Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

Chapter 3: Variables and Types

3.1. What is Programming

Before jumping into details and specifics, first consider the idea and goals of programming. A program is a collection of statements that together can perform one or more tasks. Programs can collect and use information as input and can generate information as output. This input and output can be read from and written to the command console, or can utilize? other files, databases, etc. Programs can be designed to use information to make logical decisions and can also repeat, or loop, through steps for a given duration. Programs can perform highly complicated calculations and can generate a variety of results, including music and graphics. Programs are reusable and can also be shared, extended, and built upon to increase their versatility, power, and complexity.

A program also acts as a method for humans to communicate with machines. In fact, programs are translated or compiled into machine language (and then to binary) so they can be executed. As such, each high level programing language that humans use, such as Python, R, C++, or Java, has its own rules, reserved words, syntax, and grammar that the translators (compiler for C++) can recognize and convert to machine language. As programmers (coders), we must follow the rules, the syntax, and grammar for the language we use. Therefore, the next step in programming, after setting up a functional programming environment (see Chapter 1) is to begin to learn the rules, syntax, and grammar of the language.

If you are new to programming, the process of developing functional code and learning all of the rules, syntax, grammar, and available tools and packages can seem daunting. Programming projects, like all projects, are best built in small and organized steps. It is a good idea to plan out the flow of the programming project using a flowchart. Flowcharts will be discussed throughout this book and in the next chapter. While learning the rules, syntax, and grammar of a new programming language is required, only practice and perseverance will improve programming skills. This is synonymous with the idea that learning a new language, which is not your native language, does not immediately make you a great writer.

An algorithm is a set of predefined steps, that if taken, will accomplish a task. Algorithm development and analysis are utilized throughout all areas of programming and computer science. For example, if a program is developed to sort a given dataset, the program must use an algorithm to do this. There are often many possible algorithms that can be used to accomplish a task. Sorting, for example, can be accomplished through already-created algorithms such as MergeSort, QuickSort, or HeapSort. Alternatively, a new sorting method can be created; perhaps one most applicable to the type and size of the data being sorted. Some sorting algorithms are faster or more space conservative than others. The study of the speed and space usage of algorithms is called, the *Analysis of Algorithms*. While algorithm analysis is not the topic of this book, it is an important consideration for larger projects when space, time, and other resources are a concern. And now for some examples to start with.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

3.2. Two Preliminary Examples

This section will introduce Python 3 with two preliminary practice examples. As you review each example, type the example into a new .py Python file in the Spyder IDE (or another IDE). Save and run each example. Make small changes to each example to see what these changes do.

As a reminder, this book uses Spyder (via Anaconda) as the Python 3 IDE. From this point, all references to the IDE, .py Python files, or to the command console will refer to Spyder specifically (and can be simulated in any Python 3 IDE).

Example 3.2.1: Write a program that takes a User's name as input and prints Hello Username.

```
# HelloUser.py
# Ami Gates
# This program takes a User's name as input
# and says Hello User as output

def main():
   name = input("What is your name?")
   print("Hello", name)

main()
```

Exercise 3.2.1: Saving and Running Code Using a .py Python 3 file

To complete this exercise, take these steps.

1. Open Spyder. Choose File and then New File.
2. To name the new file, choose File and then Save as….
3. Name the file HelloUser.py (with no spaces).
4. Once you name the file, the name of the file should be visible and updated in the IDE (confirm this visually).
5. View the File explorer in Spyder to confirm the creation of the file.
6. Type the code from Example 3.2.1 into the new file and save.
7. Indentation is critical in Python 3. Use the Tab to indent.
8. Run the program using the green run arrow or by choosing Run and then Run.

The output should appear on the IPython command console. The program will wait for you to type in a name and press enter. Then the program will output `Hello name`, where name is the name you typed in. Figure 3.1 illustrates the program and a possible input/output.

Solution and Review for Exercise 3.2.1:

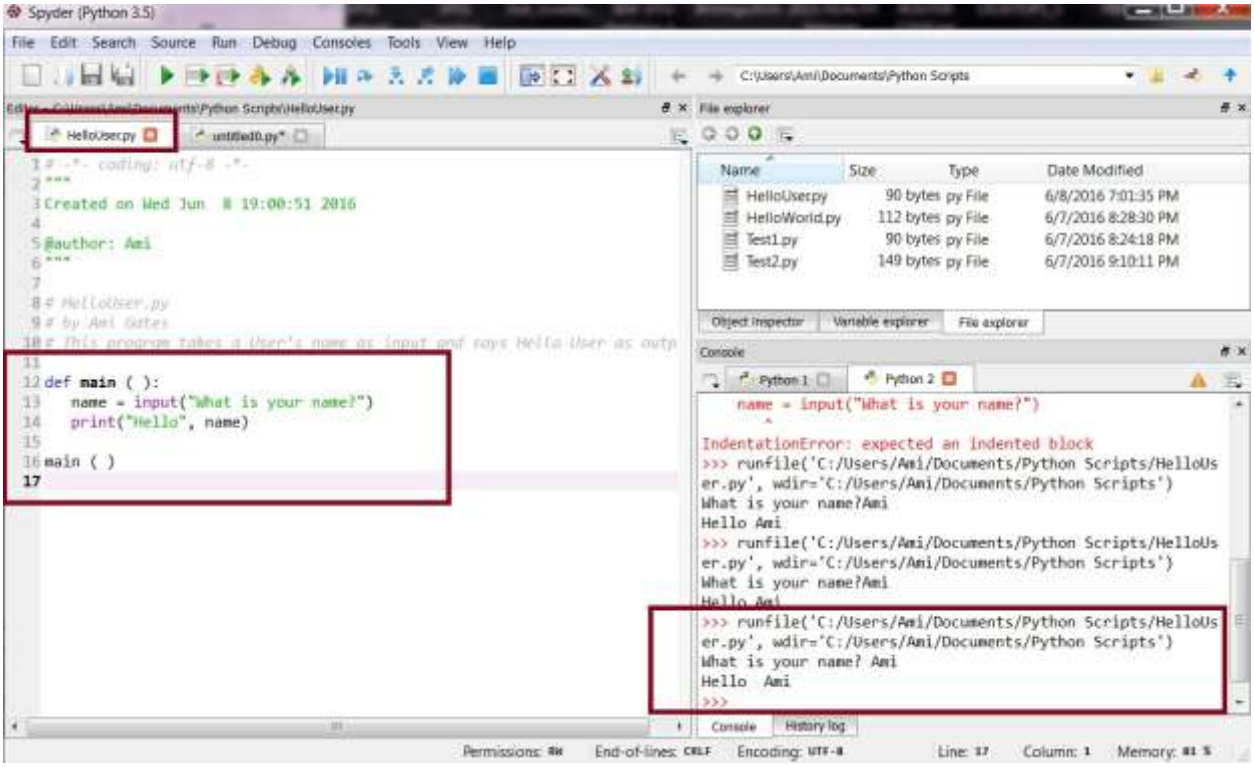

Figure 3.1: Illustration of Example 3.2.1.

Program Example Review by Line of Code:

```
1. # HelloUser.py
2. # Ami Gates
3. # This program takes a User's name as input
4. # and says Hello User as output

5. def main():
```

```
6.      name = input("What is your name?")
7.      print("Hello", name)

8. main()
```

To discuss this code, each line of code has been labeled with a number.

Code lines 1 – 4:

The first four lines of code are called **comments**. Code comments must begin with a pound symbol, `#`. The "#" symbol is used for add comments and notes into programs. Each line requires its own "#." When a line starts with a "#" it is ignored by the interpreter. Commenting code is very important because it will make the program easier to read and to update, easier to build onto, and easier for another programmer to use and alter.

Code line 5:

This statement of code defines a new function called **main**. Functions are formally covered in Chapter 5 but are used throughout the book. The basic syntax for a function definition is:

```
def FunctionName(<optional parameters>):
```

The word, `def`, is called a **reserved word**, as it is used specifically to define (create) functions. In code line 5, a function called main that has no parameters is defined. The open and closed parenthesis are required, as is the colon.

Code lines 6 and 7:

Code lines 6 and 7 are the **body** of the main function. To signify the statements that are contained within the main function, **indentation** is used (and required). Both line 6 and line 7 are indented using the tab so that they are contained in the main function. More specifically, the main function is their **scope**.

Line 6 uses an already defined Python function called `input`. The `input` function allows the program to collect and store information entered by a user. In this case, the user will enter a name that will be stored in the variable called `name`. Line 7 also uses a predefined Python function called `print`. The `print` function can print any words or characters (called **strings**) if contained in quotes (double or single). In addition, the `print` function can print the value contained in a variable. As such, the statement, `print("Hello", name)` will output the word Hello and the value saved in the variable name.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

Code Line 8:
The last statement in this program is the **call** to the main function. The general syntax of a function call (functions are covered in Chapter 5) is:

```
FunctionName(<function parameter values>)
```

Because the function main has no parameters, it is called with empty parenthesis. Notice also that the indentation of the call to the main function is not (and cannot) be contained inside the main function definition. Therefore, it is not indented.

Example 3.2.2: Write a program that contains a function that adds two numbers and prints the output of the sum. The two numbers must be input by the user.

```
#Example2AddingNumbers.py
#by Ami Gates
# This programs takes two numbers as input, adds them,
# and outputs the sum

def Adder():
   num1 = eval(input("What is the first number? "))
   num2 = eval(input("What is the second number? "))

   sum = num1 + num2

   print("The sum of ", num1, "and", num2, "is ", sum)

Adder()
```

Exercise 3.2.2: Creating a small basic function and using input, output, and variable.

To complete this exercise, take these steps.

1. Choose *File* and then *New file*.
2. Name the new file, Adder.py.
3. Next, type in the code above.
4. Save the file and run the program.
5. The Console will ask you for the first number. Enter a number and press enter.
6. The Console will ask you for the second number. Enter this as well and press enter.
7. The output will be the sum of the numbers.

Figure 3.2 will illustrate this exercise and an evaluation of the code and results follow.

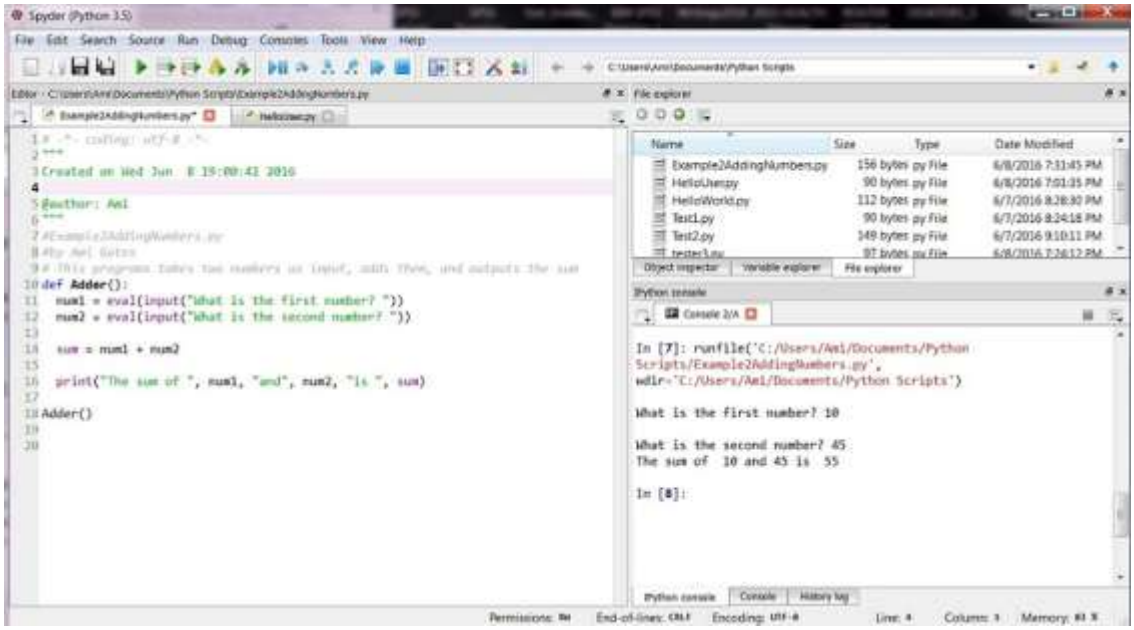

Figure 3.2: Illustration of Example 3.2.2.

Solution and Review for Exercise 3.2.2:

```
1.  # Example2AddingNumbers.py
2.  # Ami Gates
3.  # This programs takes two numbers as input, adds them,
4.  # and outputs the sum

5.  def Adder():
6.      num1 = eval(input("What is the first number? "))
7.      num2 = eval(input("What is the second number? "))

8.      sum = num1 + num2

9.      print("The sum of ", num1, "and", num2, "is ", sum)

10. Adder()
```

Code lines 1 - 4:

The first four lines of code are comments, which should include the name of the program and author, as well as what the program does in general. This area can also include a description of

input and output and other important details. Recall that comments are not interpreted nor executed. They are for information.

Code line 5:

This line of code defines a new function called Adder. This function has no parameters. Notice the use of the reserved word `def` and the colon.

Code lines 6 and 7:

These two lines of code ask for, collect, and store two numbers from the user of the program. The first of the lines of code defines a variable called, `num1`, and stores in it the first number the user enters. A variable can be thought of as a user-defined slot or space in memory where information or values can be stored, used, updated, and later removed or replaced. As an example, if the user types in a 10 for the first number, then `num1` will take on or store the value of 10. Notice also that both lines of code use the `input` Python function, which allows user input to be collected and stored in a variable. Finally, both lines of code use the `eval` function (also a Python defined function).

The `eval` function is used whenever expected input is numerical. The `eval` function can only be used, and must be used, if numerical input is to be collected and stored. Because this program collects two numbers that will be summed together, the two numbers must be collected as actual numerical values, which requires using `eval`.

Code line 8:

This line of code creates another variable called `sum`. The variable `sum` will hold the sum of the values in *num1* and *num2*. So, for example, if the inputs are 10 and 45, the value of `sum` would be 55.

Code line 9:

```
print("The sum of ", num1, "and", num2, "is ", sum)
```

This line of code will print out the sum of the two numbers. Notice that there are quotes around all the strings (words), such as *"The sum of."* Notice that there are no quotes around the variable names, such as *num1*. The *print* function in Python requires quotes around all words (called strings) and no quotes around variables. When quotes are placed around anything, such as a phrase, a word, any character, or even a number, they are intended to be taken literally. However, to access the value stored within a variable name, such as `num1`, the lack of quotes tells Python to access the value stored in that variable. As a quick additional exercise, update the code in Adder.py to be the following:

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016


```
# Example2AddingNumbers.py
# Ami Gates
# This programs takes two numbers as input, adds them,
# and outputs the sum

def Adder():
   num1 = eval(input("What is the first number? "))
   num2 = eval(input("What is the second number? "))

   sum = num1 + num2

   print("The sum of num1 and", num2, "is ", sum)

Adder()
```

The key difference here is that the `print` statement (in bold) has been changed. What does this do to the output? What do you think will happen here? The answer is that because **`"The sum of num1 and"`** is all in quotes, Python will print this precisely and will not print the value in `num1`. The following image illustrates this.

Code line 10:

Finally, notice the last line of this program. The last line is:

```
Adder()
```

This line is required so that the function called ***Adder()*** is called on to perform its internal lines of code. If you remove that line of code, and then save and run, what happens? Interestingly, nothing happens. There are no errors, because nothing is syntactically or semantically incorrect. However, logically, the ***Adder()*** function is never called on and so does not run.

?As a key point, it is not necessary to **call** on a function at some point in the program and after the function has been defined. As will be seen in Chapter 5, functions can be called throughout a program and functions can be called from other functions. In fact, most code contains many functions, and functions can be called on whenever (and as many times) as they are needed.

Finally, notice that the call to the Adder function is not inside of the function definition. All statements contained inside of the Adder function are indented to be within the Adder function. The call to the Adder function shares the same indentation as the comments.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

3.3. Errors, Bugs, and Debugging

Errors, often called **bugs** are part of the programming experience. Bugs in code are often syntactical in nature, such as accidentally typing a semicolon rather than a required colon, or having a missing quote, mistyping a function name, or having incorrect indentation. Syntactical errors must be found and corrected, or a program will not run. Spyder and other IDEs offer information about where potential error(s) might be, and what they might be caused by. This information can be helpful, but can also seem confusing. For example, in some cases, a small bug, such as forgetting to close a parenthesis, can create a cascade of errors. In this case, it might seem like there are many errors, but in fact, there is only one. IDEs also offer more advanced debugging options that allow a programmer to "step though" a program, review the contents of variables, and follow logical patterns.

In addition to syntactical errors, programs can also have logical errors. These can be much more deleterious and harder to find and correct. In the case of logical errors, the program may run, but it will not perform as expected. The act of **debugging** is the process of finding and removing errors, both syntactical and logical.

While it is not always possible to program without bugs, it is possible not to create a situation where you either cannot find bugs, or you spend hours looking for then. The key to bug mitigation is to test your code as you go. Code and test, code and test, adding just a few lines of code at a time. Using this method will allow a much faster location of bugs.

For example, in program **Example 3.2.1,** code line 6 is indented properly so that is contained in the scope of the `main` function. However, if the indentation is removed from this line of code, the program will not run, and the following error will be displayed in the Console:

```
File "C:/Users/Ami/Documents/Python
Scripts/HelloUser.py", line 13
    name = input("What is your name?")
       ^
IndentationError: expected an indented block
```

Another common error is a result of copying and pasting code from formatted documents, such as Microsoft Word or PowerPoint, directly into the Spyder Editor. In many cases, hidden issues or incorrect quotes will result and will cause "hard-to-find" errors. To avoid this, either type code by hand directly into the IDE (which is its purpose), or use a non-formatted option such as notepad.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

3.3. Naming and Reserved Words

3.3.1. Naming Rules and Reserved Words

In the previous examples, several names or words have been used. Each program file has a name, such as `HelloUser.py`. Functions have names, such as `main()` or `Adder()`. Variables have names, such as `name, num1, or sum`. User-defined names in Python are called **identifiers**.

Python also uses a set of names such as *`print`*, *`eval`*, *`input`*, and *`def`*. These names are not user defined. They are predefined and "owned" by Python. Words or names that are owned and predefined by Python are called **reserved words**. They are reserved only for their predefined definitions. Table 3.1 offers a set of the reserved words that cannot be used as names in Python.

and	del	if	or	for	class
not	with	elif	yield	break	continue
global	none	assert	except	in	def
false	as	is	import	return	finally
true	lambda	return	try	pass	
nonlocal	while	with	else	raise	

Table 3.1: List of reserved words in Python 3

In addition to not using reserved words for naming, there are a few extra rules for naming. The rules for naming demand that every identifier must start with either a letter, or the underscore character, "_". After starting with a letter or the underscore, any further sequence of letters, underscores, or numbers can follow. Identifiers cannot start with a number and cannot contain any spaces. Names are **case sensitive**, which means that if a variable has the name, *Num1*, then this is not the same as *num1* or *NUM1*.

Example 3.3.1: Examples of names that are permissible

```
Num1
SumOfNums
sumofnums
_NextName23
user_input
measure
```

Example 3.3.2: Examples of names that are not permissible

```
5Num
```

```
@VarName
print
else
75Go
```

An important programming practice to create identifiers (names) that describe the function, or variable, or object being created. For example, the name given to the variable that stores the sum of the two numbers might be `sum`. Similarly, the name of a function that collects and sums values might be named, `Adder()`.

Exercise 3.3.1: Errors caused by misusing reserved words.

Complete the following steps.

Update the program called `Example2AddingNumbers.py`, from Example 3.2.2 so that the variable called `sum` is changed to the Python reserved word of `while`. What happens?

Solution and Review for Exercise 3.3.1:

If this change is made and saved, and the code is run, there will be a syntax error because the word `while` is a Python reserved word that is not permitted for use as a variable name. Figure 3.3 illustrates the change and the resulting error.

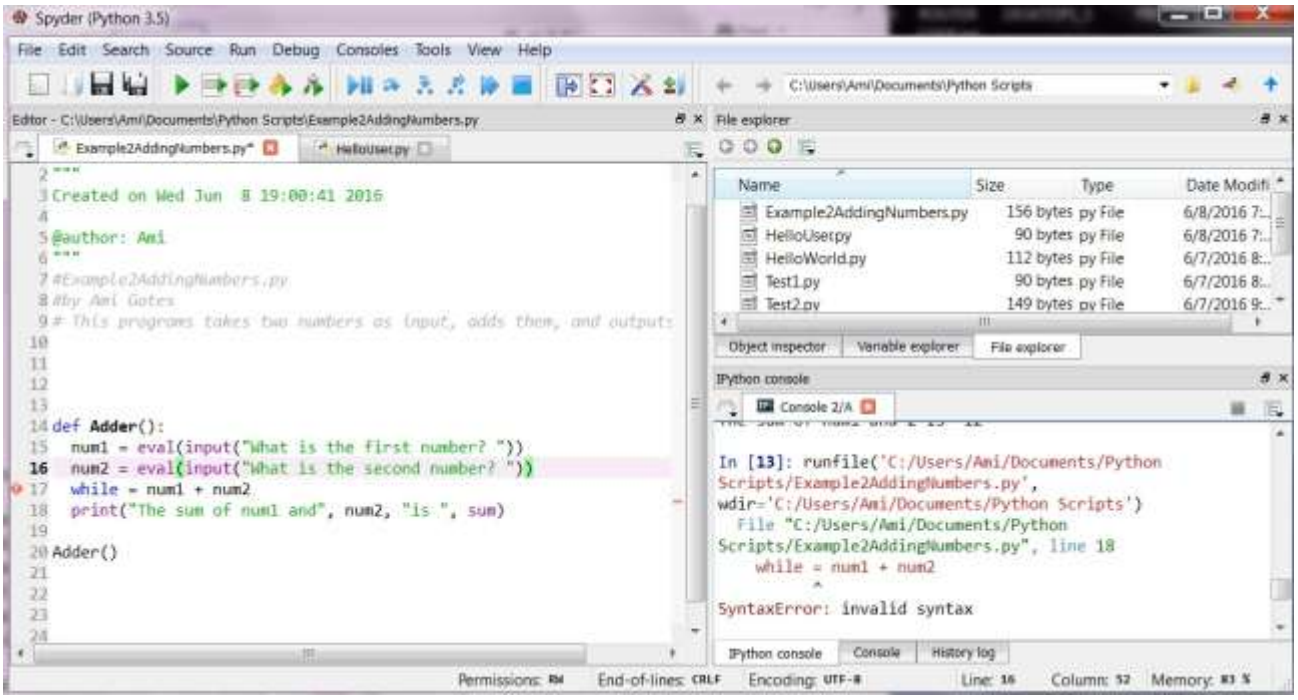

Figure 3.3: Illustration of Exercise 3.3.1.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

Exercise 3.3.2: Create a Mini Calculator

Using the examples above and the information about permissible naming, write a small program that takes as input three numbers and outputs the product and the sum of the three numbers.

Remember to start with creating a new file. Name the file MiniCalc.py. Also remember to place comments in the code that include the name of the file, your name, and what the program does (including expected inputs and outputs).

In this case, there are many options for writing this small program, and many options for naming.

Solution and Review for Exercise 3.3.2:

The following program is one possible solution to Exercise 3.3.2.

```
      # MiniCalc.py
      # Ami Gates
      # This small program takes three numbers as input
      # The program will output both the product and the
      # sum of the three numbers

      def Calc():
          firstnum = eval(input("Please enter a number: "))
          secondnum = eval(input("Please enter a second number: "))
          thirdnum = eval(input("Please enter a third number: "))

          sum_of_three_nums = firstnum+secondnum+thirdnum
          prod_of_three_nums= firstnum*secondnum*thirdnum

          print("The sum of the three numbers is ", sum_of_three_nums)
          print("The product of the three numbers is ",
prod_of_three_nums)

      Calc()
```

In this possible solution, the name of the function performing all the steps is called `Calc()`. Next, the names of the three variables that store the first, second, and third numerical inputs to be entered by the user are called respectively, `firstnum`, `secondnum`, and `thirdnum`. The variable that holds the value of the sum of the three numerical inputs is called, `sum_of_three_nums`, and the variable that holds the value of the product of the three numerical inputs is called, `prod_of_three_nums`.

Notice that the `print` statements have **strings** in them. The strings are the words inside the quotes that are intended to be printed literally. The print statements also have **variables** in them,

which are not in quotes. The variables in the print statements are not in quotes because the goal is to print what is contained in the variable (and not the literal name of the variable). Finally, the program has a call to the function `Calc()` so that the function runs. A function cannot run unless it is called. Figure 3.4 illustrates the code, the input, and the output in Spyder.

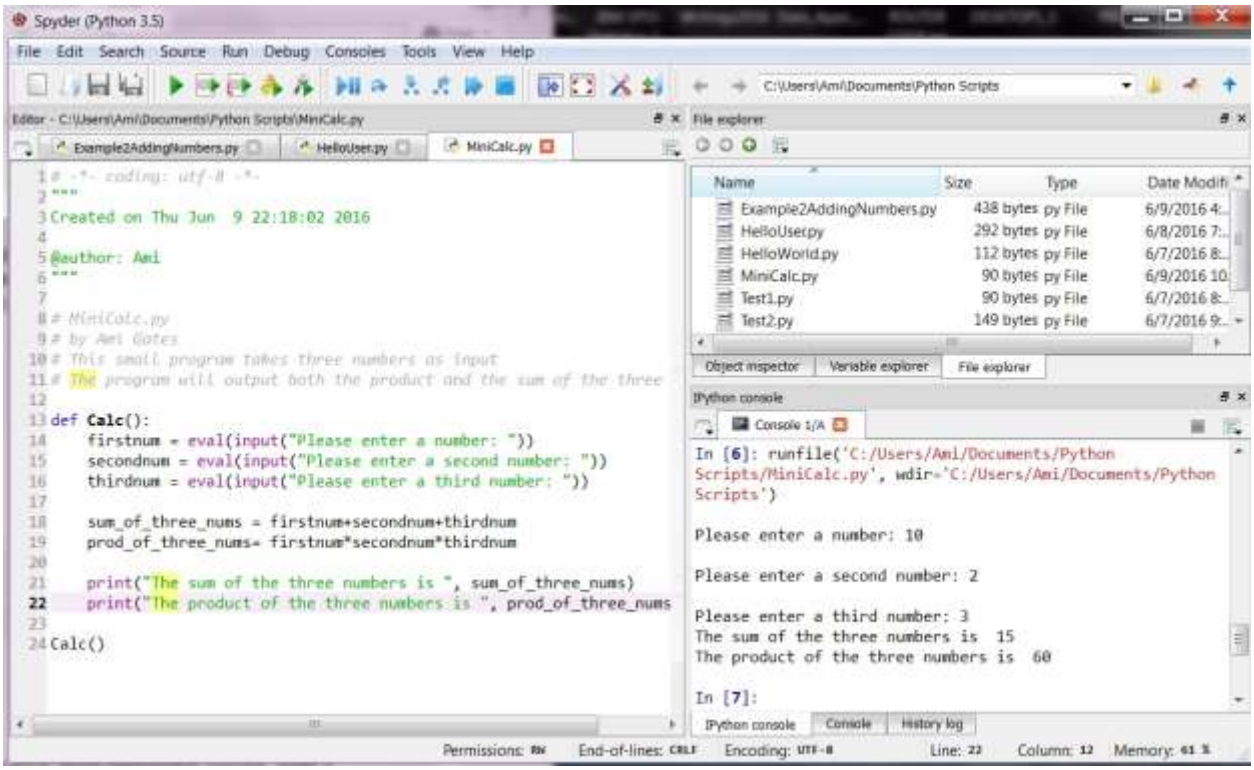

Figure 3.4: Illustration of possible solution to Exercise 3.3.2.

To complete this exercise, make changes to the program and test each change to see how it affects the results.

3.4. Expressions, Statements, and Simple Math

Very precisely, a **statement** is a line of code (or **instruction**) that once interpreted will be directly executed without further interpretation. It is a command.

Examples of statements might include:

```
print("Hello World")
value = 5
name = "bob"
for i in range(10):
```

An expression generally combines several elements, such as values, variables, and operators, and thus must first be evaluated before further action can be taken.

Examples of expressions might include:

```
(2 + 3)**7 - 12.4
print((2 + 3)**7 - 12.4)
sum1 = num1 + num2
```

3.4.1: The IPython command console and basic math operations

Statements and expressions can be simple or quite complex. Expressions will be evaluated if they are typed directly into the IPython Console. The simplest expression is an individual numerical value, such as 10 or 40.886. When typed into the IPython Console, the number will be evaluated as itself and returned as the result. Figure 3.5 illustrates this. It is recommended that you have the IPython Console open and that you try each example.

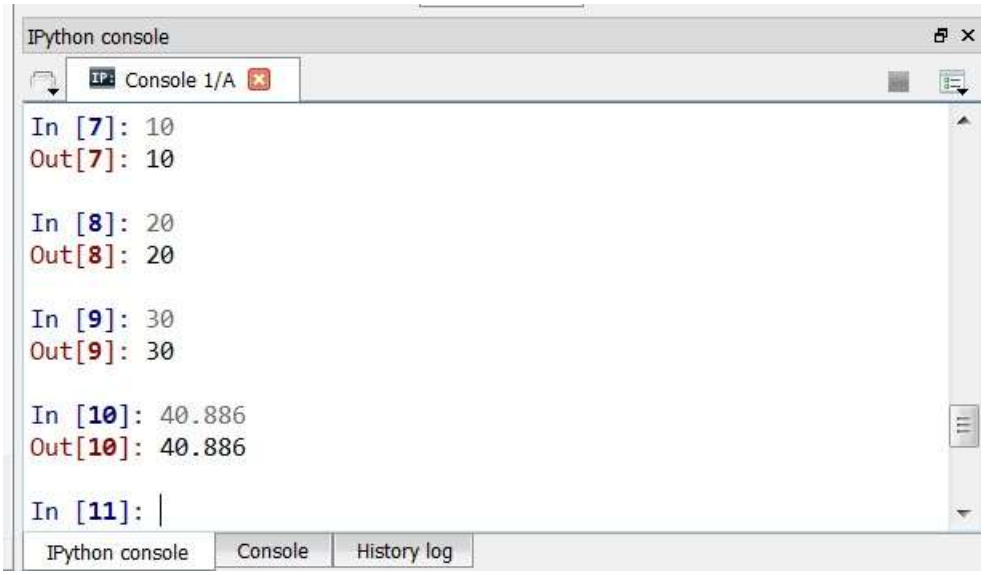

Figure 3.5: Illustration of the evaluation of a simple expression, a numerical value, in the IPython Console.

The IPython Console will also echo strings (collections of characters and numbers contained in single or double quotes). Figure 3.6 illustrates this.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

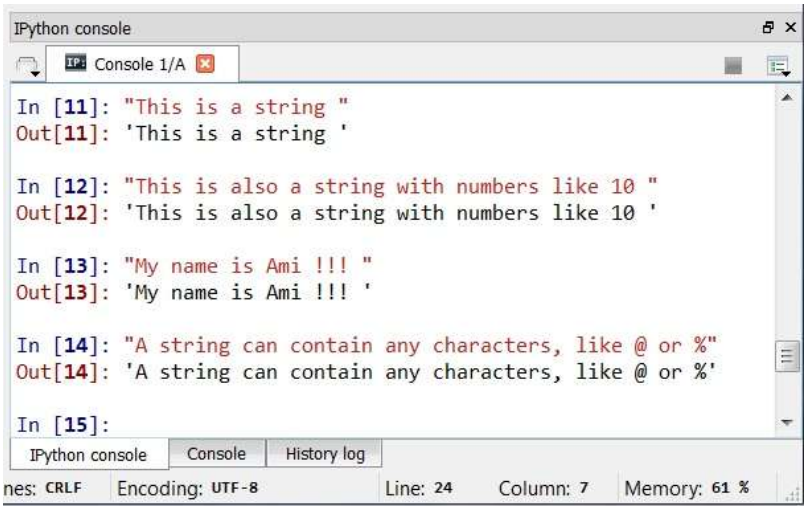

Figure 3.6: Typing strings into the console.

Expressions can also contain math. While Python can perform profoundly complex mathematics, Table 3.2 offers a collection of basic math operations standard to Python 3. Figure 3.7 illustrates the input and execution of several mathematical expressions, using the IPython Console.

Table 3.2: Basic Math operators in Python 3:

Operator	Operation	Operator	Operation
-	Subtraction	abs(x)	absolute value
/	division of floats	**	exponent
*	product	%	modulus
+	Addition	//	integer division

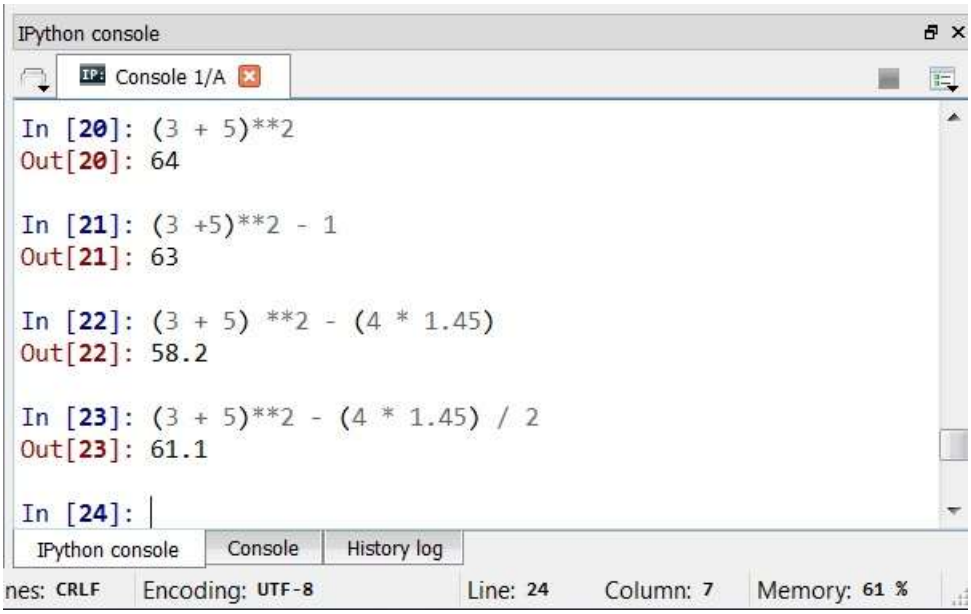

Figure 3.7: Example basic math expression in the IPython console

The IPython Console will execute any collection of statements and expressions. Figure 3.8 illustrates a `for loop` statement with a contained expression being executed within the IPython console.

```
for i in range(10):
    print(i+23)

23
24
25
26
27
28
29
30
31
32
```

The IPython console can be used to test ideas, run small pieces of code, assign variable names to values, and so on. The critical differences between using the IPython console versus a file is that nothing is saved in the console once it is closed and it is not ideal for larger programs or code organization. Any variables defined in the console will not be persistent. Any functions defined

in the console will also not be persistent and cannot be called from another location. The console can be used for program input and output, as a calculator, as well as for checking syntax.

3.5. Assignment Statements and Variables

It is often necessary to assign values to variables, to gather input and store that input to a variable named memory location, or to assign the value of an expression to a variable.

The basic form of an **assignment statement** in Python has the following structure:

```
<variable name(s)> = <expression(s)>
```

Example 3.5.1: Python assignment statements

The following statements are different examples of assignments in Python 3.

```
num = 5
num = num + 1
num2 = 56.7
x,y,z = 1, 2+3, 3**2
sum1 = num + num2
a,b = 50,45.777
x=60
mean = 55
s = 2.3
z = (x - mean) / s
getInput = input("What is your name? ")
Age = eval(input("What is your age ? "))
```

In the first assignment statement above, the variable `num` is being assigned the value of `5`. In the second statement, `num` is being **incremented** by 1. This means that whatever the value of num is (in this case it is 5), it will be increased by 1, and the result stored again in `num`. Therefore, `num` becomes 6. In the next example, `num2` is assigned a decimal value, which in Python is called a **float** (we will discuss types, numbers, and math in the next chapter).

The third statement,

```
x,y,z = 1, 2+3, 3**2
```

is a **multiple assignment** statement. The value of `x` is assigned a 1, the value of `y` will be assigned the value of 2+ 3 which is 5, and the value of `z` will be assigned 9.

The fourth statement assigns to `sum1`, the value that results from adding the values that are in `num` and `num2`. The fifth statement is a double assignment. It assigns the values 50 and 45.777 to the variables a and b.

The next four assignment statements assign 60 to `x`, 55 to `mean`, 2.3 to `s`, and then the result of the evaluation of the expression, `(x - mean) / s`, to `z`. Python will automatically evaluate expressions as part of the assignment statement.

Next, the `getInput` variable is assigned the string value of the name a user enters. If a user entered the name, "Bob Smith," for example, then `getInput` would take on the assigned value of the string, "Bob Smith." Finally, the `Age` variable will take on the numerical value of the user's input for age, once evaluated. The `eval()` function is used here so that the variable called `Age` can be assigned a numerical value (rather than a sting value).

As seen in the above example, it is possible to perform more than one assignment at the same time. For example:

```
x, y, z = 1, 2, 3
```

This assignment statement can also be written as:

```
x = 1
y = 2
z = 3
```

Similarly, the following **compound assignment** statement will assign the three variables to the three values. See the image below for the illustration.

```
firstname, lastname, ID = "Sally", "Rogers", 12345
```

3.6. Basic Input and Output: Print, Input, Eval

There are several methods for inputting and outputting information in Python, including the use of data files. File input and output is covered in Chapter 8.

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

To collect information from a user (input) and to offer results and information to the User (output), the IPython console can be used.

The two Python functions for gathering user input are `input` and `eval`. The general syntax for each is as follows:

```
Variablename=input(<string>)

VariableName2 = eval(input(<string>))
```

The Python function for offering output to the user on the console is the `print` function. The general syntax is as follows:

```
print(<statements/expressions>)
```

Example 3.6.1: Console Input and Output Statements

```
# ConsoleIO.py
# Ami Gates

def main():
    Firstname=input("Please enter your first name: ")
    Lastname=input("Please enter your last name: ")
    Height=eval(input("Enter your height in inches: "))
    Weight=eval(input("Enter your weight in lbs. "))


    print("Thank you ", Firstname, " ", Lastname)
    print("Your weight in kg is ", round(Weight/2.21,2))
    print("Your height in feet is ", Height/12 )

main()
```

Running this program can generate the following input and output:

```
Please enter your first name: John

Please enter your last name: Smith

Enter your height in inches: 75

Enter your weight in lbs. 145
```

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016


```
Thank you  John   Smith
Your weight in kg is  65.61
Your height in feet is  6.25
```

Exercise 3.6.1:

Follow these steps:

1. Type in the program above.
2. Save and run the program.
3. Confirm that the program functions as expected.
4. Update the program to request two more pieces of information: (1) The user's favorite temperature in Fahrenheit, and the user's Cell phone number.
5. Update the program to print out the user's favorite temperature as Celsius as well as the user's cell number.

Solution and Review for Exercise 3.6.1:

```
# ConsoleIO.py. This program is an example of console i/o
# Ami Gates

def main():
    Firstname=input("Please enter your first name: ")
    Lastname=input("Please enter your last name: ")
    Height = eval(input("Please enter your height in
inches: "))
    Weight=eval(input("Please enter your weight in lbs. "))
    Temp=eval(input("Enter your favorite temp in F: "))
    Cell=input("Enter your cell phone number: ")


    print("Thank you ", Firstname, " ", Lastname)
    print("Your weight in kg is ", round(Weight/2.21,2))
    print("Your height in feet is ", round(Height/12,2) )
    print("Your favorite temperature in C is ",
round(((Temp-32)*5/9),2))
    print("Your cell number is ", Cell)


main()
```

Publication Pending
Author: Dr. Ami Gates
Date: 8/11/2016

Example of possible input and output:

```
Please enter your last name: Dole

Please enter your height in inches: 60

Please enter your weight in lbs. 101

Enter your favorite temperature in F: 60

Enter your cell phone number: 333-333-3333
Thank you  Sally   Dole
Your weight in kg is  45.7
Your height in feet is  5.0
Your favorite temperature in C is  15.56
Your cell number is  333-333-3333
```

Program Example Review by Line of Code:

```
1. # ConsoleIO.py. This program is an example of console i/o
2. # Ami Gates

3. def main():
4.      Firstname=input("Please enter your first name: ")
5.      Lastname=input("Please enter your last name: ")
6.      Height = eval(input("Please enter your height in inches: "))
7.      Weight = eval(input("Please enter your weight in lbs. "))
8.      Temp=eval(input("Enter your favorite temperature in F: "))
9.      Cell=input("Enter your cell phone number: ")

10.     print("Thank you ", Firstname, " ", Lastname)
11.     print("Your weight in kg is ", round(Weight/2.21,2))
12.     print("Your height in feet is ", round(Height/12,2) )
13.     print("Your favorite temperature in C is ", round(((Temp-
32)*5/9),2))
14.     print("Your cell number is ", Cell)

15. main()
```

Code lines 1 – 2:

Lines 1 and 2 are comments. Here I have included the program name and the author name, as well as a brief description of the program goal.

Code line 3:

This line is the function definition for the main function.

Code lines 4 - 9:

These six lines of code gather input from the user via the console. In this order, the user will be asked to enter their first name, last name, height in inches, weight in lbs., favorite temperature in Fahrenheit, and cell phone number.

Because the first name, last name, and cell number are all strings (not numerical values), they can be collected using the `input` function. Each is stored in a variable. For example, the user's first name is stored in the variable called, `Firstname`.

Because the height, weight, and favorite temperature are all numerical values, they must be collected using both `eval` and `input`.

Code lines 10 - 14:

These lines of code offer the user output via the console. Line of code 10 is the following:

```
print("Thank you ", Firstname, " ", Lastname)
```

This line of code uses the `print` function. The print function can print strings, such as "Thank you," and it can also print the values stored in variable names, such as `Firstname` and `Lastname`. When this print statement is executed, it prints exactly what is inside of the quotes (including any spaces), and it prints the user's firstname (which is stored in the variable `Firstname`), as well as the users last name. Notice the `" "` between `Firstname` and `Lastname`. This will cause a space to be printed.

The line of code 13 is the following:

```
print("Your favorite temperature in C is " round(((Temp-32)*5/9),2))
```

Here again the print function can print strings, such as `"Your favorite temperature in C is"` and it can also print the evaluation of the expression, `round(((Temp-32)*5/9),2)`.

The expression, `round(((Temp-32)*5/9),2),` takes the value in `Temp`, subtracts 32 from it and then multiplies the result by 5/9. This is the well-known conversion formula from Fahrenheit to Celsius. Next, the `round` function is used to round the result to two decimal places.

The syntax for the `round` function is:

```
round(<expression>, number of decimal places)
```

Lines 11 and 12 also print a string and an expression, and both use the round function.
Line 14 prints the cell phone as a string.

Code line 15:

Line 15 calls the `main` function.

Summary

Chapter 3 offers an introduction to preliminary Python concepts. The following chapters will go into greater depth and detail and will cover several concepts, including loops, decision structures, data types, data structures, functions, file input and output, and graphics.