Topic 1.2: Structure and Formatting

Hello World

It is a time-honoured tradition to start learning a new programming language by writing a "`hello world`" program. In Java, this simple program requires a class definition, a main method, and a fair amount of syntax:

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.print("Hello World!");
    }

}
```

Study the code for the equivalent program written in Python:

```
print("Hello world!")
```

Yes, really, that's all that is required. No class definitions. No main method. No access modifiers. No semicolons. I hope that this introduction to Python will encourage you that it'll be worthwhile to learn Python.

Python Interpreter: the Read-Evaluate-Print Loop (REPL)

Python has a command line interpreter that executes code as it is typed. Each line is immediately evaluated, and the result is printed (unless it is `None`). This is invaluable for experimentation, testing small code snippets, and learning. Variables persist throughout the session, allowing incremental exploration.

Assuming you have successfully downloaded and installed the Python interpreter application, Follow the instructions to run the Python REPL using the operating system's command line interface.

On Macintosh computers, this can be done by running the application called `Terminal`. After opening `Terminal`, type `python3` on the command line.

On a PC, it may be called `Command Prompt` or `Power Shell`. After opening one of those applications, type `python` on the command line.

If the above instructions don't work, perhaps the work flow has changed since the time of writing the instructions, and more current and more detailed instructions to start the Python REPL can be found online.

Examine the example use of the Python REPL below, and try the same or similar commands on your computer.

```
% python3

Python 3.12.0 (v3.12.0:0fb18b02c8, Oct 2 2023, 09:45:56)
[Clang 13.0.0 (clang-1300.0.29.30)] on mycomputer

>>> print("Hello world!")

Hello world!

>>> 2 + 2

4

>>> exit()
```

Script Execution

For anything larger than a few lines, Python code is saved in files with a `.py` extension. These files are often referred to as Python scripts. These are plain text files executed by the Python interpreter:

```
python my_script.py
```

Unlike Java, there is no separate compilation step. The Python interpreter reads the source file, compiles it internally to bytecode, and executes it immediately. The `.pyc` cache files that sometimes appear in a `__pycache__` directory are an optimization for faster startup, not a required intermediate artifact like Java's `.class` files.

Write a few commands in a plain text file on your computer, and save the file with the `.py` extension. Ensure you save the file as *plain text*; Macintosh computers may default to saving text in *rich text format*, which adds additional formatting characters to the file that will confuse the Python interpreter.

When using the Python REPL, the result of each evaluated expression is printed to the console. This does not happen when the Python script is interpreted by the Python interpreter. In order to print to the console, the script must explicitly tell the interpreter to output to the console using the `print` function. Study the following python script and the resulting output.

Contents of the file: `helloworld.py`

```
1 print("Hello World")
2 2 + 2
3 print("3 + 3 =", 3+3)
4 print("Bye, for now...")
```

Run the script using: `python3 helloworld.py`

```
Hello World
3 + 3 = 6
Bye, for now...
```

Notice that the code on line 2 of the Python script does not result in any output to the console because the interpreted was not told to output anything to the console.

By default, the Python `print` function writes its arguments to the console, adding a newline character at the end. Multiple arguments can be passed, separated by commas, and print will insert spaces between them:

Python:

```
print("The answer is", 42)
```

Output:

```
The answer is 42
```

Unlike Java, Python will not automatically concatenate an integer to a string. Verify this by attempting the following, either in a Python script, or using the Python REPL.

Python:

```
print("The answer is" + 42)
```

More details about the `print` function will be given later.

Indentation: Code Blocks Without { Curly Braces }

Java uses curly braces, `{` and `}`, to delimit blocks of code. the body of a class, each method, any loop, or conditional. Indentation in Java is only for readability by human readers.

```
boolean isPositive = false;
if (x > 0) {
    System.out.println("Positive");
    isPositive = true;
}
System.out.println("Outside the block");
```

Python uses indentation.

```
is_positive = False
if x > 0:
    print("Positive")
    is_positive = True
print("Outside the block")
```

The colon, `:`, introduces a new block, and all lines indented at the same level belong to that block. The block ends when the indentation returns to the previous level.

The rules for indentation are:

- Indentation must be consistent within a block; spaces and tabs cannot be mixed.
- Four spaces is the convention, though any consistent number within a block works.
- Empty lines are ignored; indentation level is determined by the next non-empty line.

This example has an indentation error:

```
if x > 0:
    print("Positive")
  print("Still positive")  # IndentationError: unexpected indent
```

For programmers accustomed to braces, this takes some adjustment. However, consistent indentation is widely regarded as improving readability, and Python simply enforces what many coding standards already require.

Comments and Continuation

Did you notice the comment in the last block of code? In Python, comments begin with # and continue to the end of the line. Here are two more examples:

```
# This is a comment
x = 10  # This is also a comment
```

Long lines can be continued with a backslash `\` at the end, or more commonly, by using parentheses, brackets, or braces, which implicitly continue until closed:

```
Total = (1 + 2 + 3 + 4 + 5 +
         6 + 7 + 8 + 9 + 10)
```

Multi-Line Strings are NOT Comments

Multi-line strings are enclosed within triple quotes (`"""` or `'''`) and create string literals, not comments.

```
"""
This looks like a comment block,
but it's actually a string.
"""
```

Python will create a string object in memory, but if this string is not assigned to a variable, it is immediately discarded. This behaves like a comment in that it doesn't affect program execution, but it is semantically different.

When a multi-line string is placed as the first statement in a module, class, or function, it becomes a `docstring` – a special attribute accessible via `help()` and `__doc__`.

```
def add(a, b):
    """This is a proper docstring.
    It documents what this function does.
    It can be accessed programmatically."""

    return a + b

print(add.__doc__)  # This works
```

By all means, document your Python code with proper `docstring`, but it is bad practice to use multi-line strings for comments. Use only the # symbol.