Exercise 1: Python Language Basics

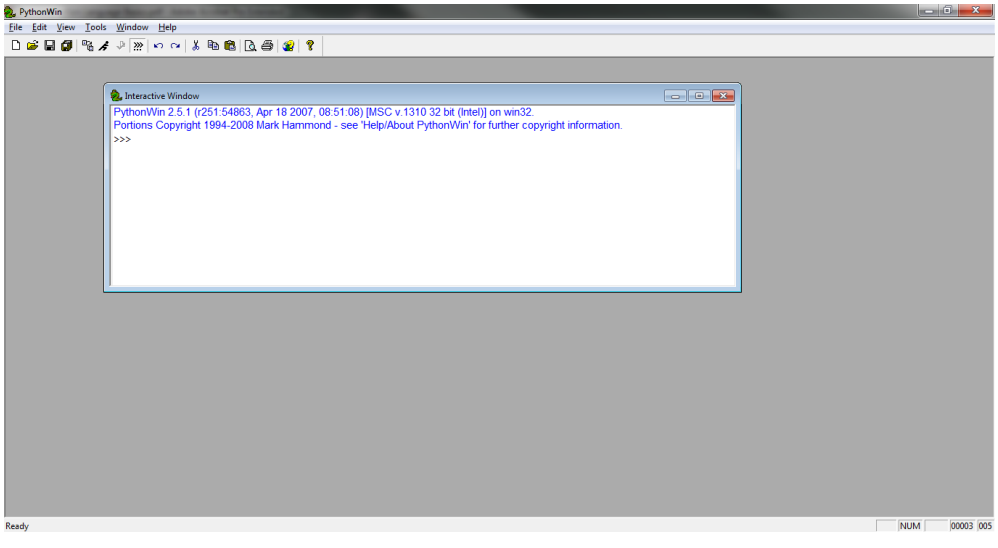
In this exercise we will cover the basic principles of the Python language. All languages have a standard set of functionality including the ability to comment code, define and use variables, makes decisions in code, loop through code blocks, import class libraries, and many others. We will cover each of these basic functional attributes of the Python language in detail during this exercise.

Step 1: Create a new Python script

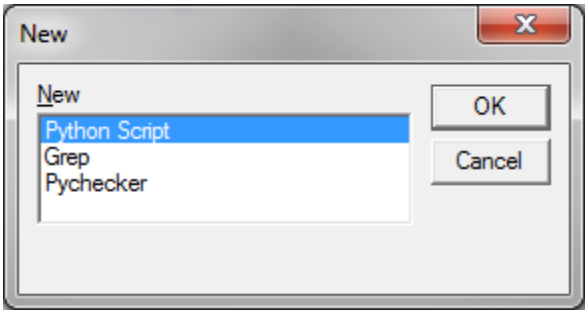
In this step you will learn how to create a new, empty Python script that you can use to write code.

- Click Start →Programs →Python 2.5.1 → PythonWin

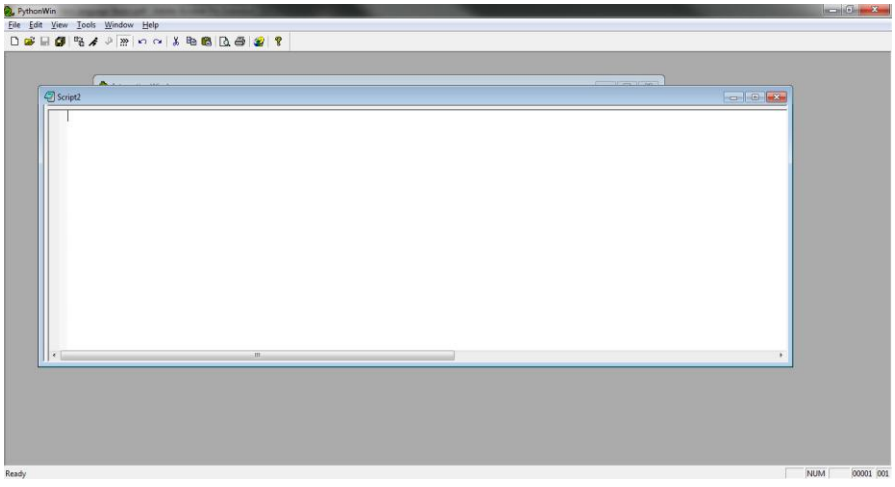
  The PythonWin Interactive Window is automatically opened as you see below. The Interactive Window provides you with a way of executing and testing individual lines of code, outputs error messages, and allows you to display output from your code via the print statement.



- Select File →New from the PythonWin menu. The, select Python Script from the New window.



A new script window will be created. Python code you write that needs to be **saved** should be placed in this new window. If you simply need to write and test a few lines of code you can use the Interactive Window.



Step 2: Commenting Python code
Python and all other programming languages provide a way to document the code that you write. Documentation serves an important purpose in detailing what your script does, who wrote the script, when it was last modified, and generally makes your code easier to understand. Commented lines of code are ignored by the Python interpreter, and serve strictly as documentation.

- In the script window, type the following lines of code.

  # Name: <your name>
  # Date: <current date>
  # Purpose: This script will be used to test the basic functionality provided by the
  # Python language

  All commented lines of code in Python start with the # or ## symbol. By default these lines of code are colored green and italicized in PythonWin. Python will ignore any line of code that begins with the # symbol.

- Click File → Save from the PythonWin menu.

- In the Save As dialog, navigate to the c:\data\Exercises directory and save the file as IntroToPyth.py.

Step 3: Variables
You can think of variables as named units of storage that your program uses to store information. Various types of data can be stored in variables including strings, numbers, lists, and other data types. Unlike many other languages, Python does not require that your variables be declared before use. What this means is that you don't have to tell Python what type of data a variable will represent. You simply give the variable a name, assign data to the variable, and start using the variable in your code. In this step you will work with various types of data in your script.

Step 3A: String variables
First, you will assign a string value to a variable. When assigning a string value to a variable you must enclose the string with single or double quotes. The quotes tell Python that we are assigning a string

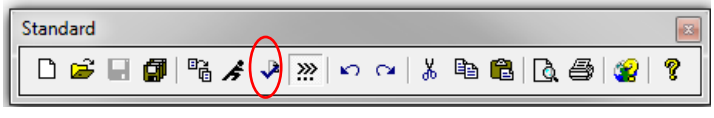
value to the variable.

- In the IntroToPyth.py window, type the following code under the comments.

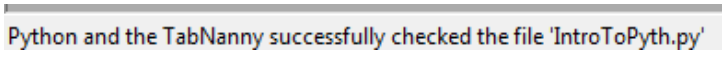
```
myName = “John Doe”
print myName
```

For this script, myName is a name that we have attached to a variable that will hold information. We then assign a value to the myName variable by using the “=” sign along with the string value “John Doe”.

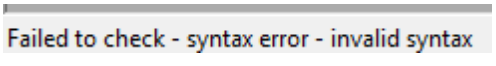
- Before running this or any other Python script you‟ll want to check for any syntax errors in your code. Click the “Check” button to check for syntax errors.



If your code is free of syntax errors the following message should be printed at the bottom of the PythonWin window.

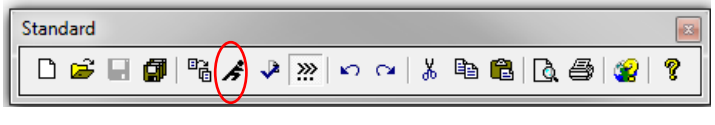


In the event that your script contains syntax errors you will see the following error message.

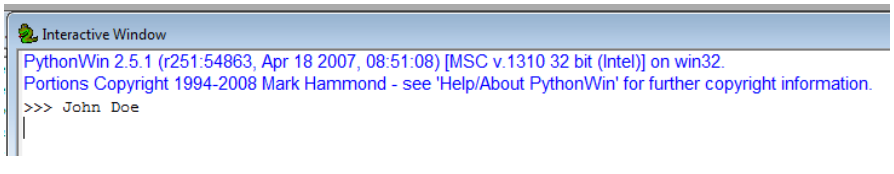


You‟ll need to check your script for errors if you receive this error message and run the check again iteratively until all syntax errors have been removed.

- Once your script is free of syntax errors you can run the script by clicking the “Run” button.



The value of the myName variable, “John Doe” will be printed to the Interactive Window as seen below. The print statement sends output to the window.



String variables can be manipulated in a number of ways. In the next example you will see an example of how strings can be concatenated or joined.

- In the IntroToPyth.py window, highlight and delete the two existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
a = “Hello”
b = “ World”
c = a + b
print c
```

- Run the script.

The text string “Hello World” is printed to the Interactive Window. When working with string variables in Python, the “+” symbol is used to concatenate two variables and returns the value to a new variable. In this case the value of a + b is stored in the variable c.

Strings can be manipulated in many other ways. In the lecture we covered the concept of string indexing and slicing. Since Python strings are an ordered collection of characters we can access the characters by position to pull out a single character or a series of characters. In this next example, we will take a look at how strings can be sliced to return a series of characters.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
theString = “Floodplain.shp”
theLayer = theString[0:10]
print theLayer
```

- Run the script.

The Interactive Window will print out the string “Floodplain”. What we’ve done in this script is take an existing variable (theString) containing the text “Floodplain.shp”, and slice it into a new string. Slicing is accomplished by taking a string and providing two offsets contained within square brackets and divided by a colon ([0:10]). This will return the first offset up to, but not including, the tenth character. This is an important and sometimes confusing distinction to make. You need to remember that the second offset, in this case the integer value 10, tells Python to get all characters up to, but not including the tenth character which is a “.” character.

Step 3B: Numeric variables

In the next example you will work with numeric variables. Unlike string variables, there is no need to enclose numeric values with quotes unless you intend to treat them as strings.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
a = 10
b = 20
```

```
c = a + b
print c
```

- Run the script.

The Interactive Window will print out a value of 30. You will notice some similarities and differences between using string and numeric variables. Similarities include the concept of dynamic typing. There is no need to tell Python the type of data that we will be assigning to the variable before putting it to use. We simply name and assign a value to the variable in a single step. We let Python determine the data type. As we mentioned, when assigning values to numeric variables, there is no need to enclose the value in quotes. The existence or non-existence of quotes is what tells Python the data type of our variable.

In addition, notice that just as in our string example, we use the line c = a + b. Python is smart enough to determine that when we are working with numeric data, it should add the two values, and when we're working with string variables it should concatenate the variables.

Step 3C: List Variables

In addition to strings and numbers, Python variables can also hold lists. From the lecture you will remember that lists are ordered collections of objects. These objects can be any type of data including strings, numbers, other lists, and other objects. Similar to strings, lists can be accessed by offset similar to what you saw with string indexing and slicing. Lists can also dynamically grow and shrink.

List variables are created by assigning a series of values, separated by commas, and embedded in brackets.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
fcList = ["Hydrants", "Water Mains", "Valves", "Wells"]
print fcList[0]
print fcList[3]
```

- Run the script.

The Interactive Window will print out the values "Hydrants", and "Wells". fcList[0] returns the first item from the list of values, while fcList[3] returns the fourth item from the list. This is similar to indexing a string. Lists can also be sliced to return a new list of multiple values from a list. For example, fcList[0:1] would return a list containing the values "Hydrants", and "Water Mains"

In addition to indexing and slicing, lists can be manipulated in a number of ways. Basic list operations include getting the number of items in a list, concatenation of lists, and repetition of lists. Furthermore, the contents of a list can be changed "in place". Changing a list value "in place" simply means that Python does not have to create a new copy of the list variable when a change takes place. In addition, list functions such as append and sort can be used to add values to a list and sort the values currently in a list.

In the next code example we will examine several of these list features including appending a value to an

existing list, sorting a list, and using the built in Len function to get the number of values in a list.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code. After pressing the ENTER KEY after the colon in the for statement the cursor should jump to the beginning of an already inner indented position within the for…loop

```
fcList = [“Water Mains”, “Hydrants”, “Wells”, “Valves”]
fcList.append(“Blowoff Valves”)
fcList.sort()

print len(fcList)

for lyr in fcList:
        print lyr
```

- Run the script.

This script highlights several of the built in features that Python provides for working with lists. In this script, we first create a new list object containing four string values that could represent geographic layers. We then use the append function to add a new string value to the list. Next, we use the sort function to sort the contents of the list alphabetically. We then use the len function to get the size of the list and print it out to the Interactive Window. Finally, we loop through each item in the list and print out the value. This type of script would be useful if you had a need to perform the same function on a list of geographic layers. For instance, perhaps you need to add a common field to each of these layers. You could create a list of layers and then loop through each layer, performing the AddField tool functionality for each layer.

Step 3D: Dictionary Variables

Dictionaries are similar to lists in that they have the capability of storing multiple objects within a single structure. However, unlike lists, dictionaries store objects as an **unordered** collection of objects that are accessed through the use of a key value. Like lists, dictionaries have the capability of growing, shrinking, and being changed in place. Dictionaries are composed of a set of key:value pairs, separated by commas and enclosed by curly braces.

Example: dictLayers = {„Roads‟:0, „Airports‟:1, „Railroads‟:2}

In this example, you will create a dictionary object and access a value in the dictionary by key.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
dictLayers = {“Roads”:0, “Airports”:1, “Railroads”:2}
print dictLayers[“Railroads”]
```

- Run the script.

This script creates a new dictionary object named dictLayers and populates is with three key value pairs. In each case the first value specified (Roads, Airports, Railroads) in each pair is known as the key. The second value of the pair (0,1,2) is the value associated with the key. Values are retrieved using the key as in this case when we specify that we‟d like the value associated with the „Railroads‟ key. The script will print out a value of 2.

Dictionaries have a keys() method that can be used to iterate through the entire collection of key:value pairs. In this example you will use the keys() method to loop through each value in a dictionary.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
dictLayers = {“Roads”:0, “Airports”:1, “Railroads”:2}
listLayers = dictLayers.keys()

for lyr in listLayers:
        print lyr
```

- Run the script.

Step 4: Decision Support Statements

The primary decision making statement in Python is the “if … elif … else” statement. Decision statements enable you to control the flow of your programs, and make decisions based on a boolean test condition. Boolean values can be either true or false. The test condition must evaluate to a “True” value before the indented block of code immediately following the statement will execute.

In this step you will write a simple script that uses the “if…elif…else” decision support statement to branch your code based on a true/false condition.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
listLayers = ["Roads","Airports","Railroads"]

fc = listLayers[1]

if fc == 'Roads':
        print "Found the Roads layer"
elif fc == 'Airports':
        print "Found the Airports layer"
else:
        print "Found the Railroads layer"
```

- Run the script.

The text “Found the Airports layer” will print to the Interactive Window.

Step 5: Looping Statements

Python provides two looping structures that enable your code to repeat lines of code. While loops allows your program to repeat lines of code over and over as necessary, repeatedly executing a block of statements while a test at the top of the loop evaluates to true. For loops execute a block of statements a predetermined number of times.

In this step you will write a somewhat more complex script that demonstrates how a For Loop functions. We will also incorporate the use of dictionaries and if statements to demonstrate how each of these programming constructs can be used in conjunction.

- In the IntroToPyth.py window, highlight and delete the existing lines of code.
- In the IntroToPyth.py window, type the following lines of code.

```
dictLayers = {"Roads":"Polyline","Rail":"Polyline","Parks":"Polygon","Schools":"Point"}

listLayers = dictLayers.keys()

for lyr in listLayers:
    if dictLayers[lyr] == "Polyline":
        print lyr
```

- Run the script.

The script will print out the Rail and Roads values. The For loop enables you to execute the If decision support statement once for each value found in the dictLayers dictionary.

When a value equals “Polyline” the dictionary key is printed to the Interactive Window. A script such as this would be useful when you have a list of feature classes that need to be tested for feature type. You could then branch your code to perform a unique procedure based on the type of feature class.

Step 6: Accessing Modules

While Python comes with a built-in set of functions, you will frequently need to access specific bundles of functionality stored in modules. For instance, the Math module stores specific functions related to processing numeric value. The win32com.client module provides functionality that allows Python to access Windows COM objects such as those found in ArcObjects. The import statement is used to provide access to these external modules.

The first module you will import in this exercise is the Math module.

- In the **Interactive Window**, type the following line of code.

```
import math
```

- Press Enter

Python has now imported the math module. You can obtain a list of all the functions provided by the math module. Type the following in the Interactive Window.

- dir(math)

Each of the functions provided by the math module will be displayed, separated by commas.

You can obtain the syntax for an individual function by using the __doc__ statement. That's two underscores followed by the "doc" text followed by two more underscores.

To find the syntax for the floor function, type the following in the Interactive Window.

- print math.floor.__doc__

Python will print out the syntax as well as a brief description of what functionality the function provides.

Code completion for each of the available math functions will also be available. In the Interactive Window type the following.

- math.

A list box containing each of the available math module functions will appear.

- Double-click the floor function.

Based on the syntax you obtained using the __doc__ statement for the floor function you can determine that the floor function takes a single numeric argument. Enter a value of 102.4 as the argument to floor. Your statement in Python‟s Interactive Window should now look as follows.

- math.floor(102.4)

Python prints out a value of 102.0 which is consistent with the functionality provided with the floor function.