Chicago Python Workshop 1:

Saturday Morning Review

Adapted from the Boston Python Workshop 6 **Friday Night Tutorial.**

Math

Arithmetic expressions can be typed directly into a Python prompt, e.g.:

```
>>> 2 + 2
>>> 1.5 + 2.25
>>> 4 - 3
>>> 2 * 3
>>> 4 / 2
>>> 1 / 2
```

Math

Order of operations works the way you'd expect in arithmetic. Use parens for readability (or when in doubt).

```
>>> 2 * 2 - 3 * 1
1
>>> (2 * 2) - (3 * 1)
1
```

Integers

Integers are (positive or negative) whole numbers numbers.

Integer division produces an integer:

```
>>> 2 / 1
2
```

Floats

Floats, or "floating point" numbers, are "real" or decimal numbers.

Use a decimal point to get a decimal result from division:

```
>>> 1.0 / 2
```

Types

Use the `type()` function to find out what *data type* Python thinks an object is:

```
>>> type(1)
>>> type(1.0)
```

Variables

```
>>> type(4)
>>> x = 4
>>> x
>>> type(x)
>>> 2 * x
```

Strings

A *string* is a data type containing sequence of characters, e.g.:

```
   "Hello"
   "What the @#(*$!? is a string?"
   "A string is some characters strung
together!"
```

Printing Strings

Use `print` to output strings to the screen.

```
>>> h = "Hello"
>>> w = "World"
>>> print h + w
```

String Concatenation

Strings can be concatenated, or put together, using '**+**':

```
>>> h = "Hello "
>>> w = "World!"
>>> print h + w
```

String Length

Use `len()` to find the length of a string.

```
   >>> len("Hello")
   >>> len("")
   >>> fish = "tuna"
   >>> length = str(len(fish))
   >>> print fish + " is a fish whose
name is " + lenth + " characters
long."
```

Quotes

Python doesn't distinguish between single (') and double (") quotes, but there are some things to keep in mind:

```
   >>> print "I'm a happy camper"
   >>> print 'Ada Lovelace is often called the
world\'s first programmer.'
  >>> print "Computer scientist Grace Hopper
popularized the term \"debugging\"."
```

Scripts

You can run Python scripts from the commandline:

```
$ python nobel.py
```

Comments in the script can be indicated with `#` as the first non-whitespace character.

Notice the *interpreter directive* on the first line.

Booleans

The boolean Python data type has two values, `True` and `False`.

```
   >>> type(True)
   >>> 0 == 0
   >>> 2 >= 3
   >>> "Perl" not in "Chicago Python
Workshop"
```

Flow Control

Use flow control to tell Python how to proceed based on certain conditions.

Python *evaluates an expression* and executes code based on the results.

`if` statements

Note the colon following the `if` statement, and the indentation within the "code block".

```
  >>> if 6 > 5:
  ...   print "Six is greater than
five!"
```

and and or

```
>>> "a" in "hello" or "e" in "hello"
>>> 1 <= 0 or "a" not in "abc"
>>> temperature = 32
>>> if temperature > 60 and
>>> temperature < 75:
...     print "It's nice and cozy in here!"
>>> else:
...     print "Too extreme for me."
```

`elif`

Use elif to extend your if statement to handle more cases.

You can have as many `elif` cases as you want; Python will check each `elif` until it finds a `True` condition or reaches the default `else` block.

Functions

- They do some useful bit of work.

Functions

- They do some useful bit of work.
- They let us re-use code.

Functions

- They do some useful bit of work.
- They let us re-use code.
- They take input and possibly produce output (*return* a value). You can assign a variable to this output.

Functions

- They do some useful bit of work.
- They let us re-use code.
- They take input and possibly produce output (*return* a value). You can assign a variable to this output.
- You call a function by using its name followed by its *arguments* in parenthesis.

Functions

```
>>> def add(x, y):
...     result = x + y
...     return result
>>> add(18,24)
```

END OF PART 1 PRACTICE!