

Lab 1

CS161

Exercise 1: In the beginning...

Why Python?

Python is a programming language that was first conceived by Guido van Rossum in the late 1980's and in 1990. While there are a number of programming languages that might be used for a first course in programming, Python offers a number of features that make it particularly applicable:

- It is interpreted, which for the introductory student means that they can type program commands into a console and immediately see the results. This feature makes learning details of the language much easier.
- The syntax of Python is generally simpler than other languages. Python's philosophy is "one way to do it" so that the number of details the student must learn and remember is reduced. This philosophy makes code more natural and more readable. The focus of learning code development should be on its readability, and Python supports it.
- Python provides high-level data structures and methods that make many programming tasks easier. This feature means that students are more productive: writing significant code, much more quickly.
- Python has many libraries that make tasks easier. There are many libraries specific to a field (biology, chemistry, physics, etc.) that make the language useful for "getting work done." Therefore, a student who knows Python has a host of software available to them for doing anything from graphics, to gene structure matching, to financial accounting, to music and anything in between.
- Python is free! Students can download it from the web for their personal machines. Furthermore, Python does not depend on a particular operating system. Thus, students are free to develop their code on Windows, Macintosh OS X or Linux and the code will (for the most part) run anywhere.

What you get

When you install Python for your computer, you get a number of features:

- a Python shell, a window into which you can directly type Python commands and where interaction between you and the programs you write typically occurs.
- a simple editor called IDLE, in which you can type programs, update them, save them to disk and run them. IDLE's interface is essentially the same on any machine you run it because it is a Python program!
- you get access to all the Python documentation on your local computer including:
 - a tutorial to get you started
 - a language reference for any details you might want to investigate
 - a library reference for modules you might wish to import and use

Let's get started

NOTE: although the text and screenshots below assume you are using IDLE, you can just as easily use Thonny or any other editor you prefer. If using Thonny, you may write your scripts in the editor window as usual and can use the shell window below it to run Python command interactively. If you are using another editor, you can run Python either in its shell or terminal window, or at the command line. Ask your instructor if you have any questions.

As previously mentioned, Python comes with both a shell into which you can directly type Python commands and a simple graphical editor called IDLE: the Integrated Development and Learning Environment. Both tools enable you to quickly and easily write Python code to explore a new idea or experiment. In these first exercises you will learn how to use both the shell and IDLE to write some simple code in Python.

The following examples assume you have Python version 3.7 installed on your computer. If you have a different version the steps described should be the same but your folders, icons, and windows may be labeled differently.

- To start IDLE on Windows go to the Start Menu → Python 3.7 → IDLE (Python 3.7 32-bit)
- To start IDLE on Mac go to the Finder and select Applications → Mac Python 3.7 → Idle.app
- Starting IDLE on Linux will vary depending on the distribution and even what window manager you are using. As a rule, you should find IDLE under either the Development or Programming menu (KDE or GNOME). Alternatively, you can open a terminal window and type IDLE at the prompt.

Working with the IDLE shell

Once you start IDLE you should see a window like the one in Figure 1.

This is the Python shell running in IDLE. It's interactive meaning you can type in Python statements and expressions and Python will immediately execute them. Notice the `>>>` prompt? That is where you will type your Python code. Try keying in the following:

```
1 + 1
```

```
print("Hi Mom")
```

```
answer = 37
```

```
print(answer + 5)
```

Your output should look much like Figure 2 below.

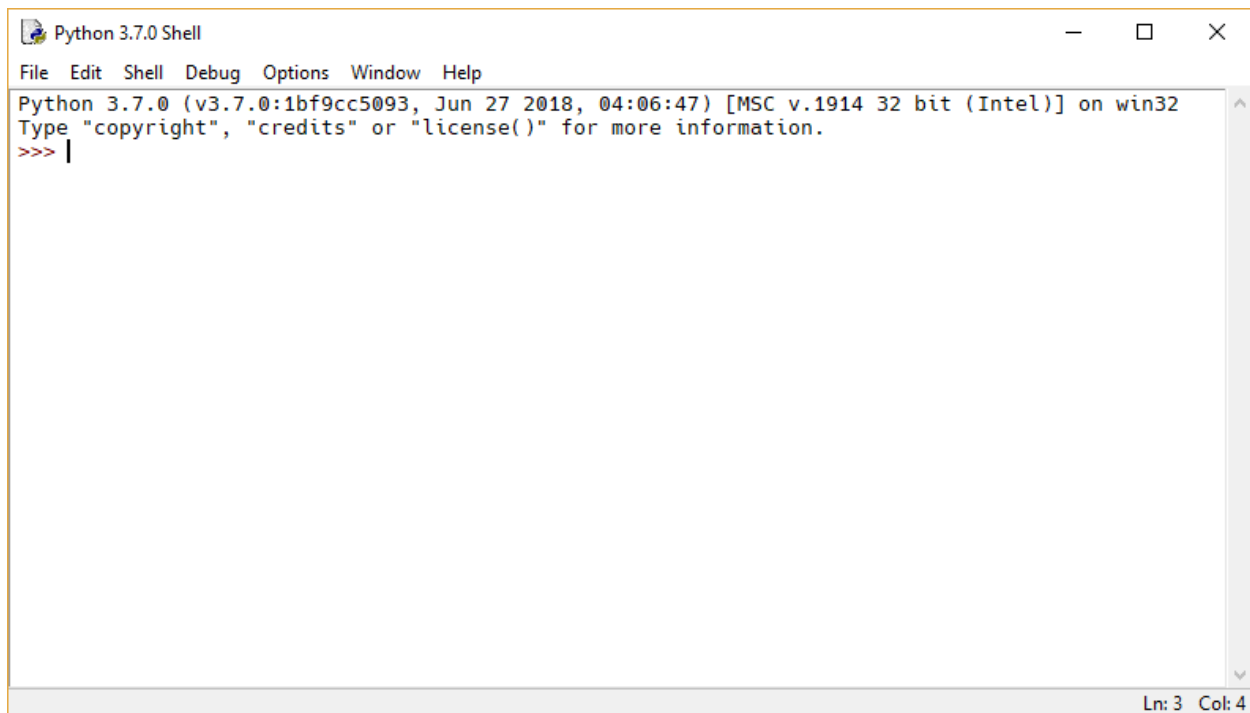


Figure 1 IDLE shell running on Windows 10

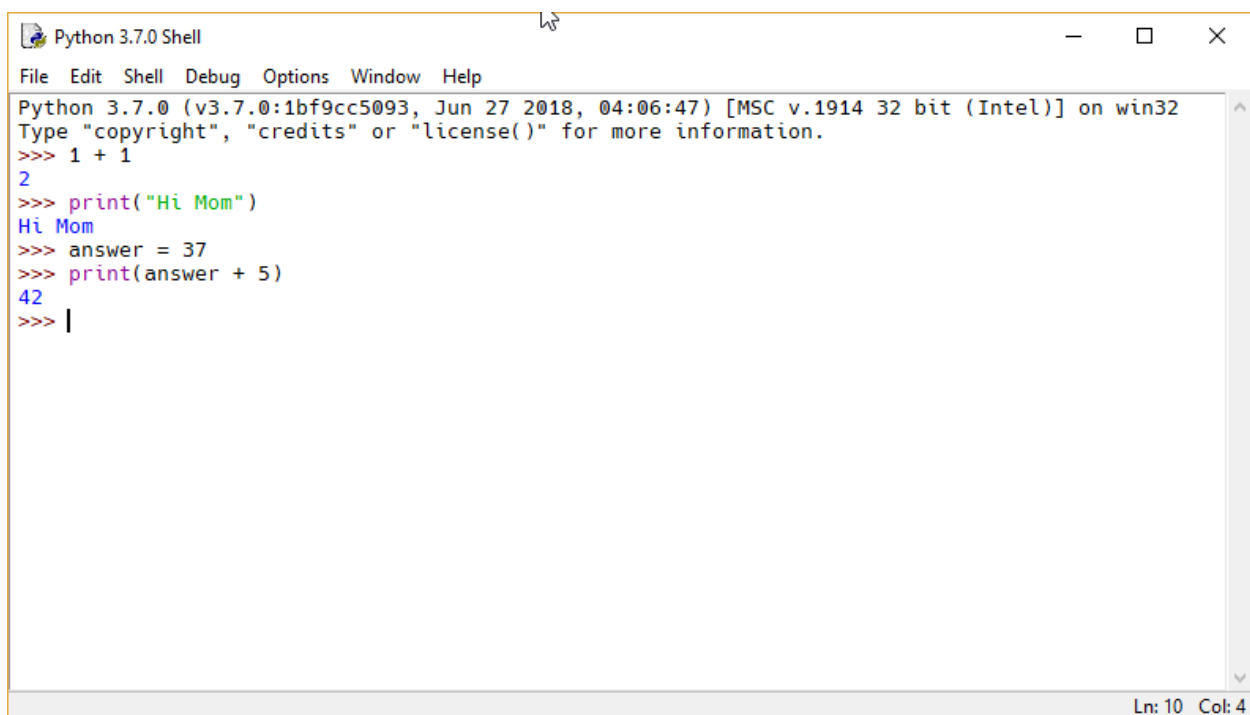


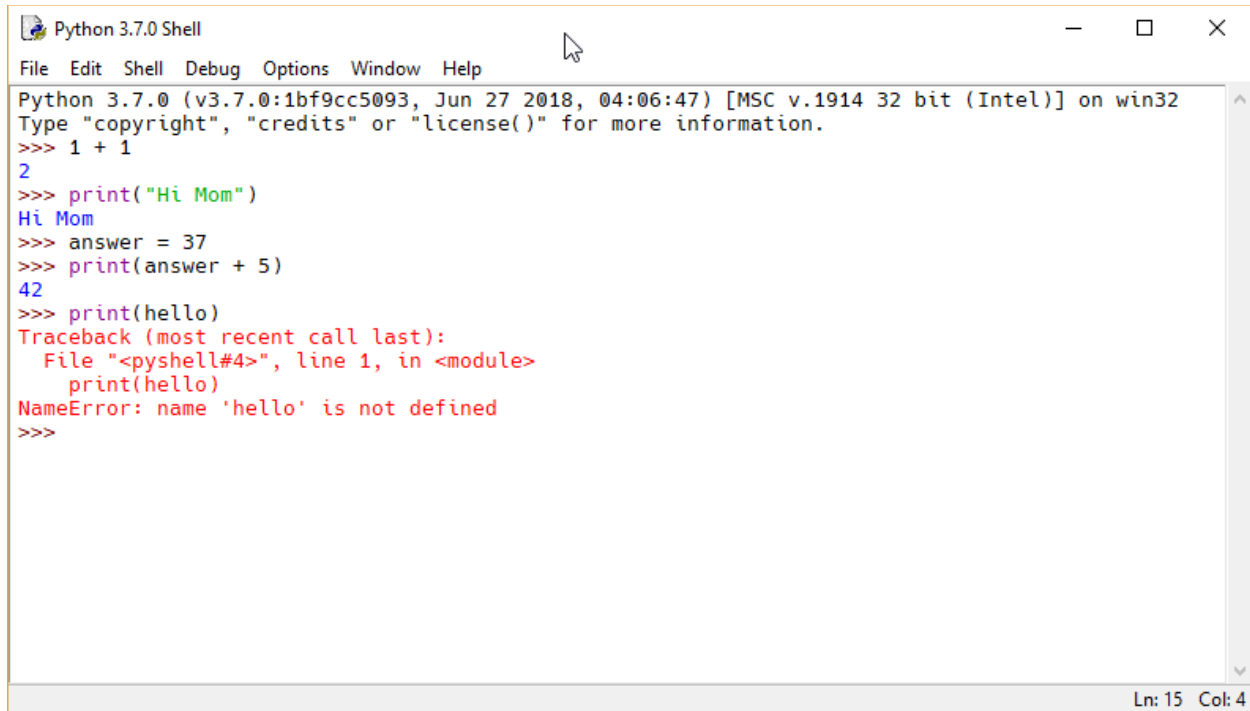
Figure 2 Simple interactive shell

What you type at the `>>>` prompt is immediately evaluated by Python when you press the Enter key. **Expressions**, like `1 + 1`, return a result. Statements, like `answer = 37`, do not return a result but do some work (in this case associating the numeric value thirty-seven with the identifier `answer`).

Occasionally you may make an error when entering your code. When you do Python will respond with an error message detailing the problem. Enter the following code to see such an error message.

```
print(hello)
```

Notice the error message in Figure 3 that was generated by that Python expression.

A screenshot of a Python 3.7.0 Shell window. The window has a title bar that says "Python 3.7.0 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area of the window contains the following text:

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 1 + 1
2
>>> print("Hi Mom")
Hi Mom
>>> answer = 37
>>> print(answer + 5)
42
>>> print(hello)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print(hello)
NameError: name 'hello' is not defined
>>>
```

At the bottom right of the window, it says "Ln: 15 Col: 4".

Figure 3 An error message

Specifically, it is a **NameError** error. This means that Python was unable to find a variable with the name you specified and thus was unable to perform the action you wanted.

Making a program

Typing into the shell is useful for exploring ideas but the commands that you write there are not saved and therefore cannot be easily reused. We need to save our commands in a script file so we can run the program again and again (and more importantly, turn it in!).

IDLE has a built-in file editor. To open a file, go to the shell window, left-click on File → New Window as shown in Figure 4. This will open the editor in a new window, see Figure 5. In here you can type all the Python code for your script. Unlike before in the interactive mode your code will not be evaluated when you press Enter. Instead, the entire script will be run line-by-line when you select Run Module from the Run menu, or press the F5 key on your keyboard. See Figure 6.

There is a long-standing tradition in computer science that requires that the first program you write in any new language is the Hello World program. This program does nothing but display the words, “Hello World” to the user. See https://en.wikibooks.org/wiki/Computer_Programming/Hello_world for listing of the Hello World program in hundreds of different programming languages.

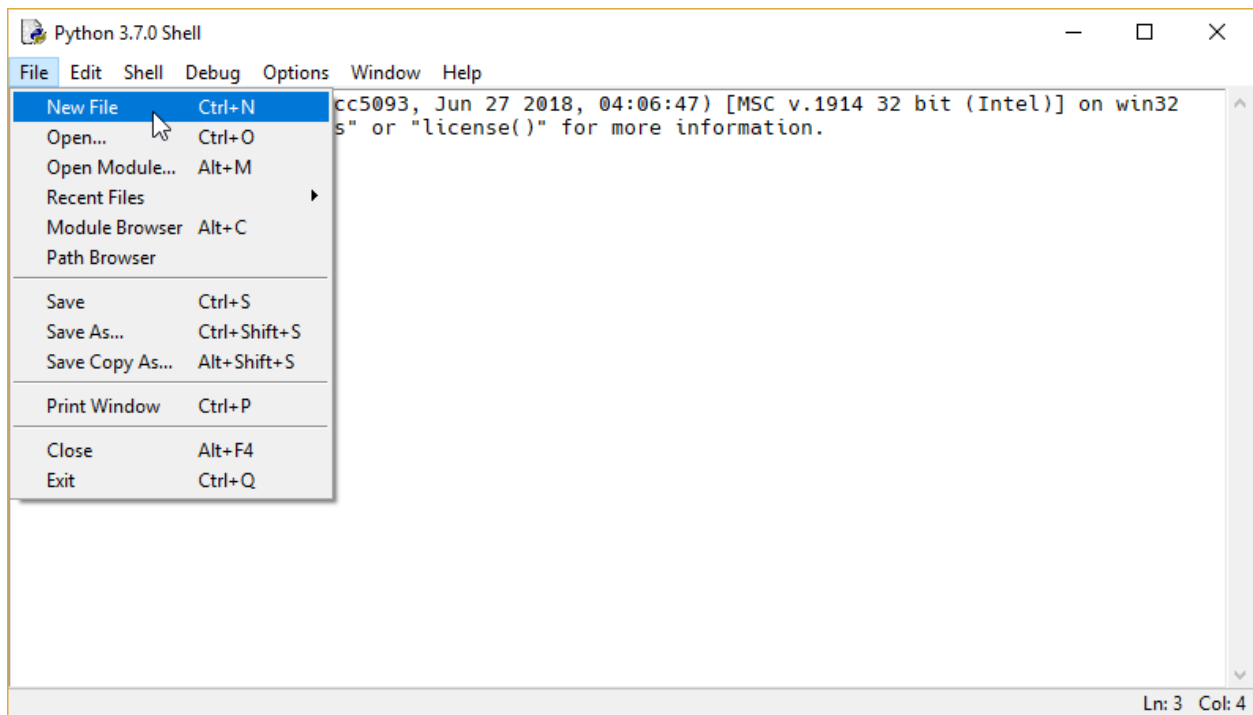


Figure 4 Creating a script file in IDLE

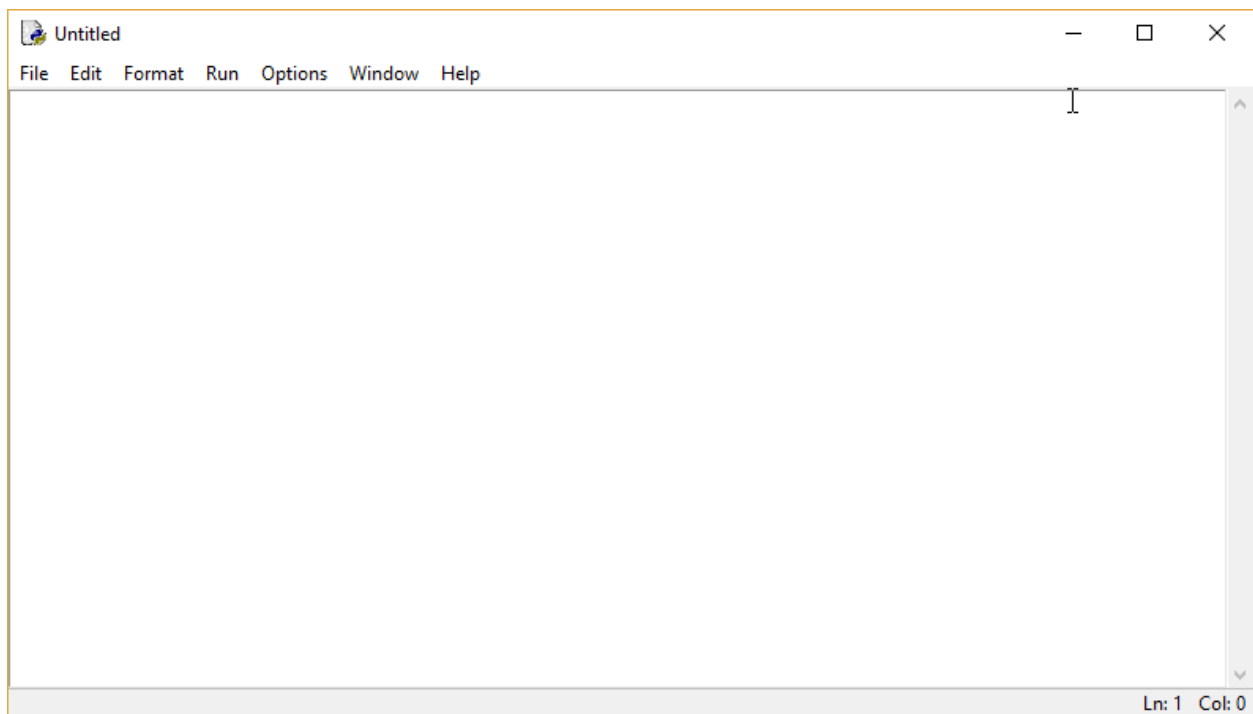


Figure 5 The IDLE script editor

In the empty, untitled file type in the following code:

```
print("Hello World")
```

Save your program as `hello_world.py` on your thumbdrive. Then, click Run → Run Module to execute your code as shown in Figure 6.

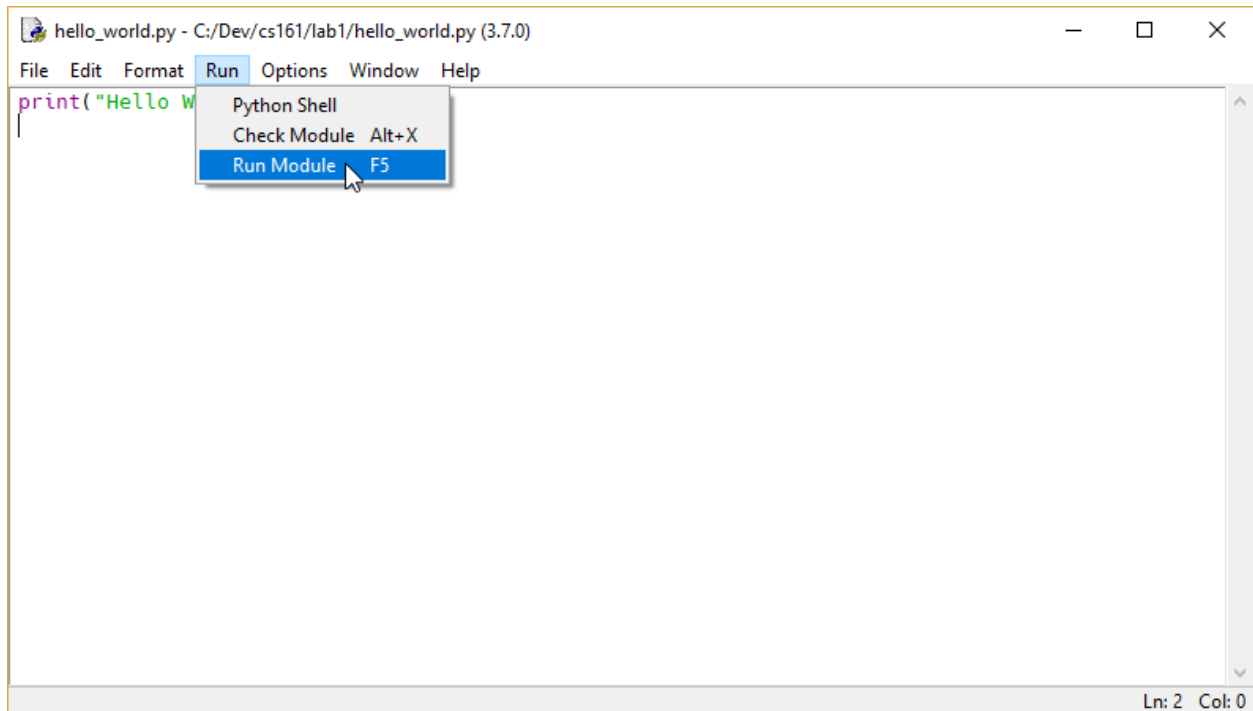


Figure 6 Running your script file

The result is new output showing up in your Python shell, as shown below.

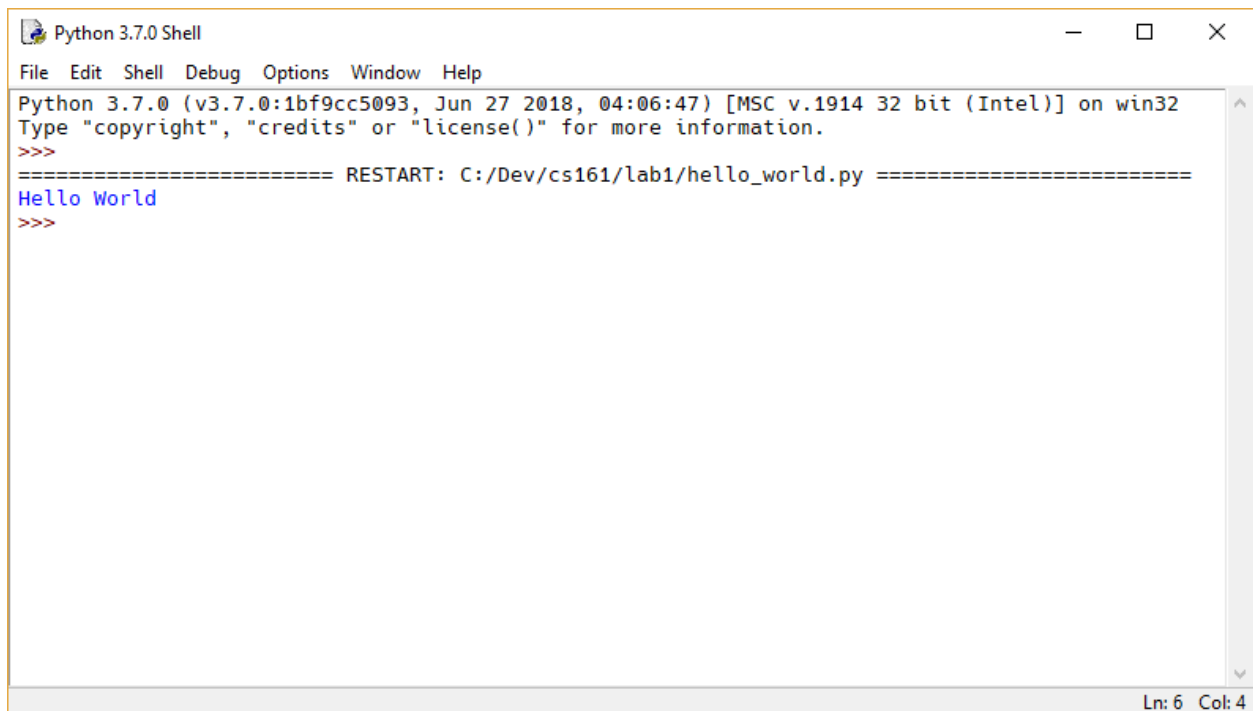


Figure 7 Output from the Hello World program

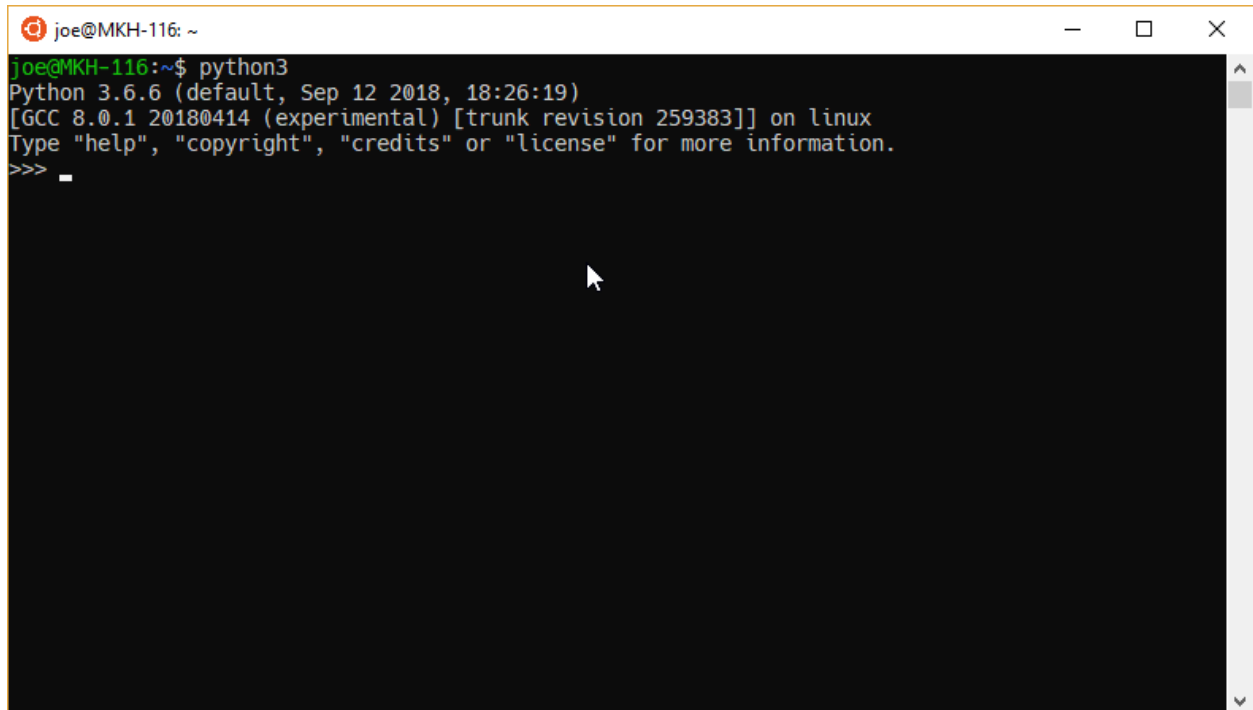
Congratulations, you wrote your first Python program of the term. Celebrate with a Spam sandwich!

Working with the terminal

You can also work with Python from the terminal. Python is an excellent language for developing programs that don't have a graphical interface. In fact, you could use Python to perform work on computers that don't even have a display attached to them!

Open a terminal window on your computer. On Windows it could be either PowerShell or the Command Prompt. On Mac OSX it's terminal.app. On Linux it will be your terminal emulator of choice. The screenshot in Figure 8 below is an Ubuntu terminal running under WSL.

At the prompt, type either `python` or `python3` to start the Python interpreter (remember, many Linux-based operating systems have both Python 2 and Python 3 installed and you must specify which version you want). Note also that this system is running Python version 3.6.6. This is fine, you will need Python 3.6 or newer for this class.

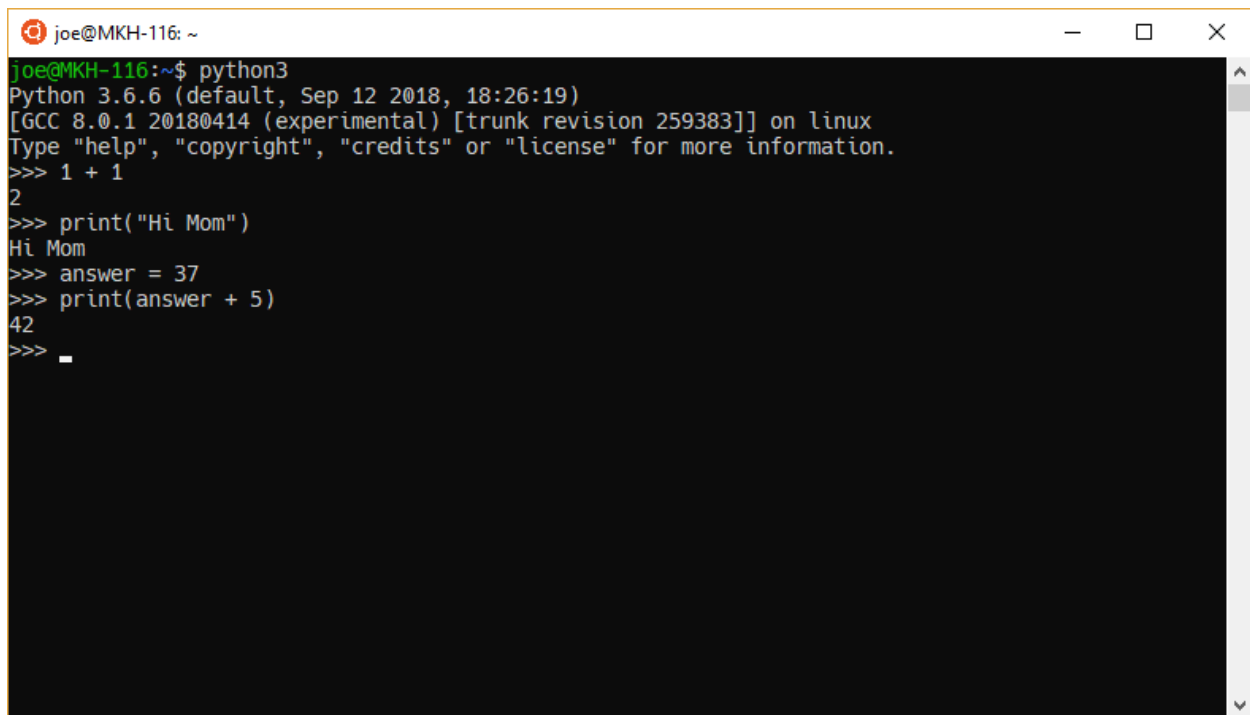
A screenshot of a terminal window titled 'joe@MKH-116: ~'. The terminal shows the command 'python3' being executed, which starts the Python 3.6.6 interpreter. The output displays the Python version, the default date and time (Sep 12 2018, 18:26:19), the GCC version (8.0.1 20180414), and the platform (linux). It also provides instructions on how to get help, copyright, credits, or license information. The prompt '>>>' is visible, indicating the interpreter is ready for input.

```
joe@MKH-116: ~  
joe@MKH-116:~$ python3  
Python 3.6.6 (default, Sep 12 2018, 18:26:19)  
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> _
```

Figure 8 Python running in the terminal

Enter the following code into the interpreter just as you did in IDLE.

```
1 + 1  
print("Hi Mom")  
answer = 37  
print(answer + 5)
```



```
joe@MKH-116: ~  
joe@MKH-116:~$ python3  
Python 3.6.6 (default, Sep 12 2018, 18:26:19)  
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 1 + 1  
2  
>>> print("Hi Mom")  
Hi Mom  
>>> answer = 37  
>>> print(answer + 5)  
42  
>>> _
```

Figure 9 An interactive shell running in the terminal

As you can see, you get the same output. This is because you are running the same Python as you did previously. Whereas before it used IDLE to get input and display its results to you it's now using the terminal. The underlying engine is the same! Exit out of the Python interpreter by typing `exit()` at the `>>>` prompt.

There is one difference, however, the terminal doesn't have a built-in editor like IDLE does. This is fine, however, you can still write scripts and execute them. In this case, you will write them in a separate text editor and run them from the prompt as shown in Figure 10 and Figure 11 below.

Now, you try it. From the terminal, run your `hello_world.py` script by navigating to the folder where you saved it and typing `python hello_world.py` (or `python3 hello_world.py` depending on your operating system) at the prompt. You see output similar to Figure 11 below.

Show your instructor your Hello World program running in both IDLE and the terminal to receive credit for this part of the lab and before moving on to the next.

Exercise 2: Turtle graphics

Turtle graphics is a popular way for introducing programming to kids of all ages, 3 to 99. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966.

Imagine a robotic turtle starting at (0, 0) in the x-y plane. After an `import turtle`, give it the command `turtle.forward(15)`, and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves. Give it the command `turtle.right(25)`, and it rotates in-place 25 degrees clockwise.

By combining together these and similar commands, intricate shapes and pictures can easily be drawn. For example, consider the following Python script:

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

Running it will product the following star.

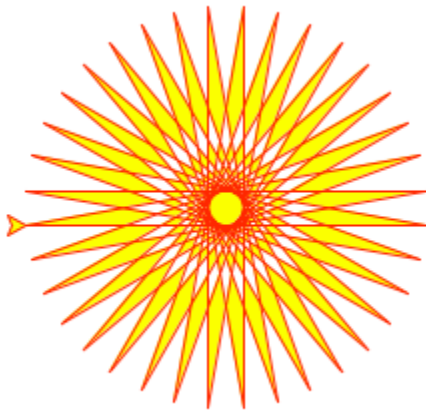


Figure 12 Turtle star

For each of the tasks below:

1. Analyze the problem
2. Determine the specifications for your solution
3. Create a design for your solution in pseudocode (include your pseudocode in the file docstring)
4. Implement your design in Python
5. Test/debug your solution

Each Python module you turn in must begin with a block, called a *docstring*, that looks like the example below (see [PEP 257](#) for more details). The opening triple quotes must be on line one of your file.

```
"""
Brief, prescriptive, one sentence description of what the program does.

A more detailed explanation of what the program does. Use complete
sentences and proper grammar. Treat this as if it is documentation someone
else will be reading (because it is). Use this paragraph to also discuss
your analysis of the problem and any assumptions you've made regarding the
program, the user, or the system on which it will run.

<pseudocode>

Firstname Lastname
Firstname Lastname (of partner, if applicable)
"""
<blank line>
<your code>
```

In your module docstring show the work you did following the software development stages above. Make notes about the problem as you analyze it; write specifications for what your program must do to find a solution; write your program out in pseudocode *before* you start to write Python; and design a set of test data that you can use to prove your program is producing correct results. Only after you've done all of this should you try to translate your pseudocode to Python.

Every function or method you write must also include its own docstring. That header will also use triple quotes and begin with a prescriptive, one sentence description of what the function or method does. If the function or method takes arguments, they should be described as well as any non-obvious return value(s). See examples below or [PEP 257](#) for more details).

```
def kos_root():
    """Return the pathname of the KOS root directory."""
    global _kos_root
    if _kos_root: return _kos_root
    ...

def complex(real=0.0, imag=0.0):
    """
    Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

Assignment Overview

This part of the lab focuses on rendering different colors and some simple graphics using Python libraries. You will use both selection (*if*) and repetition (*while*, *for*) in this lab.

The Problem

You will use Turtle graphics and both

1. Loops: To draw a regular polygon given the number of sides provided by the user.
2. Selection:
 - a. Add the capability to your program to fill polygons with a color indicated by a combination of the colors red, green and blue provided by the user.
 - b. You will check to ensure that correct color values are input.

Part 1

In this part of the lab you will write a program for drawing shapes using Python's Turtle Graphics module. Specifically, your program will:

1. Prompt the user for the number of sides of a polygon
2. Draw a regular polygon using "for"
3. Move away from the first polygon.
4. Draw the regular polygon using "while"

Write and save your code in a file named `turtle_1.py` for full credit.

Using turtle graphics:

In order to use turtle graphics in Python you must first import the turtle module. You can then use the help function in IDLE to find out what methods this module includes and what they do. Just type "import turtle" in the IDLE command window, hit enter, and then type `help(turtle)` and scroll up through the list and information. For more details Google "Python 3 turtle" or see docs.python.org/3/library/turtle.html.

The basic concept behind the turtle is the pen. The pen is either up or down. When down, the turtle draws as it moves around the Cartesian graph. There are a number of methods that are needed in this project:

`turtle.up()`, `turtle.down()`: Set the pen state to be up (not drawing) or down (drawing)

`turtle.right(degrees)`, `turtle.left(degrees)`: Turn the direction that the turtle is facing. The amount of turn is indicated in degrees.

`turtle.forward(distance)`, `turtle.backward(distance)`: Move the turtle forward or backward the amount of distance indicated. Depends on the direction the turtle is facing. Draws a line if the pen is down, not if the pen is up.

`turtle.goto(x, y)`: The goto method moves the turtle to a specified point, drawing a line along the way if the pen is down, and not drawing if the pen is up. Note: The turtle always starts at the point (0,0). The goto method moves to the indicated x,y coordinates.

`turtle.color(r, g, b)`: The color method sets the color that the pen will hold for all drawing until the color is changed. It takes three arguments, each a floating-point number between 0.0-1.0. The first is the amount of red, the second, the amount of green and the third the amount of blue.

`turtle.begin_fill()`, `turtle.end_fill()`: Use the command `turtle.begin_fill()` before you start drawing a figure. Draw the figure, then execute the command `turtle.end_fill()`. The figure drawn between the two fill commands will be filled with the present color setting.

`turtle.bye()`: Close the turtle drawing window.

First, try it!

The first thing you should do is try out some of the turtle commands by just typing them in the IDLE window or in the interpreter in the shell. You can get a much better feel for how the whole thing works by just trying it. Here is how to draw a line, turn left 45 degrees, and then draw another line:

```
>>> import turtle
>>> turtle.down()
>>> turtle.forward(20)
>>> turtle.left(45)
>>> turtle.forward(40)
>>> turtle.bye()
```

Regular Polygon

The formula for the total degrees of all angles on the inside of a regular polygon with n sides is $180 \text{ degrees} \times (n - 2)$ so each interior angle is $180 \times (n - 2) / n$. From that you can calculate how much to turn (exterior angle): $180 - 180 \times (n - 2) / n$. Use a loop to draw the regular polygon.

Loops

A for loop is useful when you know how many times you want to repeatedly do something, e.g. you know how many sides of a polygon you want to draw. Here is an example showing how to print something n times.

```
>>> n = 5
>>> for i in range(n):
    print("Hi!")
Hi!
Hi!
Hi!
Hi!
Hi!
```

A while loop is most useful when you don't know when writing the program that you will be repeating a fixed number of times, e.g. asking for input until correct input is provided. Here we will use while for a fixed number of times simply to practice using while. To contrast with for we will print something n times. Note how with the while we need to manage the counter variable count.

If you mess up the count management, you may end up repeating forever. In that case use the `ctrl-c` key combination to terminate the loop.

```
>>> n = 5
>>> count = 0
>>> while count < n:
    print("Hi!")
    count = count + 1
Hi!
```

Hi!
Hi!
Hi!
Hi!

Part 2

Save a copy of your program in a file named `turtle_2.py` and make the changes described below. You must turn in both files named as directed to receive full credit.

Your new program will read in color values from the user and then use the methods in the turtle module to draw the polygons (above) filled with the color indicated. There are many ways to create a color but a common one used in computer graphics is the process of additive color (see http://en.wikipedia.org/wiki/Additive_color). Combining different amounts of red, green and blue can create most, but not all, colors.

For example, here is how to draw a red circle: all red, no green, no blue.

```
>>> import turtle
>>> turtle.color(1,0,0)
>>> turtle.begin_fill()
>>> turtle.circle(20)
>>> turtle.end_fill()
>>> turtle.bye()
```

Add the following to your polygon program:

1. Add the capability to your program to fill polygons with a color indicated by a combination of the colors red, green and blue provided by the user (i.e. Python `input` command).
2. You will check to ensure that correct color values are input. Use selection (`if`) to check that the values input are greater than or equal to 0 and less than or equal to 1, i.e. $0 \leq n \leq 1$.

Part 3

Using turtle graphics, write a Python program to draw a picture of your choosing. Perhaps a simple drawing of a stick figure(s) standing next to a house? Fill your drawing with color where appropriate. Remember, you can use `turtle.up()` and `turtle.down()` to control whether or not the turtle draws as it moves making it easy to draw unconnected shapes. Write and save your code in a file named `turtle_3.py` for full credit.

Submitting Your Lab

Your instructor will walk you through the process of submitting your lab electronically before it is due.