# 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'>
```

## Values and types

You may be tempted to use commas to improve readability in large integers, such as `` `1,000,000` ``. However, this is not a legal *integer* in Python:

```
>>> 1,000,000
(1, 0, 0)
```

Python interprets `` `1,000,000` `` as a comma-separated sequence of integers

## 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, even integer division, are typed as floating-point numbers.