Part 02 - Variables

Part 02 - Variables

Using Booish as a Calculator

There are four basic mathematical operators: addition `+`, subtraction `-`, multiplication `*`, and division `/`. There are more than just these, but that's what will be covered now.

```
$ booish
>>> 2 + 4 // This is a comment
6
>>> 2 * 4 # So is this also a comment
8
>>> 4 / 2 /* This is also a comment */
2
>>> (500/10)*2
100
>>> 7 / 3 // Integer division
2
>>> 7 % 3 // Take the remainder
1
>>> -7 / 3
-2
```

You may have noticed that there are 3 types of comments available, `//`, `#`, and `/* */`. These do not cause any affect whatsoever, but help you when writing your code.

Recommendation
When doing single-line comments, use `//` instead of `#`

You may have noticed that 7 / 3 did not give 2.333..., this is because you were dividing two integers together.

Definition: Integer
Any positive or negative number that does not include a fraction or decimal, including zero.

The way computers handle integer division is by rounding to the floor afterwards.

In order to have decimal places, you must use a floating-point number.

Definition: Floating=point Number
Often referred to in mathematical terms as a "rational" number, this is just a number that can have a fractional part.

```
$ booish
>>> 7.0 / 3.0 // Floating point division
2.33333333333333
>>> -8.0 / 5.0
-1.6
```

If you give a number with a decimal place, even if it's .0, it become a floating-point number.

Types of Numbers

There are 3 kinds of floating point numbers, `single`, `double`, and `decimal`.
The differences between `single` and `double` is the size they take up. `double` is prefered in most situations.
These two also are based on the number 2, which can cause some problems when working with our base-10 number system.
Ususally this is not the case, but in delecate situations like banking, it would not be wise to lose a cent or two on a multi-trillion dollar contract.
Thus `decimal` was created. It is a base-10 number, which means that we wouldn't lose that precious penny.
In normal situations, `double` is perfectly fine. For a higher precision, a `decimal` should be used.

Integers, which we covered earlier, have many more types to them.
They also have the possibility to be "unsigned", which means that they must be non-negative.
The size goes in order as such: `byte`, `short`, `int`, and `long`.
In most cases, you will be using `int`, which is the default.

Characters and Strings

> **Definition: Character**
> A written symbol that is used to represent speech.

All the letters in the alphabet are characters. All the numbers are characters. All the symbols of the Mandarin language are characters. All the mathematical symbols are characters.
In Boo/.NET, characters are internally encoded as UTF-16, or Unicode.

> **Definition: String**
> A linear sequence of characters.

The word "apple" can be represented by a `string`.

```
$ booish
>>> s = "apple"
'apple'
>>> print s
apple
>>> s += " banana"
'apple banana'
>>> print s
apple banana
>>> c = char('C')
C
>>> print c
C
```

Now you probably won't be using `chars` much, it is more likely you will be using `strings`.
To declare a `string`, you have one of three ways.

1. using double quotes. "apple"
2. using single quotes. 'apple'
3. using tripled double quotes. """apple"""
   The first two can span only one line, but the tribbled double quotes can span multiple lines.
   The first two also can have backslash-escaping. The third takes everything literally.

```
$ booish
>>> print "apple\nbanana"
apple
banana
>>> print 'good\ttimes'
good	times
>>> print """Leeroy\Jenkins"""
Leeroy\Jenkins
```

Common escapes are:

- {{
  }} literal backslash
- `\n` newline
- `\r` carriage return
- `\t` tab
  If you are declaring a double-quoted `string`, and you want a double quote inside it, also use a backslash.
  Same goes for the single-quoted `string`.

```
$ booish
>>> print "The man said \"Hello\""
The man said "Hello"
>>> print 'I\'m happy'
I'm happy
```

`strings` are immutable, which means that the characters inside them can never change. You would have to recreate the `string` to change a character.

Definition: Immutable
Not capable of being modified after it is created. It is an error to attempt to modify an immutable object. The opposite of immutable is mutable.

String Interpolation

String interpolation allows you to insert the value of almost any valid boo expression inside a `string` by preceeding a lonesome variable name with $, or quoting an expression with $().

```
$ booish
>>> name = "Santa Claus"
Santa Claus
>>> print "Hello, $name!"
Hello, Santa Claus!
>>> print "2 + 2 = $(2 + 2)"
2 + 2 = 4
```

String Interpolation is the preferred way of adding `strings` together.
It is preferred over simple `string` addition.

String Interpolation can function in double-quoted `strings`, including tripled double-quoted `string`.
It does not work in single-quoted `strings`.

To stop String Interpolation from happening, just escape the dollar sign: \${}

Booleans

> **Definition: Boolean**
> A value of `true` or `false` represented internally in binary notation.

Boolean values can only be `true` or `false`, which is very handy for conditional statements, covered in the next section.

```
$ booish
>>> b = true
true
>>> print b
True
>>> b = false
false
>>> print b
False
```

Object Type

> **Definition: Object**
> The central concept in the object-oriented programming programming paradigm.

Everything in Boo/.NET is an object.
Although some are value types, like numbers and characters, these are still objects.

```
$ booish
>>> o as object
>>> o = 'string'
'string'
>>> print o
string
>>> o = 42
42
>>> print o
42
```

The problem with `objects` is that you can't implicitly expect a `string` or an `int`.
If I were to do that same sequence without declaring `o as object`,

```
$ booish
>>> o = 'string'
'string'
>>> print o
string
>>> o = 42
--------^
ERROR: Cannot convert 'System.Int32' to 'System.String'.
```

This static typing keeps the code safe and reliable.

Declaring a Type

In the last section, you issued the statement `o as object`.
This can work with any type and goes with the syntax `<variable> as <type>`.
`<type>` can be anything from an `int` to a `string` to a `date` to a `bool` to something which you defined yourself, but those will be discussed later.

In most cases, Boo will be smart and implicitly figure out what you wanted.
The code `i = 25` is the same thing as `i as int = 25`, just easier on your wrists.

Recommendation
Unless you are declaring a variable beforehand, or declaring it of a different type, don't explicitly state what kind of variable it is.
i.e.
use `i = 25` instead of `i as int = 25`

List of Value Types

Boo type	.NET Framework type	Signed?	Size in bytes	Possible Values
sbyte	System.Sbyte	Yes	1	-128 to 127
short	System.Int16	Yes	2	-32768 - 32767
int	System.Int32	Yes	4	-2147483648 - 2147483647
long	System.Int64	Yes	8	-9223372036854775808 - 9223372036854775807
byte	System.Byte	No	1	0 - 255
ushort	System.Uint16	No	2	0 - 65535
uint	System.UInt32	No	4	0 - 4294967295
ulong	System.Uint64	No	8	0 - 18446744073709551615
single	System.Single	Yes	4	Approximately ±1.5 x 10-45 - ±3.4 x 1038 with 7 significant figures
double	System.Double	Yes	8	Approximately ±5.0 x 10-324 - ±1.7 x 10308 with 15 or 16 significant figures
decimal	System.Decimal	Yes	12	Approximately ±1.0 x 10-28 - ±7.9 x 1028 with 28 or 29 significant figures
char	System.Char	N/A	2	Any UTF-16 character
bool	System.Boolean	N/A	1	true or false

Recommendation
Never call a type by its .NET Framework type, use the builtin boo types.

Exercises

1. Declare some variables. Go wild.

Go on to **Part 03 - Flow Control - Conditionals**