About Python I

Python is interpreted

Interpretation

In an interpreted system, program instructions are executed inside a program called an **interpreter**. The interpreter examines and executes a program's source code *instruction-by-instruction*. In other words, the “translation” from source code into machine code is done in real-time by the interpreter while the program is “running”. If the interpreter encounters a syntax or runtime error during this process, your program stops running and the interpreter tries to tell you what went wrong.

Without the right interpreter installed on your system, you cannot run interpreted language programs.

Compilation

In a compiled system, the translation from source code into machine code is done *all at once* by a program called a **compiler**. Often times the translated code must be connected to other modules in system libraries and such. This connection to library modules is done by a program called a **linker**. The result of a successful compile-and-link process is an executable machine-code module that you can run directly as a standalone program.

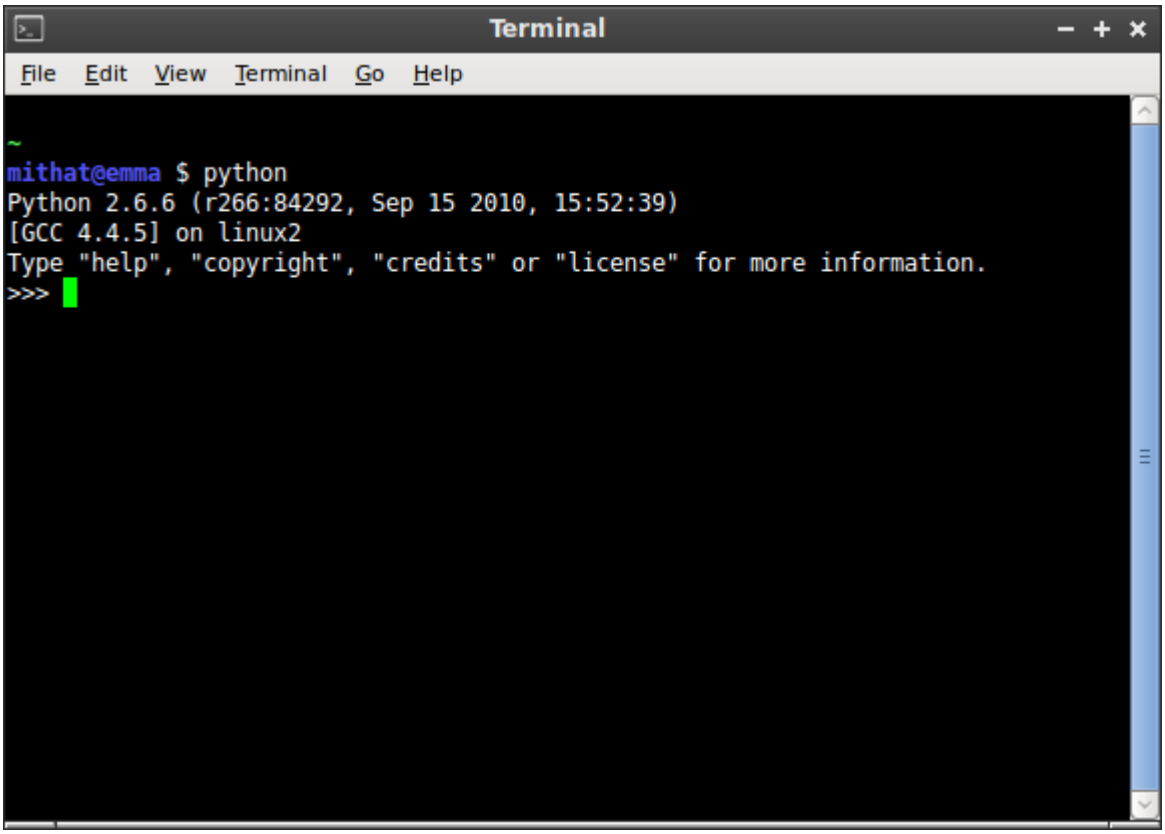
If there are syntax errors in the source code, the compiler will fail to produce a machine-code translation of the program. This means that it will be impossible for you to link and then execute it. Almost all compilers will give you information about the nature of the error to help you debug the program.

Once the executable module is made, you do not need a compiler or linker to run the program.

Exercise: Python in interactive mode

The name of the command to start the Python interpreter is `python`. If you start the Python interpreter without any additional arguments, it will start in **interactive mode**. In interactive mode, you can enter program statements one-by-one and watch Python interpret them.

1. Open a shell (i.e., a “terminal emulator” in Linux and OS X or a “command window” in Windows.)
2. At the command prompt, type `python` and hit the `<Enter>` key.
3. If Python was installed correctly on your system, you should see something that looks like this:



The >>> string is Python's *interactive-mode prompt*. At this prompt, you can type Python commands and the interpreter will run them.

4. At the >>> prompt, type:

```
7 + 3
```

and hit the <Enter> key. What do you see?

5. Enter:

```
7 - 3
```

What do you see?

6. Enter:

```
7 * 3
```

What do you see?

7. Enter:

```
7 / 3
```

What do you see? Why?

8. Enter:

```
7.0 / 3.0
```

What do you see? Why?

9. Enter:

```
7 % 3
```

What do you see? Why?

10. Enter:

```
print "Hello world!"
```

What do you see?

11. Enter:

```
print "Seven plus three is: ", 7 + 3
```

What do you see?

To exit the Python interpreter, type `Control-D` or type `exit()` and hit the `<Enter>` key.

Python is scriptable

Using Python in interactive mode is great if all you want is a super-calculator.[1)] However, the true power of Python is realized when you use it to run Python *scripts*. A Python script is a normal text file that contains a set of Python instructions. Typically, Python scripts use the `.py` extension. Comments begin with #.

An example of such a script appears below.

```
# A sample Python script
print "Hello. Watch as I compute a value:"
print "(2012 - 13) / 3 is", (2012 - 13) / 3
print "Evil!!!"
```

If you provide an argument to the `python` command that points to a script file, the Python interpreter will execute the statements in the file as though you typed them in manually at the interactive prompt.

Exercise: Running a Python script

1. Create a text file called `evil.py` and copy the code above into the file. Save the file.

2. Open a shell (i.e., “terminal emulator” or “command window”).
3. Navigate to the directory where you saved `evil.py`. (Use the Windows ''cd'' command or the Linux/OS X ''cd'' command and be careful about spaces!)
4. At the command prompt, type `python evil.py` and hit the <Enter> key
5. Cower in fear.

Python is typed

Values in Python are typed. Common simple types include `int`, `float`, `long`, `complex`, `str`, and `bool`. (Full details about all of Python's built-in types can be found here). You can determine the type of just about anything with the `type()` function.

Exercise: Types (with literals)

1. Start Python in interactive mode.
2. Enter the following commands and observe the results:

```
type(66)
type(3.15189)
type(3 + 5j)
type("Hello world")
type('Hello world')
type("a")
type('a')
type(False)
type(True)
type(true)
```

You should notice a few important things:

- You can wrap string literals in either `'single quotes'` or `"double quotes"`.
- Single characters are simply strings with one character in them. There is no Python equivalent to C and Java's `char` type.
- Python's Boolean values are `True` and `False`.
- Python is case sensitive.

Variables

Variables are created in Python automatically when they are first used (i.e., when they are first assigned a value). Python's assignment operator is =

Identifiers can begin with any lowercase letter, uppercase letter, or the underscore character and be followed with any number of uppercase letters, lowercase letters, digits, or underscore characters.

Traditionally, variable names begin with a lowercase letter and words are separated with underscores.

Exercise: Variable creation and assignment

Execute the following statements in an interactive session and observe the results.

```
a = 66
print a
type(a)

b = 1.5708 * 2.0
print b
type(b)

c = 3 + 5j
print c
type(c)

foo = "Hello world"
print foo
type(foo)

goo = 'Hello world'
print goo
type(goo)

bar = "a"
print bar
type(bar)

baz = 'a'
print baz
type(baz)

is_it = False
print is_it
type(is_it)

to_be = True
print to_be
type(to_be)
```

Python's typing is dynamic

The types of Python variables change as needed to permit the assignment.

Exercise: Dynamic typing

Execute the following statements in an interactive session and observe the results.

```
x = 3
type(x)

x = 3.0
type(x)

x = "reassign float to str"
type(x)
```


1)
It also turns out to be very useful for debugging and testing ideas.