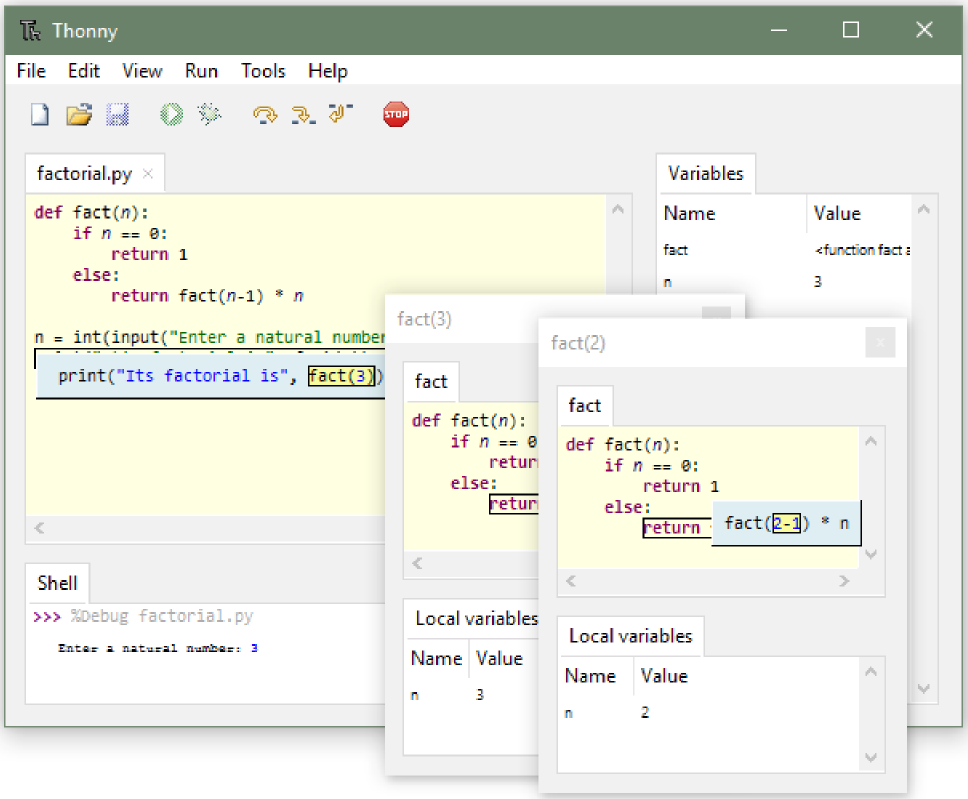
ICS3U: Installing Python and Basic Mathematics

Installing Python

In this course, we will be learning to program using a language called **Python**. It is free software, distributed under an Open Source license, and runs on most modern operating systems, including Windows, Mac OS X, and Linux. Before you can begin programming with Python, however, you need to install it on your computer. This course uses Python 3, not Python 2. While the two versions are very similar, they are *not* fully compatible with each other, so make sure you install the correct version.

Python can be obtained from a number of different sources. My preferred implementation Is Thonny. It comes bundled with Python 3, and is designed for teaching and learning, so it is an excellent tool for beginners. It is available for Windows, Mac and Linux operating systems. Simply click the link to download the appropriate version for your computer. Below is a typical session in Thonny.



```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(n-1) * n

n = int(input("Enter a natural number
print("Its factorial is", fact(3))
```

Name	Value
fact	<function fact
n	3

```
>>> %Debug factorial.py
  Enter a natural number: 3
```

```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(2-1) * n
```

Name	Value
n	3

Name	Value
n	2

Figure 1: Thonny running on Windows.

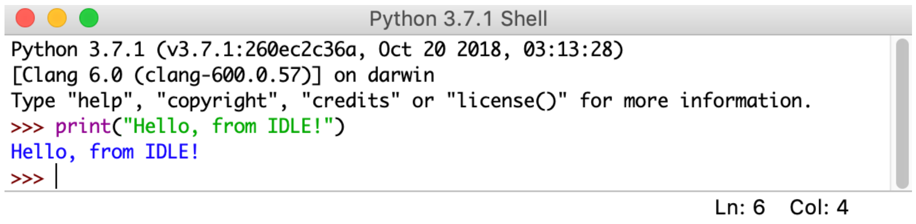
The *official* Python website is at python.org. If you are unable to install Thonny, or just want to stick with the official packages, you can install it and do everything that this course requires. You will, however, miss out on some features that I find are extremely useful when learning how to program for the first time. Releases for all operating systems can be found via the Download menu on the site.

Note: If you are running macOS or a modern Linux distribution (e.g. Ubuntu, Fedora, SuSE, etc.), you may find that Python 2 is already installed on your system. As noted earlier, this version is incompatible with the code that we will be writing in this course. To check if Python is installed on either of these systems, open a terminal and type `python3 --version`. If you see output like `Python 3.x.x`, then Python 3 is installed and you are good to go. If you receive an error, try typing in `python --version`. If you see output like `Python 2.x.x`, then you will need to install the latest version instead.

If you can't (or don't want to) install Python on your computer, or if you want to use a cloud-based version of Python so that yo can access your files from multiple locations, you can try an online service like repl.it. Again, you won't have all of the handy features that Thonny provides, but you will still be able to do everything that this course demands.

The Integrated Development Environment

Most implementations of modern programming languages use an **Integrated Development Environment**, or **IDE**. An IDE provides the programmer with various tools, such as a **code window**, a **compiler** or **interpreter**, a **debugger**, and so on. Thonny's IDE is shown in Figure 1 on the first page of this document. If you are using Python's native IDE (named **IDLE**), it appears slightly differently but includes all of the features mentioned earlier.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello, from IDLE!")
Hello, from IDLE!
>>> 
```

Figure 2: IDLE running on macOS.

Whichever implementation of Python you are using, one of the key areas present in the IDE is the **shell**. This area, usually marked with three chevrons (>>>), is where you can enter individual Python commands to evaluate. While we will enter most of our code in the code window throughout the course, the shell is a good place to do quick calculations or to test a particular command. It also serves as the area in which the user inputs information, and where program output is directed.

Basic Mathematics

Computers excel at mathematical calculations, and Python makes it easy to perform simple operations like addition, multiplication, and so on. To use the Python shell as a calculator, try adding two numbers like in the example below.

```
>>> 3 + 5
8
```

The result of the addition, 8, is shown underneath the prompt. Python will also handle subtraction, multiplication and division.

```
>>> 6 - 9
-3
>>> 2 * 4
8
>>> 7 / 2
3.5
```

Notice that Python is aware of negative values, and can handle decimal numbers when dividing. If you want to do "middle school division", that is find the quotient and remainder, you can do this using the `//` and `%` operators.

```
>>> 7 // 2
3
>>> 7 % 2
1
```

Note that whitespace on a given line in Python generally does not matter, as long as it does not "break up" keywords and operators. For example, both of the following expressions result in the same answer.

```
>>> 8//4
2
>>> 8      //      4
2
```

The following expression, however, causes a problem.

```
>>> 8 / / 4
SyntaxError: invalid syntax
```

A **syntax error** occurs when there is a problem with the way in which a command, or sequence of symbols, is written. In this case, the Python interpreter treats the two slashes as if they are separate division operations. It is equivalent to writing $8 \div \div 4$ in mathematics – it uses proper symbols and values, but does not make sense as a mathematical expression. Syntax errors will prevent the Python interpreter from executing the command. It is common, especially when you first start programming, to encounter syntax errors often. Once you have a better grasp of the language, you may receive fewer of them, but they will still appear once in a while.

Exponentiation is another mathematical operation that is used frequently. Many students, when they first try to use exponents in Python, do something like this.

```
>>> 5^2
7
```

Something is clearly not right here. $5^2 = 5 \times 5 = 25$, not 7. This is an example of a **logical error**. The expression was executed, and an answer was generated, but it was not the correct answer; however, this is not Python's fault! It is a "logical" error because we assumed that the command would do one thing, when in reality, it is supposed to do something else entirely. In Python, the caret symbol (`^`) is not used for exponentiation at all. Python uses a different notation for exponents: two asterisks (`**`). The caret actually does some mathematical comparisons on the individual bits of a binary number, but that is outside the scope of this course to cover. Just remember that exponents use ** instead.

To see the correct answer, type the following.

```
>>> 5**2
25
```

Just like in mathematics, Python obeys the order of operations (BEDMAS or PEMDAS).

```
>>> 6 + 8 / 2
10.0
```

Note that division was performed before addition, as it should have been. We can force the addition to precede the division using brackets.

```
>>> (6 + 8) / 2
7.0
```

Mathematicians often use differently-shaped brackets to improve readability, especially when using nested brackets, as in $[3+(5\times4-1)]$. Unfortunately, using non-round brackets may result in the something like the following.

```
>>> 3+(4*2)
11
>>> 3+[4*2]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    3+[4*2]
TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

This is a third type of error, called a **run-time error**. It is not a syntax error, because the symbols, operators and values are all understood by the interpreter. It is not a logical error, because it did not generate an answer. Instead, the error occured as the command was being executed (run) by the interpreter. Like syntax and logical errors, these occur fairly often. As you practice and develop your programming skills, you should see a decline in the number of errors that occur.

Note that square brackets, [], and curly brackets, {}, *are* used in Python, but for different purposes. We will see how to use some of them later in this course.