

COMPUTER PROGRAMMING

COURSE MANUAL

CMP414_22A



LESSON PLANS - TESTS - QUARTER REPORT FORMS



Week One

Day 1

IMPORTANT: This course requires additional software, which can be downloaded free of charge. For full instructions on downloading and installing the software for this course, see www.setonhome.org/python

You can check off work as you complete it!



In *Python Programming*, read the first section of the **Preface** and **sections 1.1 to 1.5**.

You don't need to understand all of this right now, but you should have a general grasp of what programming is. Since you are going to be programming, you want to know what exactly you are doing. Figure 1.2 and 1.3 on pages 8 and 9 might look a bit complex. What you need to understand from those is that programming takes inputs and gives outputs. The programmer decides what the inputs and outputs will be.

Inputs and outputs should be familiar concepts to you. You use inputs and outputs everyday. For example, if you want to get a drink out of a soda machine, you put in your money, and the machine outputs your drink.

Page 8 talks about the difference between a compiler and an interpreter. Python is an interpreted language rather than a compiled language. Be sure you understand the difference.

Day 2



Read and do the following:

On page 10, your book talks about using the "Python Shell." Using the Python Shell, you can type a command and the computer will run it immediately. Since we are using PyScripter, things will be a little bit different in some ways, but you still have access to the Python Shell. The Python Shell is located at the bottom of the screen of PyScripter. Any time the book talks about typing a command into the Python Shell, you can type the command there. See Image 1:

You may notice that the book is using Python version 3.4.3. Your version of Python will almost certainly be different from that. For example, the version of Python in the image above is 3.10.2. Don't worry about the program version, though. Later versions should still work fine with this course.

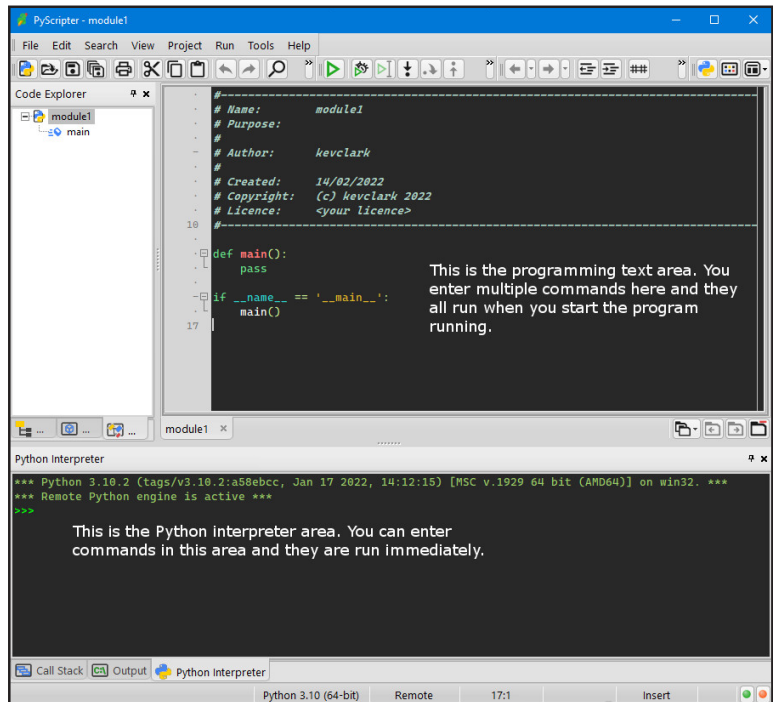


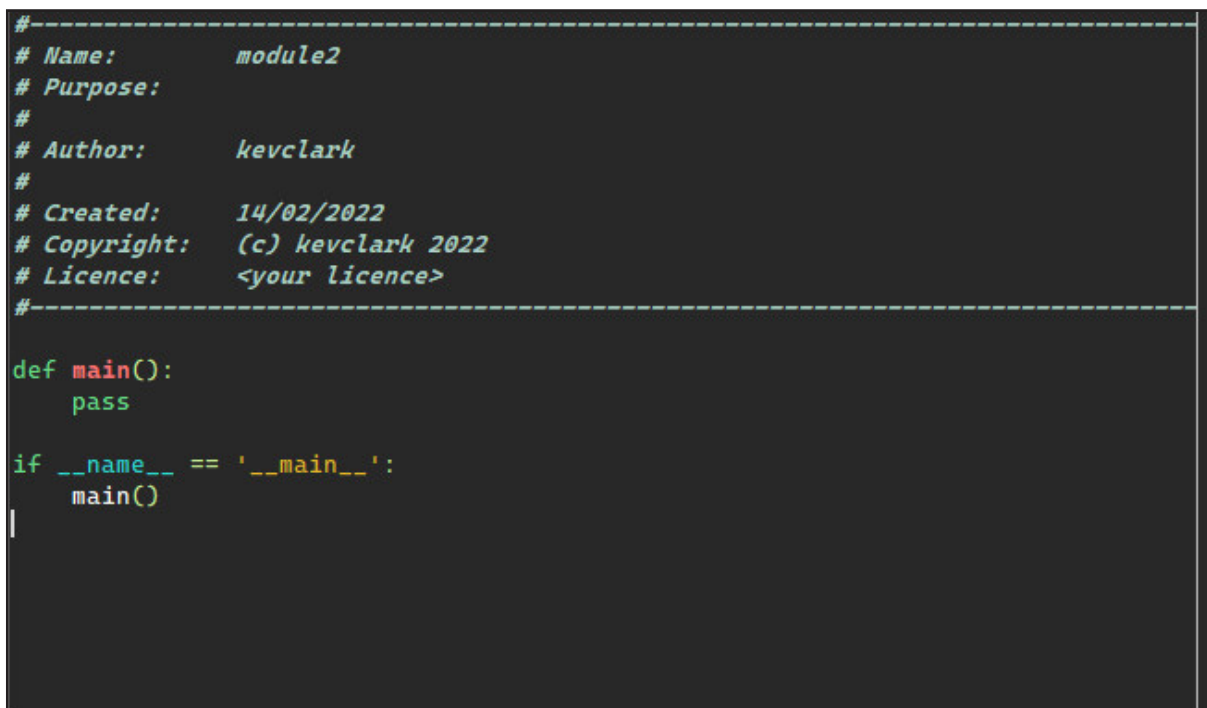
Image 1

Using the Python Shell (the part at the bottom left of PyScripter with the >>> prompt), you can type in the commands on pages 10 through 12 and they should work exactly as the book shows.

Starting on page 11, the book shows you how to "define a function." The lines that show how to do this can still be typed at the prompt as the book shows. However, you can also enter those lines into the word processing area at top right. You can replace the line that says "pass" with the two print lines you entered earlier at the prompt. You would have a program then that looks like this:

```
def main():
    print("Hello")
    print("Computers are fun!")

if __name__ == '__main__':
    main()
```



```
#-----
# Name:      module2
# Purpose:
#
# Author:    kevclark
#
# Created:   14/02/2022
# Copyright: (c) kevclark 2022
# Licence:   <your licence>
#-----

def main():
    pass

if __name__ == '__main__':
    main()
```

Image 2

Anytime you create a new Python program, the PyScripter word processor automatically starts it off for you with something that looks like this:

The first ten lines of the file are called "comments" and are not commands for the computer to perform. The remaining few lines are programming commands. For the time being, all you need to know is that anything you put in place of "pass" will be run by the computer.

The book has you define functions called `hello()` and `greet()`. To create something that is very close to what the book has, you could modify the text in PyScripter to this:

```
def hello():
    print("Hello")
    print("Computers are fun.")

hello()
```



How to start with a blank Python module

The starting template that PyScripter uses for new Python modules is often helpful, but you might find it confusing because then your program won't look exactly as programs look in the book. If you prefer, you can tell PyScripter to start each new Python module without any pre-entered lines. In other words, you will start out from a blank page. That will allow you to enter the programs exactly as they are in the book.

To start a new empty module, use the menu item Tools, then Options, then IDE Options. In the box that pops up, click the + sign next to IDE. Look for the item that says **File template for new Python scripts**. Click in the textbox across from that item and replace "Python Script" with "None" and press Enter. Click OK to close the window. Now when you create a new Python module it will be empty. If you ever want to go back to having the pre-existing template, then do the same as mentioned above and type "Python Script" in the textbox.

Indentation

The book hasn't mentioned it yet, but indentation is an essential part of Python programming. You'll notice how there are 2 lines indented under the "def hello():" line above. The indentation of those 2 lines means that they are part of the hello function. The line that says "hello()" isn't part of the hello function. That is the line that tells Python to run the hello function.

Day 3



Do **Sections 1.6 and 1.7** using the Python Shell.

The Chaos program in Section 1.6 looks like it has some complicated math, but the math isn't really the important thing. The math is simply an example of what Python can do.

Using PyScripter, for the chaos program on page 13, you only need to type in the five lines under the line that says "def main():" To begin writing the program, click on the File menu and choose New and then choose "New Python module". (You can also start a new module by pressing Ctrl+N.) Your new module will automatically be given a name such as "module1.py". To save the file as "chaos.py" click on File and then Save As and change the name to "chaos.py".

Notice that PyScripter tries to do some of the work for you. For example, when you type in a parenthesis the program adds a closing parenthesis for you. When you type a quotation mark, PyScripter automatically adds a closing quotation mark. It also tries to help you out with "syntax," which refers to the rules for how a computer language is written. Notice that when you type a parenthesis after "print", it not only puts in a closing parenthesis but also shows you some information about how to use the print function. It actually gives you more information than you need at this point in the program, so you can safely ignore that box.

Section 1.7 goes line by line through the chaos program. As mentioned, if you haven't set the PyScripter IDE to start out as blank, then your program will look slightly different due to the how PyScripter works. The PyScripter program using the default template would look something like this:

```

#-----
#Name:  module1
#Purpose:
#
#Author:      kevclark
#
#Created:     16/02/2022
#Copyright:   (c) kevclark 2022
#Licence:     <your licence>
#-----

def main():
    print("This program illustrates a chaotic function")
    x=eval(input("Enter a number between 0 and 1"))
    for i in range(10):
        x= 3.9 * x * (1-x)
        print(x)

if __name__ == '__main__':
    main()

```

NOTE

In the section above, the copyright notice is included simply because it is the default for PyScripter. There is no actual copyright claim being made.

Day 4

Skip section 1.8. It is an interesting explanation of some mathematical concepts, but does not directly relate to computer programming.



Read and study the **Chapter Summary**. Other than the last paragraph, this is all important information.



Read the following:

Program Tracing

When you run the chaos program, the program executes all the program statements and then ends. At the end of the program, you can see that it prints 10 values on the screen. The book on page 17 mentions that "for i in range(10)" is an example of a loop. So, even though the chaos program has very few lines (only 7 if you don't count comment lines and empty lines), more than 7 lines are being executed when you run the program. When you run it, the program executes the two lines beneath "for i in range(10)" a total of 10 times. So, the total number of lines being executed is actually 23.

Now, suppose that instead of just seeing the output from the program, you'd like to see how the program is actually being run, line by line. Watching what a program does in each line and the order in which lines are run is called *program tracing*.

PyScripter includes the ability to do program tracing—in fact, that's why we are using PyScripter for this course rather than the IDLE shell that the book mentions.



You can start tracing the chaos program in one of two ways. Either press the F7 key or click on the toolbar icon that is a down arrow with a dot at the bottom. If you hover your mouse over the right icon, the tooltip should say, "Step into subroutine (F7)". When you first click the icon or press F7 you will see that the first line of the program has a blue background line. The line with the blue background is the line which is going to be executed next. If you click or press F7 again, you'll notice that the blue line does not move to the next line in the file; rather, it moves to the line that says "main()", as shown here:

The reason that the first line under "def main():" isn't executed right away is that "def main():" means that the program is defining a function to be run later, rather than running those lines directly. (See page 16 for more information on the "main" function.) The line says "main()" tells the computer to execute the lines within the main function. So, if you trace to the next line (F7 or menu icon), the blue background bar will be on the first line of the main function.

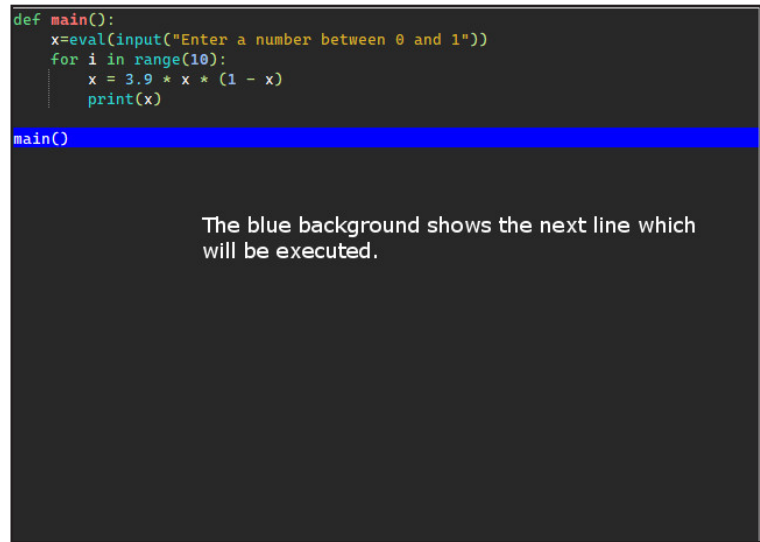


Image 3

If you trace to the next line, the program will ask you to enter a starting value for the chaos program. Once you enter the starting value, then the program will move to the blue bar to the line which says "for i in range(10)", which is the start of the loop. If you continue tracing, you'll see that the blue bar moves between the two lines within the loop a total of 10 times. Each time you trace over the "print(x)" line, the output area will show the value of x.

Another very helpful thing about the program tracer is that it also shows you the value of all the variables in your program. As the book says on page 16, "A variable is used to give a name to a value so that we can refer to it at other points in the program." In the case of the chaos program, there are only two variables: i and x. The x variable is the variable which the program mathematically processes for each loop. The i variable is what is called an index variable; its value increases by 1 each time the loop is run. If you hover your mouse over the i variable or the x variable as you trace the chaos program, you'll see a box pop up telling you the current value of each variable. As you trace through the loop multiple times, the values of these variables will change. As mentioned, the i variable will increase by 1 each time the loop is executed. The value you see for the x variable will change wildly, as the book mentions.

Once you start tracing your program, you can stop tracking in a few different ways.

1. You can keep clicking the icon or pressing F7 until the program is done.
2. You can press Shift-F8 or the menu bar icon with an arrow pointing up and a dot at the top (the tooltip says "Step out of the current subroutine (Shift+F8)"); this will also run the program to completion but won't show you line by line anymore.
3. You can stop the program from running and not complete the program by pressing Ctrl+Alt+F9 or clicking the menu bar item with the red square icon (the tooltip says "Abort debugging (Ctrl+Alt+F9)").

The PyScripter program tracer has many more features, but the ability to see what the program is doing line by line and also to see the contents of variables are the two most important features. The chaos program is only a few lines and it would be hard to make many mistakes in writing the program. If you did make a mistake, it would be relatively easy to fix, since there are so few lines that could have an error. However, Python programs can be thousands, or even hundreds of thousands, of lines long. They may include thousands of functions and thousands of variables. You can imagine that in a case like that, it would be very helpful to see exactly how the program is running and be able to check the value of variables at different times within the program.

As we go through the course we will talk more about program tracing.



Day 3



Look at Programming Exercise 7. If you wanted to do this, you would have to modify the program in a couple of different ways. Can you think of how it would have to be modified?

First, you would need to ask for two input numbers. You'd want to add a line similar to the line that inputs the value of x . You might want to call the new variable y . You would also want to inform the user that they were entering the first or second input. Instead of "Enter a number between 0 and 1", you would want to say, "Enter the first number between 0 and 1" and then "Enter the second number between 0 and 1".

Next, you would need to have the program print the header, which displays the inputs and then draws a dividing line. You would need something like:

```
print("input ", x, y)
print("-----")
```

Third, you need to generate and display a value for x and a value for y ten times. You would need to add a line to calculate y . You could copy the existing line that calculates x and substitute y for x :

```
y = 3.9 * y * (1 - y)
```

Finally, you would need to change `print(x)` to `print(x,y)`

The above changes would print the input and the outputs from the two numbers, but they would not be lined up in columns. The book mentions that chapter 5 shows how to print in columns; you don't have to know how it works, but the print lines below do line up. If you modified the chaos program to print the values lined up in columns, your finished program would look like this:

```
# File: chaos.py
# A simple program illustrating chaotic behavior, modified

def main():
    print("This program illustrates a chaotic function")
    x = eval(input("Enter the first number between 0 and 1: "))
    y = eval(input("Enter the second number between 0 and 1: "))
    print("input %0.3f      %0.3f" % (x,y))
    print("-----")
    for i in range(10):
        x = 3.9 * x * (1 - x)
        y = 3.9 * y * (1 - y)
        print("%0.6f      %0.6f" % (x,y))

main()
```

Day 4



Read and study **section 2.1 and 2.2.**

The explanation in section 2.1 of the process of implementing a computer program is very important. Too many people try to start writing a solution before the problem is well understood. If you don't understand the problem, it's hard to find a satisfactory solution. Much of computer programming consists in thinking about solutions.



Read the following:



Order of operations and parentheses

Notice the line in the program on page 30 that reads:

```
fahrenheit = 9/5 * celsius + 32
```

That line outputs the correct calculation due to "order of operations." You may recall from algebra that when doing mathematical computations, multiplication and division are done before addition and subtraction. This is important for the line above because 32 is added to the product of celsius times 9/5. It would give a different result to add 32 to celsius and then multiply by 9/5.

While you can rely on order of operations to get the right result, it's usually a better idea to use parentheses to group the operations. In this case, you might want to group the multiplication and division together like this:

```
fahrenheit = (9/5 * celsius) + 32
```

The line above makes clear that the multiplication and division are being done first and that 32 is added then added. Both lines do the same thing, but the second version makes it easier to quickly see what is happening.



The Virgin Adoring the Host, Jean Auguste Dominique Ingres

COMPUTER PROGRAMMING

CMP414_22A

This Course Manual is the property of Seton Home Study School and must be returned to Seton when the course has been completed.

We encourage you, however, to write in this Course Manual, or highlight in it to mark student progress.

For more information, visit:
setonhome.org/return-lp



Seton
Home Study School

1350 Progress Drive, Front Royal, VA 22630
www.setonhome.org



CMP414_22A

Updated 5/4/2022