CSC110 Supplemental Reading: Week01

Contents

1 Working with IDLE 2

1.1 Trying IDLE 2

1.2 The print Statement 2

1.3 Comments 3

1.4 Uses for Interactive Python 4

2 Creating a Python Script 5

3 Variables, and Data Types 7

3.1 Variables 7

3.2 Identifiers 7

3.3 Data Types 7

4 Input and Output, Using Variables 8

4.1 Function 8

4.2 String Input 8

4.3 String Output 8

5 Getting Numerical Input 10

1 Working with IDLE

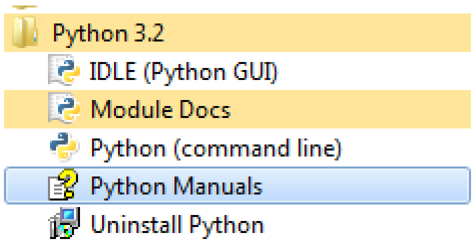
IDLE is the Integrated DeveLopment Environment that comes with the standard Python distribution from http://www.python.org.

IDLE is an interactive environment. You can type Python commands into IDLE and they will be performed immediately. This is possible because Python is an interpreted language. The prompt for IDLE is >>>.

1.1 Trying IDLE

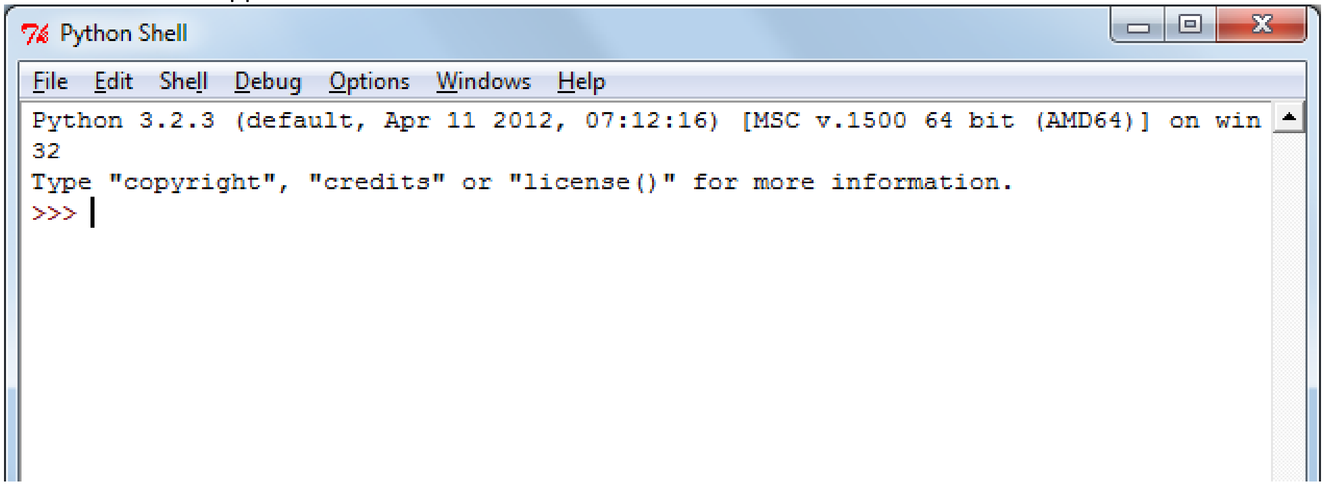
Let's try it.

1. Run IDLE by selecting IDLE (Python GUI).



   (This images assumes that you have installed Python 3.2 on Windows.)

2. The IDLE window appears.



```
Python 3.2.3 (default, Apr 11 2012, 07:12:16) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

3. Notice the prompt: >>>
4. You can now type in commands. Let's start by just asking Python to do some arithmetic calculations for us.
5. At the >>> prompt, type **5 + 3** and press Enter.

   Python performs the calculation, displays the answer, and prompts for the next command.
6. You can try other operations, like subtraction, multiplication, and division using the hyphen (-), asterisk (*), and slash (/) characters.

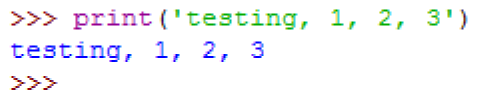
This is nice, but it's not quite Python yet. Our first Python command is the **print** statement.

1.2 The print Statement

The print statement prints out strings. It begins with the keyword **print,** which is a **built-in function**. Following the keyword is the string that we want to print out. A string is a bunch of characters enclosed within quote marks. Officially, Python uses apostrophes (single quotes - '), but quotation marks (double quotes - ") are also accepted by Python.

Let's try it

1. At the IDLE prompt, type **print('testing, 1, 2, 3')** and press Enter.

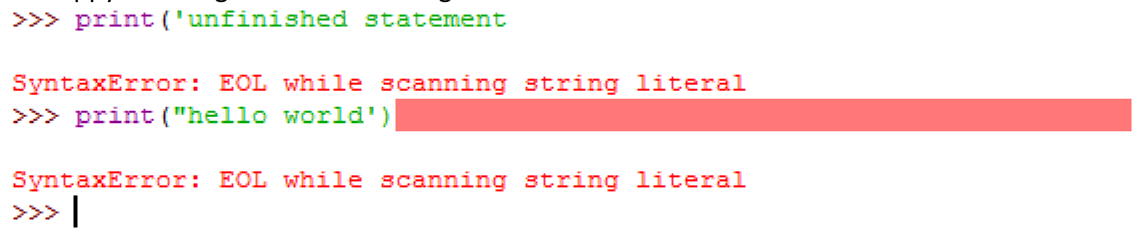
   

   ```
   >>> print('testing, 1, 2, 3')
   testing, 1, 2, 3
   >>>
   ```

   Python then prints out the contents of the string, that is the stuff within the quote marks.
2. Try it again, typing very slowly.
   a. Notice that the letters start out black.

   When you complete the keyword print, they become orange. This is syntax coloring. It's IDLE's way of showing you that you have typed a keyword.

   b. Notice that if you type the keyword incorrectly, for example using a capital **P**, it does not turn orange. This is because Python is *case-sensitive*. Upper- and lower-case letters are different.
   c. The characters that you type after the quote mark are part of a string. IDLE shows strings in green.
   d. If you don't put the closing quote mark (or if you mix single and double quotes), Python will be "unhappy". You'll get an error message.

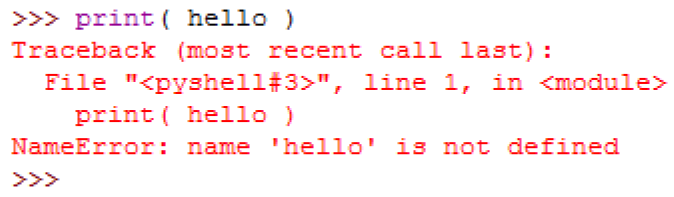
      

      ```
      >>> print('unfinished statement

      SyntaxError: EOL while scanning string literal
      >>> print("hello world')

      SyntaxError: EOL while scanning string literal
      >>> 
      ```

      The error is a SyntaxError, as indicated by the first part of the error message, just before the colon (:). IDLE also indicates where it thinks the error is by highlighting it in red. This is a guess about where the error is, but often it's a good guess.
   e. You'll get a different error if you completely omit the quote marks.

      ```
      >>> print( hello )
      Traceback (most recent call last):
        File "<pyshell#3>", line 1, in <module>
          print( hello )
      NameError: name 'hello' is not defined
      >>>
      ```

      What is the name of the error in this case?

1.3 Comments

Comments are parts of the program that are ignored by the computer. They are for "human consumption". That is, the comments are notes that are included in a program to explain what's going on to other programmers. These notes can be very helpful when changes or corrections need to be made to the program. This even applies to the original programmer as well. It's easy to forget what you were thinking when you wrote some code, particularly when you need to return to it later.

In Python, comments start out with a pound sign (#). They continue to the end of the line.

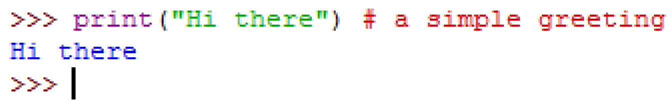
When you type comments in IDLE, they don't do anything.

Let's try it.

1. At the IDLE prompt, type in a comment, starting with a pound sign.

```
>>>
>>> # This is my first Python comment
>>>
```

2. You can also type comments at the end of a line.



```
>>> print("Hi there") # a simple greeting
Hi there
>>> 
```

I discourage this practice, because the comments can be difficult to find.

1.4 Uses for Interactive Python

Working interactively in IDLE is a good way to test out new techniques. However, anything you type into IDLE like this is going to be gone when you shut IDLE down.

We can use these same Python commands in script files. Now, the commands will be saved in a file and can be run whenever we would like.

2 Creating a Python Script

We have worked interactively in IDLE a bit. We have seen the **print** statement, which uses the **print**() function. We have also seen how to make comments in Python.

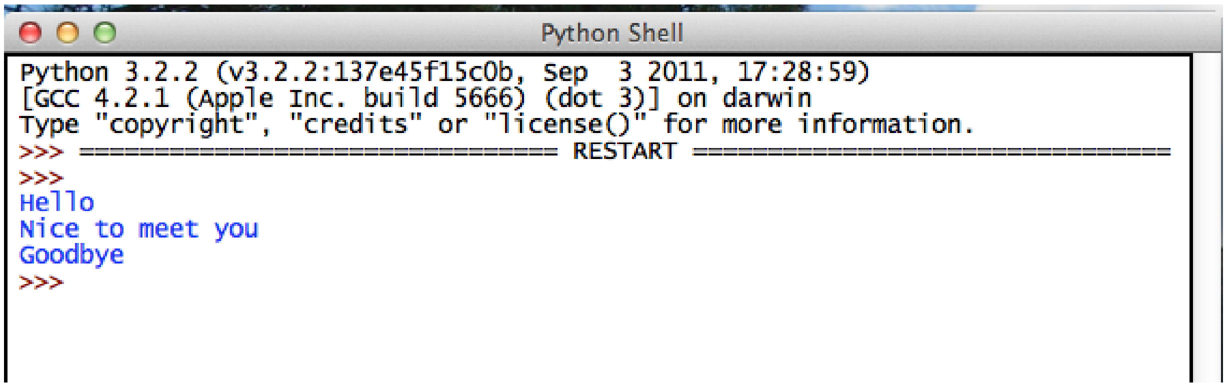
Let's use those now to create our first Python script.

1. In the IDLE window, from the File menu, select New Window.
   A new window appears. This is where we will write our Python script.
2. In the new window, type the following Python commands:

```
print ( 'Hello' )
print ( 'Nice to meet you' )
print ( 'Goodbye' )
```

   Notice that the IDLE prompt >>> is not included in the Python script file.

3. Save the file. From the File menu, select Save. It will prompt you for a file name. Name the file **Greeting.py**. The .py file extension shows that this is a Python script file. Notice that the file name appears in the title bar of the script window.
4. Run the file. From the **Run menu**, select Run Module. This will run the script file. The output will appear in the IDLE window.

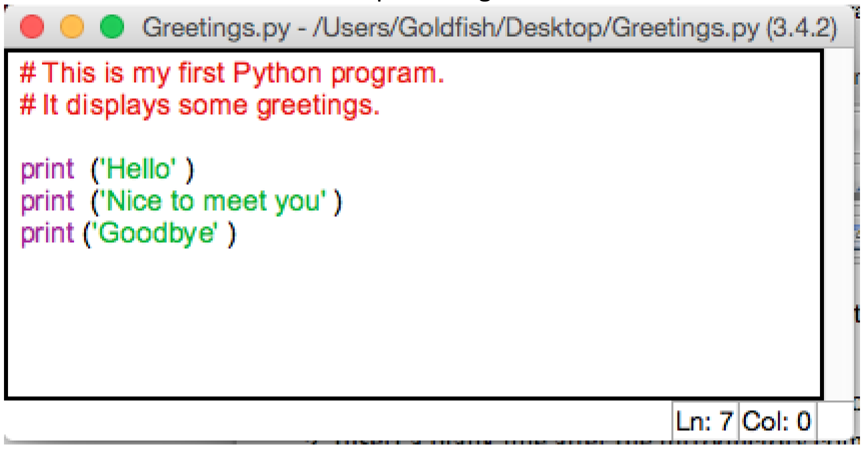


```
Python 3.2.2 (v3.2.2:137e45f15c0b, Sep  3 2011, 17:28:59)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello
Nice to meet you
Goodbye
>>>
```

   This starts out by "restarting" the IDLE environment. We'll discuss this later and find out what it means when we talk about variables.

We're almost done. The program really isn't complete until we add some comments to the code.

1. At the top of the file, insert some comments that briefly describe the purpose of this script. Remember, each comment line starts out with a pound sign.



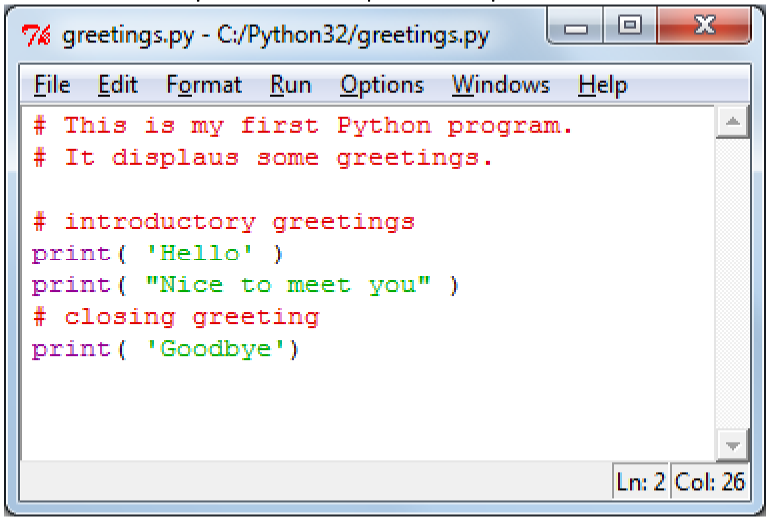
Greetings.py - /Users/Goldfish/Desktop/Greetings.py (3.4.2)

```
# This is my first Python program.
# It displays some greetings.

print  ('Hello' )
print  ('Nice to meet you' )
print ('Goodbye' )
```

Ln: 7 Col: 0

   This kind of introductory comment block should be included in each of your script files. It will be part of the requirements for all of the scripts you turn in this quarter.
2. Insert a blank line after the introductory comment block to set it apart from the rest of the script.
3. Insert comments within the code, explaining what the different parts of the code do.
4. Here is an example of the completed script.



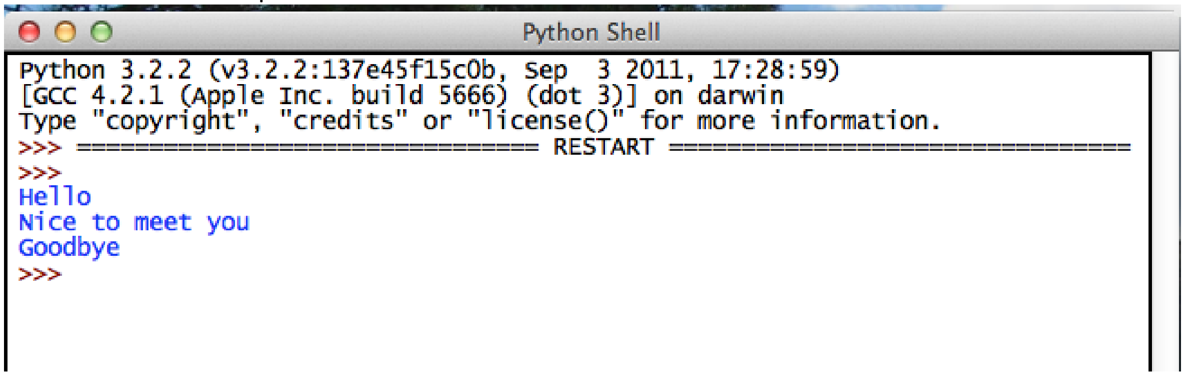
greetings.py - C:/Python32/greetings.py

```
# This is my first Python program.
# It displaus some greetings.

# introductory greetings
print( 'Hello' )
print( "Nice to meet you" )
# closing greeting
print( 'Goodbye')
```

Ln: 2 Col: 26

   Notice that IDLE marks the modified (and unsaved) file with an asterisk at the end of the filename in the title bar.
5. Save your changes. (The asterisk disappears.)
6. Run the modified script. The actions should be the same.



Python Shell

```
Python 3.2.2 (v3.2.2:137e45f15c0b, Sep  3 2011, 17:28:59)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello
Nice to meet you
Goodbye
>>>
```

Congrats! You have just created your first Python script.

3 Variables, and Data Types

3.1 Variables

A *variable* is a named container. It can hold a single value. In Python, assigning a value to variable will create the variable. There is no need to declare the variables separately. Here is an example of an *assignment* statement:

```
phrase = 'Hello'
```

The assignment statement has three parts:

variable = *value*

The variable in the example is named phrase. The value is the string value, 'Hello'.

3.2 Identifiers

The variable name needs to be a valid *identifier* in Python. Identifiers are used to name all of the resources in Python. There are some requirements for identifiers:

1. begin with a letter
2. following the initial letter, only letters and digits
3. not a keyword of Python

In Python, there are 53 letters; 26 upper-case letters, 26 lower-case letters, and the underscore. The upper-case and lower-case letters are not equivalent. This is another way of saying that Python is *case-sensitive*.

Keywords are words (letter sequences) that have special meaning within Python. We have already seen one of these keywords, **print**. A list of the keywords of Python can be found on page 18 of the textbook. In IDLE, the keywords are colored orange (or purple) so they are easy to identify.

3.3 Data Types

The values in assignment statements can be one of several different *data types*. Let's begin by discussing two numeric types, *int* and *float*. The *int* data type represents integers. The *float* data type represents floating-point, or real numbers. When you type in numbers, if you include a decimal point, the value is a *float*. If it consists only of digits (possibly with an initial negative sign), it is an *int*. The names *int* and *float* are the ones used by Python to refer to these two data types. You are familiar with constants of these types from math. Some examples of integers are 4, -1, 0. Float examples are 3.14, -17.0, 2.2222.

Another important data type in Python, the *string*. A **string literal**, or string constant, is a sequence of zero or more characters enclosed within quotes, either single quotes or double quotes. The Python name for strings is *str*.

4 Input and Output, Using Variables

Recall from the text that the classic programming paradigm is

Input → **Processing** → **Output**

We already were acquainted with basic output using the print statement. Now we'll learn about input. Input is getting information into the program. Right now, we'll take a look at how the program can gather information from the user, the person sitting at the mouse and keyboard.

4.1 Function

There is a *built-in function* that we can use to gather data from the user and the keyboard. It is called *built-in*, because it come as a standard part of Python. It is a *function*, because, like in algebra, they always behave the same way when given the same values to work with.

This is all a little abstract, so let's take a look at an example (with a little simplification). The function we'll look at calculates the square root. The name of this function is **sqrt()**. To use the function, we need to call it, using the parentheses ( ). The *argument* for this function is the value whose square root we want. Just like in algebraic notation, we put the value within the parentheses.

```
answer = sqrt(4)
```

Notice that the equal sign here is *not* like in algebra. It does not assert equality of the variable and the expression; instead, it causes assignment of a value of the expression to the variable. The value 2 is stored in the variable answer.

4.2 String Input

To get a input from the user, we use the function **input()**. The **input**() function has an optional argument, the prompt to display to the user, telling them what the program is expecting. This function returns a string that contains whatever characters the user typed. It can be one letter or a paragraph of text. It will return everything that the user types up to (but not including) the newline (when the user presses the return key).

Here is a simple program that uses the **input** function:

```
# example that asks the user's favorite color

# ask for the favorite color
favColor = input('What is your favorite color? ')

# use the value.
# notice the use of commas in the print statement
print ( 'You said that', favColor, 'is your favorite color.' )
print ( 'My favorite color is green.' )
```

4.3 String Output

Look closely at the output section of the example above. The **print** statement can also be used with variables and/or combinations of variables and string literals.

```
print ( 'You said that', favColor, 'is your favorite color.' )
```

Notice that first it prints a string literal (the phrase 'You said that') then it prints the contents of the favColor, and final the second string literal.

As you read in the textbook, you can have the **print** statement print out multiple values by giving a list of values separated by commas to the **print** statement. The **print** statement will print out the values separated by a single space.

When the **print()** has no arguments inside the parentheses, it prints a blank line.

5 Getting Numerical Input

All user input that comes from the keyboard into a Python program is string data. That is, it is treated as a **string of characters**. What happens when a program executes this statement?

```
userInput = input("Enter something here")
```

Here are some examples that a user could enter in response to the input statement above:

```
Joe
Today is Monday.
I have 3 brothers and 1 sister.
18.4122
```

The last example is very important. The computer will store it as a string of characters, each character being a digit or the '.' symbol.

But a string of characters cannot be using in a calculation! So the programmer must convert the string of digit characters to a numerical format.

Floating Point Values

The built-in **float()** function will convert any valid number from a string of digits to a numerical format. It is best with floating point values, such as:

3.1289

-44.0789

6.02×10^{23} (which you can enter as 6.02E23 or 6.02e23)

The best way to use this function is at the time a user is expected to enter a number. This requires us to use **nested functions:**

```
length = float( input( 'Enter the exact length of the box in meters' ) );
```

The variable length will now contain a floating point number.

Integer Values

The built-in **int()** function will convert any valid ***whole number*** from a string of digits to a numerical format. It is best with integer values, such as:

```
6
-12
89323
```

The best way to use this function is at the time a user is expected to enter a number. This requires us to use **nested functions:**

```
age= int( input( 'Enter your age in years' ) );
```

The variable length will now contain an integer. Any following '.' or decimal digits will be ignored.