Lab #1: Introduction to Python

Purpose

The purpose of this lab is to write simple Python programs and run them.

Tasks

In this lab we will write Python programs to get a feel for programming using Python.

Success

To succeed in this lab, we will have to pay close attention to the exact characters we use – both spelling and case (uppercase vs. lowercase). Programming languages are picky in general and Python especially so about spaces on each line.

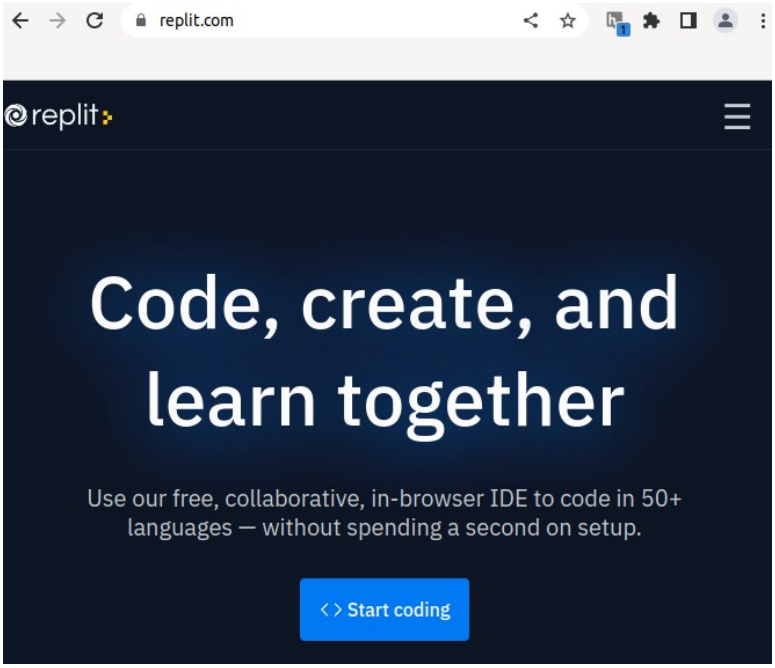
Introduction

In this lab we will use the replit.com website to write and run programs using the Python programming language. We will need internet access since we will use the website to write and run the Python programs.

Programs due 11:59pm Sept. 22nd

Preliminary Steps

Use a browser to go to replit.com and you will very likely see a page like this:

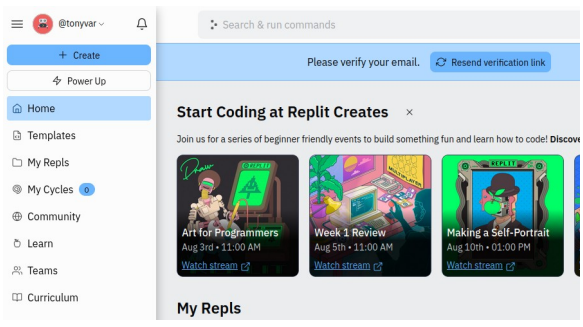


Scroll down this page to read more about what this site lets you do and when you are done reading about replit, click the "<>Start coding" button.

This will ask you to create a Replit account with your own Username, Email, and Password. Write these down so you can log back in later - saving these in your browser may help - and then click "+ Create Account".

This creates a replit account which you can use from anywhere (even a smartphone) as long as you have an internet connection.

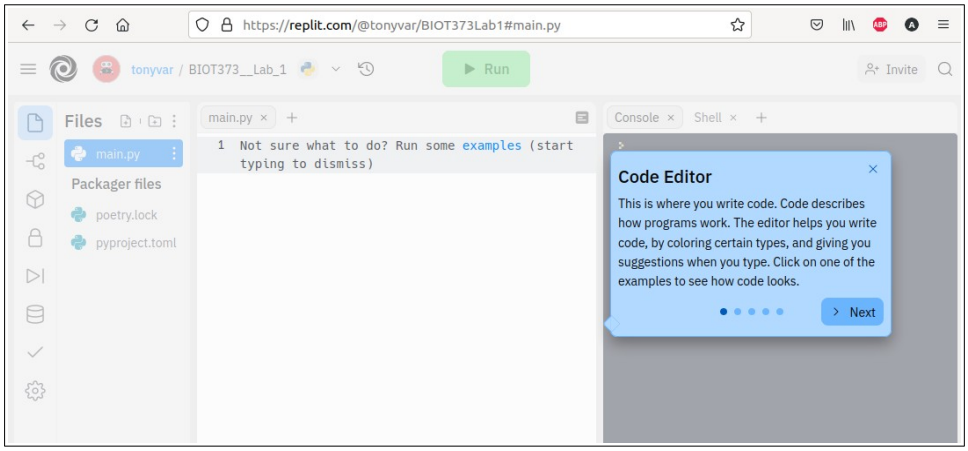
You will be asked how familiar you are with programming and you can choose from "Not at all" to "Expert" or skip this.
Replit will ask if you want to start with a coding tutorial. You can choose whether to pick a tutorial or skip it and, instead go to your "home page" which may look as shown here:



Go ahead and see what the menu items, "Home", "Templates", "My Repls", and "Learn" show you. Although there is a lot of information here, we only really need our "Home" for now. Check your email for a message from replit and follow the directions to verify your Replit account.

Click the “+ Create” button towards the upper-left part of the window to create a new replit project – a “repl”: use the “Python” template and give it the default “title” or a unique name like “BIOT373Lab1” and click “+ Create Repl”. Note that, by default, our repls will be publicly viewable.

This will put you in the main page for your “repl” – it creates a python file named “main.py”

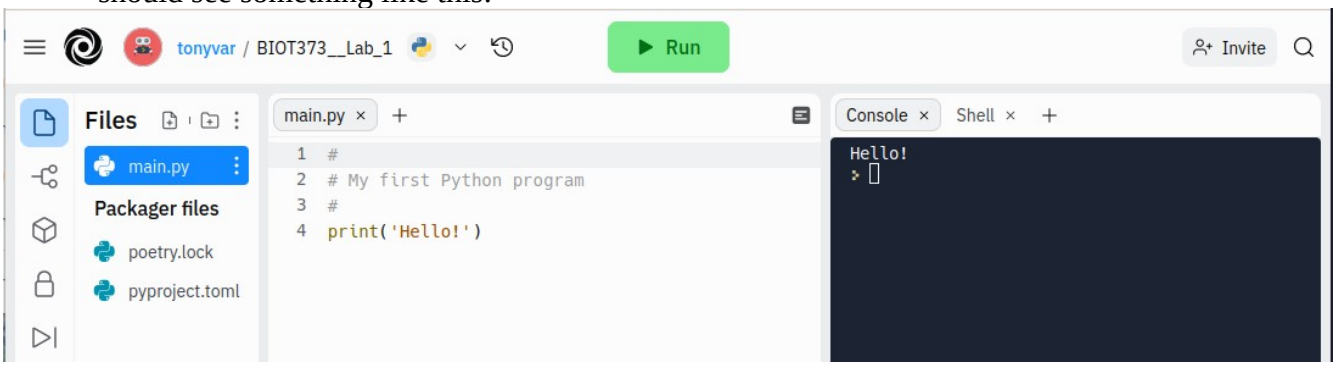


In the pop-up window, read the descriptions of the various parts of this page like the “Code Editor”. You are not expected to memorize any of these things – eventually we all remember the useful things by using them. No need to worry if you cannot remember the names right now.

In the Code Editor, enter the following text EXACTLY including the “#” symbols:

```
#
# My first Python program
#
print('Hello!')
```

As you make changes to this **`main.py`** file (that's a **`.py`** as in “**`python`**”), your changes are automatically saved. You may notice that the colors of the text may change because the Code Editor treats the content of your file as a Python program and recognizes its different components. When you are done entering in the program, click the green “Run” button and you should see something like this:



If everything worked, your program will have printed a “Hello!” in the “Console” window. If so, Congratulations! You just wrote and ran your first Python program!!

Feel free to take a break and think about the important steps: writing your program, running it, and observing what it does. We will do a lot of this and will get a lot of practice.
There are a couple of things to be careful about – the spellings of things are very important; in general,

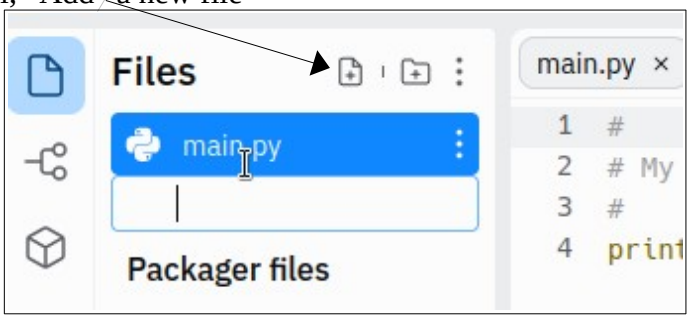
we cannot expect a spelling mistake to be tolerated very well at all. Nor do computers tolerate use the wrong “case”: we cannot expect that using uppercase letters instead of lowercase letters (or vice versa) will work. So be very careful and check spelling, and especially the case of letter characters.

The first three lines of our program do not “do” anything; they are there to help us to read the program more easily and understand how it works. They are called “comments”.

The last line is the only one in this program that actually “does” anything and it says that the string `"Hello!"` should be displayed in the “console”. If the above program worked, great! If it did not, we have an opportunity to learn how to get it working! Talk to the instructor and your classmates about what you thought of the whole process. All of this can be overwhelming and so be sure to take your time and proceed at whatever pace is comfortable for you.

Basics - arithmetic

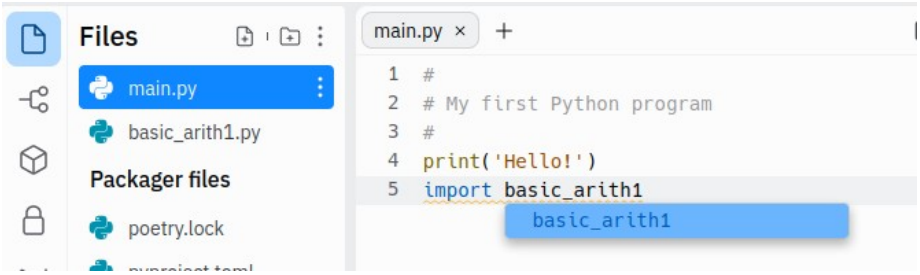
1. In your Lab 1 repl, “Add” a new file



   and call it “basic_arith1.py”.
2. Using the Code Editor, enter this program:

```
#
# Python program for arithmetic
#
print( 502 + 34 )
```

   To run this new program, modify the old “main.py” program so that it includes an “import” statement as shown here.



   Click Run and make sure the output of the program makes sense. What we did was to run the `main.py` program which, in turn, imports and runs the new `basic_arith1.py` program.

   In the following steps, whenever we create a new Python program, we will have to add an `import` statement to the `main.py` file to be able to run the new program.

Computer programs typically store the values like **502** and **34** in what are called "variables". They are called variables because the values they hold can vary over time. At one point in time, the value can be a 37 and later on, it can be something different, like 103.
For example, we can re-write the above program as:

```
#
# Python program variables and arithmetic
#
x = 502
y =  34
print( x + y )
```

Variables start with a letter or an "underscore" character. Examples of valid variable names are: **number**, **length_of_coding_sequence**, **x1**, **_x2**, **x2_times_y2**. Variable names cannot start with numbers.

The "**=**" is called the assignment operator and it signifies that the value on the right of the "=" symbol is assigned to the variable name to the left of the "=" operator. Read this operator as "assign"; it does not mean "equals".

Once a variable is assigned a value, we can use that value again and again:

```
#
# Python program with variables and arithmetic
#
# a pair of values
x = 502
y =  34
name = "Freddy"
#
# arithmetic using x and y:
xplusy  = x + y
xminusy = x - y
xdivy   = x / y
xtimesy = x * y
#
# print out the results:
print( name, ": x + y is " , xplusy   )
print( name, ": x - y is " , xminusy  )
print( name, ": x * y is " , xtimesy  )
print( name, ": x / y is " , xdivy    )
```

The variable `name` in the above program is used to store a sequence of characters (called a **string**) and Python allows us to perform many kinds of string manipulations. Anything in single-quaotes (`'Hello'`), double-quotes (`"Hello"`), triple single-quotes (`'''Hello'''`) or triple double-quotes (`"""Hello"""`) is a string – a sequence of characters.

We can also use formatting and an operation called "**string interpolation**" to do the same kind of thing as above. For example, we could have used this instead of what we did above:

```
print( " {0!s}: x + y is {1}".format(name, xplusy) )
```

When we use `{0!s}` inside double-quotes we allow the ***value*** of the first thing in the format list - `name` - to appear in the double-quoted string. That is, in the above program, **`"{0!s}: "`** means **`"Freddy: "`**. We interpolated one string into another.

Time for some more arithmetic operations: The expression **`x % y`** computes the modulus of x with respect to y – the remainder after dividing x by y.

1 % 3 is 1
2 % 3 is 2
3 % 3 is 0
4 % 3 is 1

Some other math operations we can perform:

`x**2` computes the value of **`x`** raised to the second power.

If we import the "**`math`**" library of functions, we can use the square root function:

```
import math
math.sqrt(x)
```

computes the square root of **`x`**

Program 1: Math operations

The length of the hypotenuse of a right triangle is given by the Pythagorean theorem as the square root of the sum of the lengths of the other two sides. Write a program that will compute the hypotenuse of a right-triangle with the other two sides being 12 and 9 and print out the result.

Program 2: Logic and Decision-making

There are many situations where we would want to choose what parts of a program we want to execute. Try the following program:

```
#
# Python program with basic logic
#
import math
# a pair of values
x = 5
y = -4
#
# arithmetic using x and y
x1 = math.sqrt(x - 6)
y1 = 1 / (y + 4)
#
# print out the results:
print ("x1 is ", x1)
print ("y1 is ", y1)
```

What happens when you run the program? Do the print statements get executed?

In order to avoid having our program "crash", we can introduce control logic to choose what we are going to execute in our programs:

If the condition **x < 6** is true, then do the indented statement(s).

```
x1 = 0
if (x < 6):
    print( "x is less than 6: cannot compute sqrt" )
else:
    x1 = math.sqrt(x - 6)
```

Use 4 spaces before the "**p**" in **print**

If the condition **x < 6** is not true do the statement(s) in the **else** part.

Python is different from other languages in that it assumes you have indented your statements using **4 spaces** to indicate where these statements belong.

In addition to the relational operator "less than" (<), there are a number of others:

```
<=      less than or equal to
>       greater than
>=      greater than or equal to
==      equal to
!=      not equal to
```

Modify the above program ("with basic logic") so that it avoids any problems in computing the y1 variable as well as the x1 variable.

Program 3: Logical operators

Logical operations can be combined into more complex logic using the following 3 operators:

and
or
not

For example:

```
( x > 100 ) and ( x < 150 )
```

This condition will be true if x is between 101 and 149 and false if x is less than or equal to 100 or if x is greater than or equal to 150.

```
( x == 1 ) or not( y == -10)
```

This condition will be true if x is exactly 1 or if y is anything except -10 or if both conditions are satisfied.

String comparison operators

Strings can be compared as well but they are implicitly ordered alphabetically.

```
#
# Python program with string comparison
#
# a pair of strings
x = "ATGCT"
y = "ATGCCA"
#
# compare variables x and y:
if (x < y):
   print( "x is less than y" )
elif (x == y):
   print( "x and y are the same" )
else:
   print( "x is greater than y" )
```

The standard comparison operators listed above work for strings too: they compare the first character of each string and, if they are the same, compare the second character of each, and so on until we run out of characters to compare.

Modify the above program to check if a string **x** is greater than or equal to string **y** first and then do other comparisons. Use different strings than the ones in the above program.

Python resources:
https://www.python.org/
https://docs.python.org/3/tutorial/index.html
https://www.learnpython.org/