**Working with Python Basic Syntax**

Running Interactively on UNIX

$python (for python 2.x and python3 for python 3.x)

>>>3+3

6

Python prompts with ‘>>>’.

To exit Python:

In Unix, type CONTROL-D

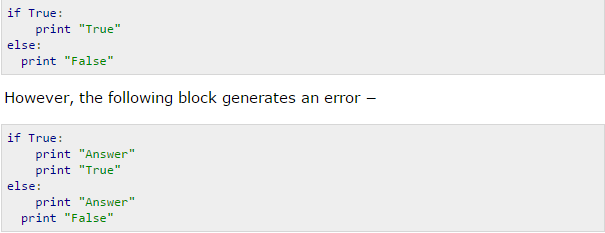
In Windows, type CONTROL-Z + <Enter>

**Lines and Indentation**

There are no braces to indicate blocks of code for class and function definitions or flow control.

Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.



**Multiple Statement Groups as Suites**

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite.

For example −

if expression :

suite

elif expression :

suite

else :

suite

**Multi-Line Statements**

* Statements in Python typically end with a new line.
* Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.
* For example:

total = item\_one + \

item\_two + \

item\_three

* Statements contained within the [], {} or () brackets do not need to use the line continuation character.
* For example:

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

**Quotation in Python**

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes can be used to span the string across multiple lines.

word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

**Running Programs on UNIX**

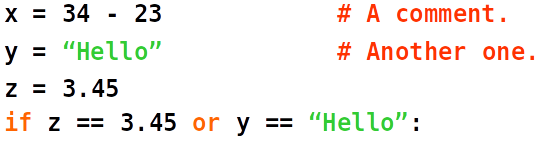
$python filename.py

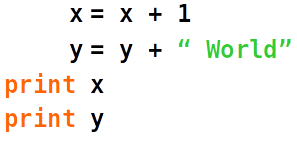
You can create python files using emacs.

To make a python file executable, make this text the first line of the file :

#!/usr/bin/python

A Code Sample





**Enough to Understand the Code**

* Indentation matters to the meaning of the code:
  + Block structure indicated by indentation
* The first assignment to a variable creates it.
  + Variable types don’t need to be declared.
  + Python figures out the variable types on its own.
* Assignment uses = and comparison uses ==.
* For numbers + - \* / % are as expected.
  + Special use of + for string concatenation.
* Logical operators are words (and, or, not), not symbols
* Simple printing can be done with print.

**Python Identifiers:**

* A Python identifier is a name used to identify a variable, function, class, module or other object.
* An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).
* Python does not allow punctuation characters such as @, $ and % within identifiers. Python is a case sensitive programming language.
* Thus, Manpower and manpower are two different identifiers in Python.

**Reserved Words:**

These reserved words may not be used as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and exec not assert finally or

break for pass class from print

continue global raise def if return

del import try elif in while

else is with except lambda yield

**input() and raw\_input()**

In python 2.x, raw\_input() returns a string and input() evaluates the input in the execution context in which it is called.

>>> x = input()

"hello"

>>> y = input()

x + " world"

>>> y

'hello world'

In python 3.x, input has been scrapped and the function previously known as raw\_input is now input. So you have to manually call compile and than eval if you want the old functionality.

python2.x python3.x

raw\_input() --------------> input()

input() -------------------> eval(input())

In 3.x, the above session goes like this

>>> x = eval(input())

'hello'

>>> y = eval(input())

x + ' world'

>>> y

'hello world'

>>>

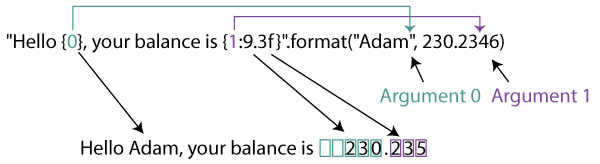
So you were probably getting an error at the interpreter because you weren't putting quotes around your input. This is necessary because it's evaluated.

**format() # do not take notes for this topic,**

str.format() is one of the ***string formatting methods*** in Python3, which allows multiple substitutions and value formatting. This method lets us concatenate elements within a string through positional formatting.

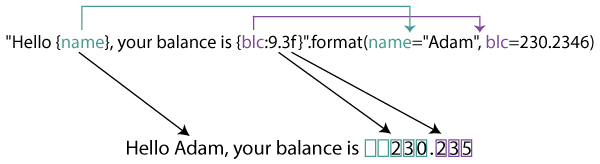
The format() reads the type of arguments passed to it and formats it according to the format codes defined in the string,

For positional arguments

****

Here, Argument 0 is a string "Adam" and Argument 1 is a floating number 230.2346.

### For keyword arguments

****

We've used the same example from above to show the difference between keyword and positional arguments.

Here, instead of just the parameters, we've used a key-value for the parameters. Namely, name="Adam" and blc=230.2346.

Since, these parameters are referenced by their keys as {name} and {blc:9.3f}, they are known as keyword or named arguments.

#program for format and import  
a='bread'  
b='butter'  
print('I love {} and {}'.format(a,b))  
print('I love {0} and {1}'.format('bread','butter'))  
print('I love {1} and {0}'.format('bread','butter'))  
#using keyword arguments  
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))

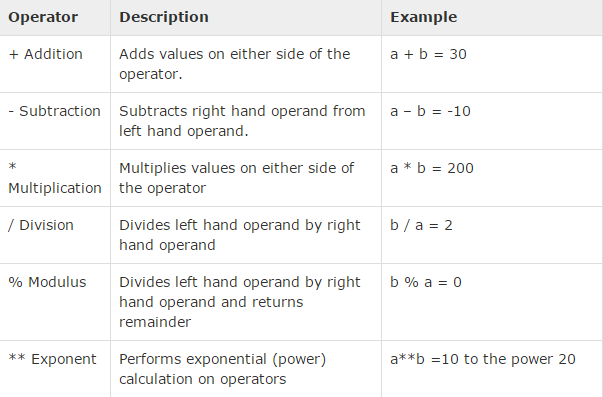
import math  
print(math.pi)  
import sys  
print(sys.path)

**Python Basic Operators**

Types of Operator

Python language supports the following types of operators.

1. **Arithmetic Operators**

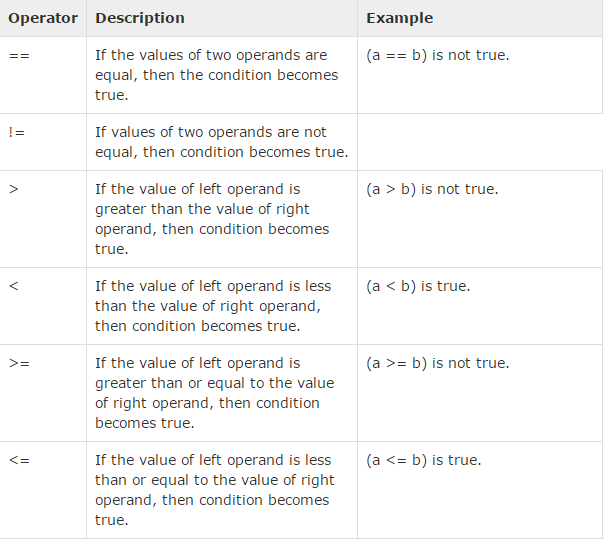
****

**10 / 4 = 2.5 <----normal division**

**10 // 4= 2 <----floor division # it basically cuts of the part after the period**

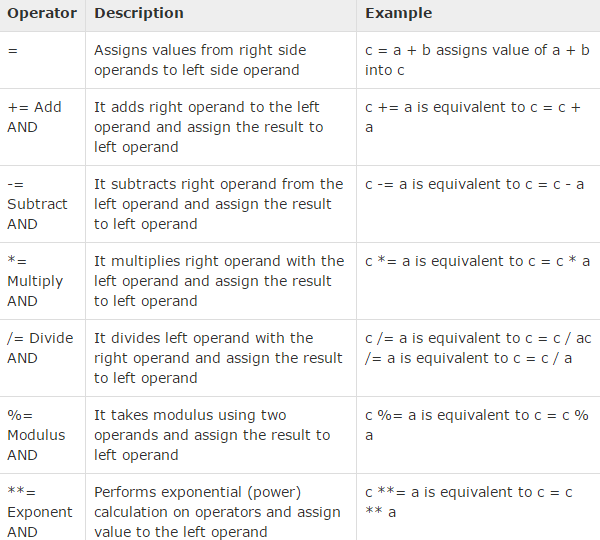
****

1. **Comparison (Relational) Operators**

****

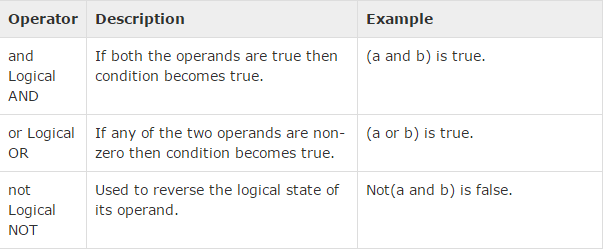
****

1. **Assignment Operators**

****

****

1. **Logical Operators**

****

Note: **Any** and **All** are two built ins provided in python used for successive And/Or.

****

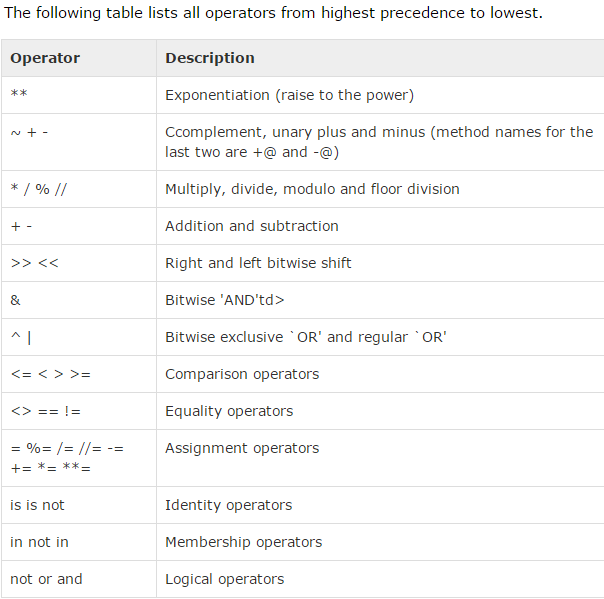
1. **Bitwise Operators**

****

1. **Membership Operators**
2. **Identity Operators**

****

**Operator Precedence**

****