Hello, Variables!

What is a program?

A **program** is a sequence of instructions that specifies how to perform a computation.

Every program is comprised of just a few basic instructions:

- **input:**

  Get data from the keyboard, a file, the network, or some other device.
- **output:**

  Display data on the screen, save it in a file, send it over the network, etc.
- **math:**

  Perform basic mathematical operations like addition and multiplication.
- **conditional execution:**

  Check for certain conditions and run the appropriate code.
- **repetition:**

  Perform some action repeatedly, usually with some variation.

Running Python in interactive mode

In Computational Expression, we will use the **Python** programming language.

One way to run Python code is to use the **Python interpreter**, a program that reads and executes Python code.

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

In **interactive mode**, Python code is read and executed line by line.

You can start the Python interpreter on JupyterLab by running `python` in a terminal.

The last line (`>>>`) is a **prompt**. If you type a line of code and hit **Enter**, the interpreter displays the result:

```
>>> 1 + 1
2
```

To exit out of the Python interpreter, type `exit()` and hit **Enter**.

Writing your first program

"Hello, World!" is traditionally the first program you write in a new language. It is called such because all it does is display the words, "Hello, World!".

In Python, it looks like:

```
>>> print('Hello, World!')
Hello, World!
```

This is a **print statement**, which displays a result on the screen. Notice that the quotation marks do not appear in the results.

If you ever get "stuck" within the interpreter (e.g. it continually prints `...` when you hit **Enter** when you don't want it to), hit **CTRL + C** to "break out" of current command.

Arithmetic operators

Python provides **arithmetic operators**, which are special symbols that represent arithmetic computations like addition and multiplication.

Operator	Name
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`**`	Exponentiation

```
>>> ( 2 * 47 - 11 + 1 ) / 2
42.0
```

Arithmetic operators

Notice that in

```
>>> 6 ^ 2
4
```

the result is 4, not 36. This is because in Python, `^` is a bitwise operator called XOR, which we will not cover in Computational Expression. Some other languages may use `^` for exponentiation.

Values and types

A **value** is one of the basic things a program works with, like a letter or a number. `2`, `42.0`, and `'Hello, World!'` are all values.

Each value has a type: `2` is an integer, `42.0` is a floating-point number, and `'Hello, World!'` is a string, so-called because the letters it contains are strung together.

The interpreter can tell you the type of a value:

```
>>> type('Hello, World!')
<class 'str'>
```

Here, the word "class" is used like the word "category"—a type is a category of values.

Some values may look like numbers, but if they are in quotation marks, they are strings.

```
>>> type(42.0)
<class 'float'>
>>> type('42.0')
<class 'str'>
```

Revisiting division

While adding, subtracting, and multiplying integers gives integers results

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
```

dividing integers gives a floating-point number result:

```
>>> 84 / 2
42.0
```

This is because in Python all results of division are typed as floating-point numbers.

Variables

A **variable** is a name that refers to a value.

An **assignment statement** creates a new variable and gives it a value:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897932
```

Choose a meaningful name for your variable—it should document what the variable is used for.

```
>>> n = 'Maria'
>>> first_name = 'Maria'
```

```
>>> num = 3
>>> num_of_apples = 3
```

Variables

Naming rules

- Can contain letters, numbers, and underscores (`_`)
- Must start with either a letter or underscore
- Are case-sensitive

```
76trombones = 'big parade'
```

`SyntaxError: invalid syntax`: Starts with number

```
more@ = 1000000
```

`SyntaxError: invalid syntax`: `@` is not a letter, number, or underscore

```
class = 'Advanced Theoretical Zymurgy'
```

`SyntaxError: invalid syntax`: `class` is a **Python keyword**

Python keywords

Python reserves these keywords to recognize the structure of the program.

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

They cannot be used as variable names. But, they can be *in* a variable name.

```
>>> class_name = 'Computational Expression'
```

No need to memorize them!

Expressions and statements

An **expression** is a combination of values, variables, and operators.

```
>>> 112
112
>>> x
-520
>>> x + 112
-408
```

A **statement** is a unit of code that has an effect, like creating a variable or displaying a value.

```
>>> n = 17
>>> print(n)
```

In general, statements don’t have values.

Running Python in script mode

To write programs longer than a few lines of code, it is easier to save the program in a Python **script** (`.py`) and to run the interpreter in **script mode** to execute the script.

1. Write code in script, such as `hello_world.py`
2. Execute script with interpreter by running `python <PATH-TO-SCRIPT>`, e.g. `python hello_world.py`

Generally, you use interactive mode to *test* bits of code and *write* programs in scripts to run in script mode.

In interactive mode, expressions are evaluated and their results are displayed:

```
>>> miles = 26.2
>>> miles * 1.61
42.182
```

However, in script mode, expressions have no visible effect. You need to use a print statement to display their results:

```
miles = 26.2
print(miles * 1.61)
```

Assignment operators

Often, you want to add or subtract from a variable's current value.

```
count = count + 1
count = count - 1
```

You can also write these statements using the `+=` and `-=` assignment operators.

```
count += 1
count -= 1
```

Though both ways of writing these statements are *valid*, the latter is more concise and easier to read.

Debugging and types of errors

Programmer errors are referred to as **bugs**. Tracking down these errors is known as **debugging**.

Syntax error:
"Syntax" refers to the rules about the structure of a program that are defined by the programming language. A syntax error occurs when a rule is violated.

```
>>> print'Hello, World!')
  File "<stdin>", line 1
    print'Hello, World!')
         ^
SyntaxError: invalid syntax
```

Debugging and types of errors

Runtime error:
Runtime errors occur after a program has started running. They are also referred to as **exceptions** because they indicate that something exceptional has occurred. They are rare in simpler programs.

Semantic error:
The word "semantic" means related to meaning and semantic errors occur when your code does not do what you *meant* it to do. Your code will run without generating error messages, but it will not do the right thing.

What kind of error was made when you tried to define a large integer value using commas (1,000,000), but the value was interpreted as a comma-separated sequence of integers instead?

Identifying the type of error can help you track down bugs more quickly.