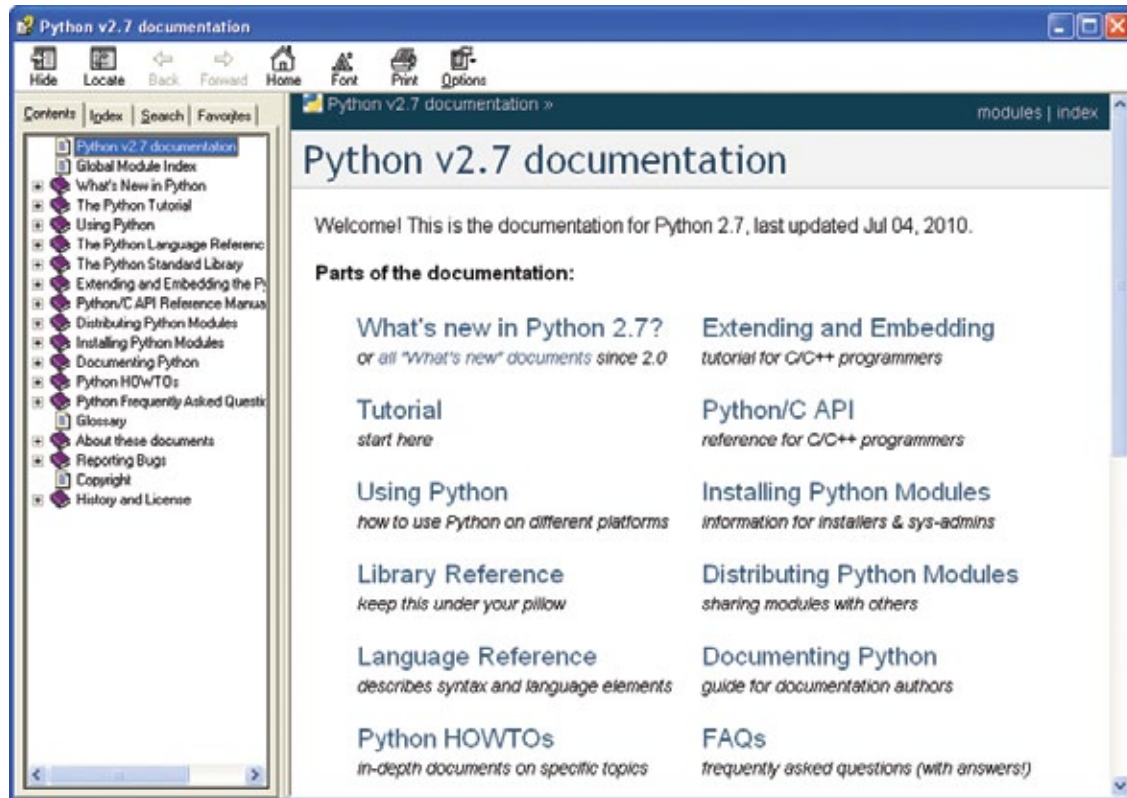
```
>>> math.floor.__doc__
'floor(x)\n\nReturn the floor of x as a float.
\nThis is the largest integral value <= x.'
```

The syntax is `floor(x)`, which means the only parameter of this function is a single value. The function returns a float (such as 7.0), not an integer (such as 7). This information allows you to determine whether the function is really what you are looking for and how to use it correctly.

There are numerous modules available in Python. A complete list can be found in the Python manuals, which you'll look at next.

***Note:** Remember that you need to add a prefix to any non-built-in functions using the name of the module, as in `math.floor(x)`, not just `floor(x)`. If you do not use a prefix, you will get an error stating that the name `floor` is not defined.*

- 5 To access the Help documentation, on the taskbar, click the Start button, and then, on the Start menu, click All Programs > ArcGIS > Python 2.7 > Python Manuals.



- 6 In the documentation table of contents, under “Indices and tables,” click Global Module Index. The index provides an alphabetical list of all the available modules. ➔

Global Module Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Z

<code>__builtin__</code>	The module that provides the built-in namespace.
<code>__future__</code>	Future statement definitions
<code>__main__</code>	The environment where the top-level script is run.
<code>_winreg (Windows)</code>	Routines and objects for manipulating the Windows registry.

A

<code>abc</code>	Abstract base classes according to PEP 3119.
<code>aepack (Mac)</code>	Deprecated: Conversion between Python variables and AppleEvent data containers.
<code>aetools (Mac)</code>	Deprecated: Basic support for sending Apple Events
<code>aetypes (Mac)</code>	Deprecated: Python representation of the Apple Event Object Model.
<code>aifc</code>	Read and write audio files in AIFF or AIFC format.