Python Coding Camp – Session 1

The Python Shell

The development environment we will be using to write Python programs is called **IDLE**, which stands for Integrated DeveLopment Environment.

When you start up IDLE, it should pop up a window containing a Python **shell**, which is an interactive application that allows you to test out small pieces of Python programs. In the shell it should display information about which version of Python is running on your system along with a **prompt** that looks like this:

```
>>>
```

The prompt lets you know where to type your Python code.

The Python shell can do two different things:

1. Evaluate **expressions**.
2. Execute **statements**.

Expressions

An expression is a piece of code that can produce a value. If you type a Python expression in the shell and hit Enter, it will attempt to compute a value based on the rules of the language. Some examples (NOTE: any time there are examples you should type them in yourself in the Python shell to see what happens):

```
>>> 24
24
>>> 3 + 5
8
>>> 2.5 * 8.7
21.75
>>> "hello"
'hello'
>>> "Bat" + "man"
'Batman'
>>> True
True
```

Not that in some cases – such as `24`, `"hello"`, and `True` – the value of the expressions is the same as the expression itself or at least pretty close. These kinds of expressions are called **constants** because they don't change or **literals** because they mean exactly what they say. The other expressions use **operators** to produce a new value from the **operands** on either side.

Not every combination of constants and operators works. For example:

```
>>> "number" + 9
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    "number" + 9
TypeError: must be str, not int
```

Python can't evaluate this expression because there is a mismatch between the operands and the operator. When the Python shell can't perform an operation it displays an error message to the user. These messages will make more sense when we start writing Python programs.

Python supports several different types of expressions. To determine the type of an expression, you can use the `type` function:

```
>>> type(42)
<class 'int'>
>>> type(3.14159)
<class 'float'>
>>> type("hello")
<class 'str'>
>>> type(False)
<class 'bool'>
```

These are all examples of some of the basic types provided by Python:

- The `int` type refers to **integers** or whole numbers. They can be positive or negative or zero, but they can't have a fractional part.
- The `float` type refers to **floating point numbers**, which are a particular way of representing numbers that can have a fractional part.
- The `str` type refers to **strings**, which are sequences of **characters** in between a pair of single quotes or double quotes.
- The `bool` type refers to **Booleans**, which can be either `True` or `False`. A character can be a letter, a digit, a punctuation symbol, a blank space, or a number of other special symbols.

Python supports several operations on numbers.

The `+`, `-`, `*`, `/` , and `**` operators represent addition, subtraction, multiplication, division, and exponentiation.

```
>>> 45 + 72
117
>>> 37 - 12
25
>>> 2.5 * 67
167.5
>>> 43 / 6
7.166666666666667
>>> 2 ** 8
256
```

The `//` and `%` operators calculate the quotient and remainder that result from dividing one integer by another.

```
>>> 87 // 12
7
>>> 87 % 12
3
```

Some of these operations can also be used on strings.

The + operator can be used to **concatenate** two or more strings:

```
>>> "strong" + "bad"
'strongbad'
>>> "home" + "star" + "runner"
'homestarrunner'
```

The * operator can be used to concatenate zero or more instances of the same string:

```
>>> "ding" * 5
'dingdingdingdingding'
>>> "ding" * 0
''
```

In the second example, we ended up with a string with no characters in it. This is called the **empty string**

Statements

A statement is a piece of code that performs an action.

One of the most common actions that is performed in a program is **assigning** the value produced by an expression to a **variable**. The variable can then be used later to refer to that value.

In Python the = operator is used for assignment:

```
>>> name = "Inigo Montoya"
>>> "My name is " + name
'My name is Inigo Montoya'
>>> total = 45 + 25 + 72
>>> total
142
```

Variable names may only contain letters, numbers and _s and may not start with a number. So `number1` is legal, but `1number` is not. Conventionally, Python variables usually start with a lowercase letter and only use uppercase letters to separate words. For example, `firstName` and `lastName` might be used as variable names in a Python program.

Remember that the variable must always be on the left-hand side of the = operator, otherwise the assignment doesn't work:

```
>>> height = 25.5
>>> 25.5 = height
SyntaxError: can't assign to literal
```

Using variables in expressions does not change the value of the variable:

```
>>> value = 45
>>> value * 3
135
>>> value
45
```

Try creating your own expressions and statements based on what you have learned to far.