Part 17 - Macros

Part 17 - Macros

print Macro

The `print` Macro will display one or more objects to the screen.

There are two ways to call the `print` macro.

1. With only one argument
2. With two or more arguments

print Example

```
print "Hello there"
print "Hello", "there"
```

Output

```
Hello there
Hello there
```

In the second case, for every case except the last, it will write the string to the screen, write a space, then move on.

In the end, the two will have the same end result.

assert Macro

The `assert` Macro makes sure that a condition is true, otherwise it raises an `AssertionFailedException`.

`assert` can be called with one or two arguments.
The first argument must always be a boolean condition.
The optional second argument is a string that will be sent if the condition fails.

assert Example

```
assert true // this will always pass
assert false, "message" // this will always fail
```

Output

```
Boo.Lang.Runtime.AssertionFailedException: message
   at Tutorial.Main(String[] argv)
```

Recommendation
Never `assert` a condition that would, in itself, change your code.
e.g. `assert iter.MoveNext()` would be a bad idea.

using Macro

The `using` Macro can take any number of arguments, it merely duplicates its behavior each time.

It creates a safety net for objects to be handled during a block, then disposed of as soon as that block is finished.

There are three types of arguments you can declare:

1. `<object>`
2. `<object> = <expression>`
3. `<expression>`

In all three of these, it checks if the underlying `object` is an `IDisposable`, which it then disposes of afterward.

using Example

```
import System.IO

using w = StreamWriter("test.txt"):
    w.WriteLine("Hello there!")
```

This will create the file, write to it, then close it as soon as the `using` block is finished. Makes it very safe and convenient.

lock Macro

The `lock` Macro makes sure that, in a multithreaded environment, that a specified object is not being used and prevents another object from using it at the same time.

`lock` must accept at least one argument, and it will put the `lock` on all that are given.

lock Example

```
lock database:
    database.Execute("""
        UPDATE messages
        SET
            id = id + 1""")
```

debug Macro

The `debug` Macro is the exact same as the `print` Macro, except that it sends its messages to `System.Diagnostics.Debug` instead of `System.Console`.

Go on to **Part 18 - Duck Typing**