Python Programming

by Narendra Allam

Chapter 1

Introduction

Topics Covering

- Introduction to Python
- Values and variables
- Python data types
- type(), id(), sys.getsizeof()
- Python labeling system
- Object pooling
- Conversion functions
- The language which knew infinity
- Console input, output
- Operators in python
    - Arithmetic operators
    - Relational operators
    - Logical operators
    - Assignment operators
    - Bitwise operators
    - Membership operators
    - Identity operators
    - Ternary Operator
- Built-in functions
- Type conversions
    - int()
    - float()
    - bool()
    - str()
    - complex()
- Interview questions
- Exercise Programs
- Summary

Introduction

There are 5000+ programming langauges in the world and we are still counting... Python is in the top 5 programming langauges as of 2017. Java, C, C++ and C# are other langauges in the top 5.

Python developed by Guido Van Rossum in 1989, and came into the programming world in 1991.

Python came into software industry as a scripting language.. PERL and Bash were other alternatives for scripting by that time.

Scripting languages are mainly used for automation.

Automation:

- *Anything that can speed up the development process and reduce manual effort.*
- *Writing as little code as possible and hiring as little programmers as possible.*

Later python was widely adopted in the research community and became a complete programming language with robust futures like object orientation and functional programming. Having machine learning and data analysis packages, Python has extensive support for research.

> In europe python taught to kids as an introductory language.
> And it is extensively used by scientists in NASA.
> So now we can understand that python has a broader audience.

Why should I learn Python ?

- Python is one of the top 5 languages(among 5000+ programming languages).
- Python is heavily funded by Google..
- There are more career paths after learning Python, one can choose his career path as,
  a web developer
  a data scientist
  a devops engineer
  a server side programmer
  etc..

Python features

1. High-level programming language

High-level programming languages are close to business problems. . Whereas low-level programming languages are close to system's problems..

Example of high-level programming languages: Python, Java, c#, Rube etc..

Low-level Programming languages: C, Assembly programming languages etc..

A high-level programming language makes the process of translating business requirements to computerized solutions faster than low-level langauges. So time taken for software development is very less compared to low-level languages.

Examples of Business problems are, writing banking software, building an ecommerce website etc., Examples of system problems are, writing a device driver, memory manager, anti-virus, etc.

We do not choose high-level programming languages to solve system problems. Even though, development time is more, low-level languages give real-time performance.

So we use low-level languages to solve the system problems.

2. Interpreted

Python is interpreted programming language. Code which is written in english, has to be translated to binary code and understood by a computer. This is the process oftranslation.

There are two types of translations

1. Compilation.
2. Interpretation.

In compilation a compiler translates all the lines of code at the same time then starts the execution. In interpretation an interpreter takes the first line, translates to machine code and executes, then takes the second line and so on.

3. Multi-purpose

Python can be used for developing multiple application types.

- Web Applications using django, flask
- Data analysis using pandas, numpy, bokeh
- Devops Automation using boto
- Embedded apps using raspberry pi API
- Depplearning using google's Tensorflow API

4. Not just a scripting language

Python is a general purpose language. One can use python as a scripting language or as a programming language, the puspose for which we use different. In automation field, python programming is called as scripting. And in application development side, it is called programming. In both cases, a single set of python keywords and constructs are used. As a computer language there is no difference between scripting and Programming in python.

5. Extensible

High-level languages are not performant, because they are more focused to provide developer friendly environment than optimizing for speed.

In real-time and time-critical applications, when performance required, we still have to consider languages with low-level features, like C and C++. Python provides futures required to merge other programming languages with python code. Java code can be used in python using 'jython', C# code can be accessed using 'IronPython'.

6. Multi paradigm

Python welcomes programmers from various backgrounds, as python supports procedural, functional and object oriented programming styles.

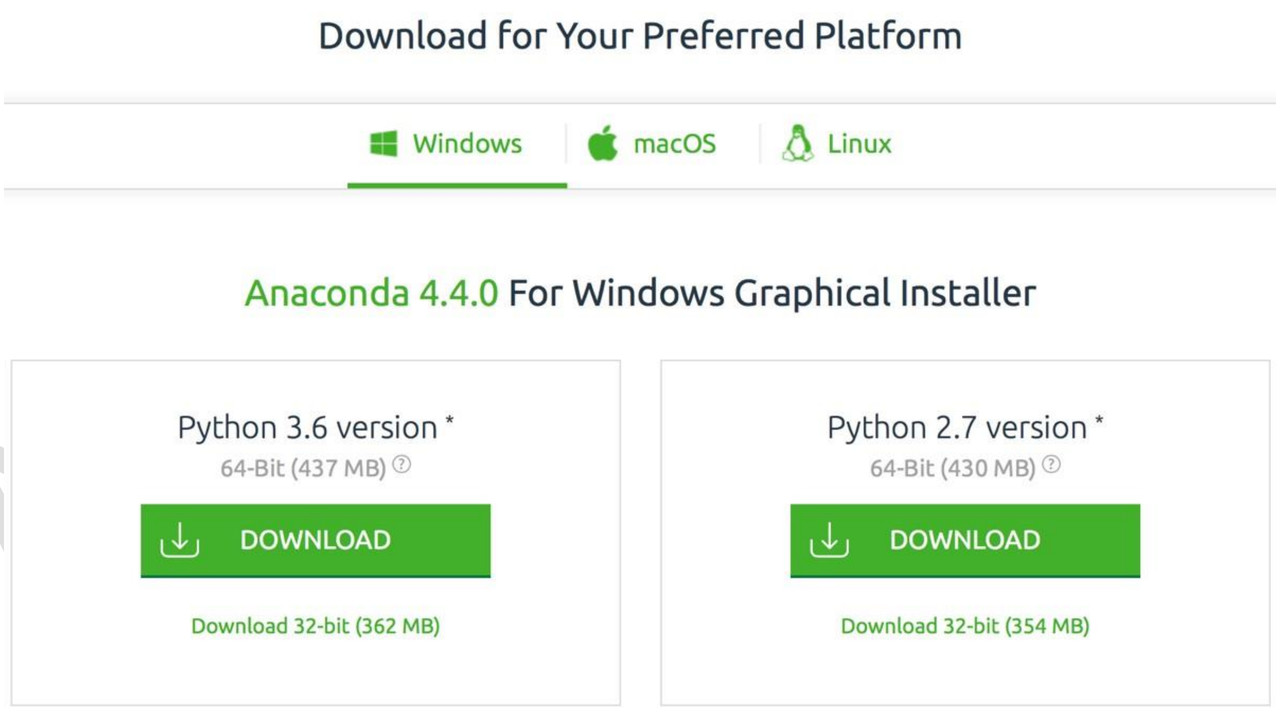
Software Installation

All the code examples in this book, are developed and tested using cutting edge tools, *jupyter notebook* and *PyCharm IDE*.

Anaconda installation:

Download and install python 2.7 www.continuum.io (anaconda python). We are suggesting anaconda python for practice, as it bundles all the packages required for a programmer. If you install python community version from www.python.org, you need to install required packages seperately.

- Download python 2.7 version from https://www.continuum.io/downloads (https://www.continuum.io/downloads)

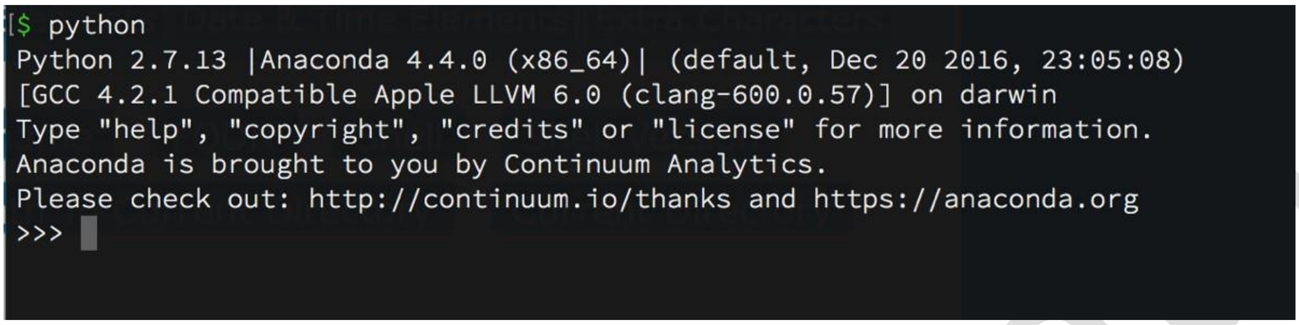


- Just double click on the installer and follow the wizard to complete the installation.

Note: On windows, installer asks to *'add the python path to PATH environment variable'*. Please check the box.

- Open **cmd**(command prompt) on Windows or **terminal**(shell) for Linux and Mac Users.

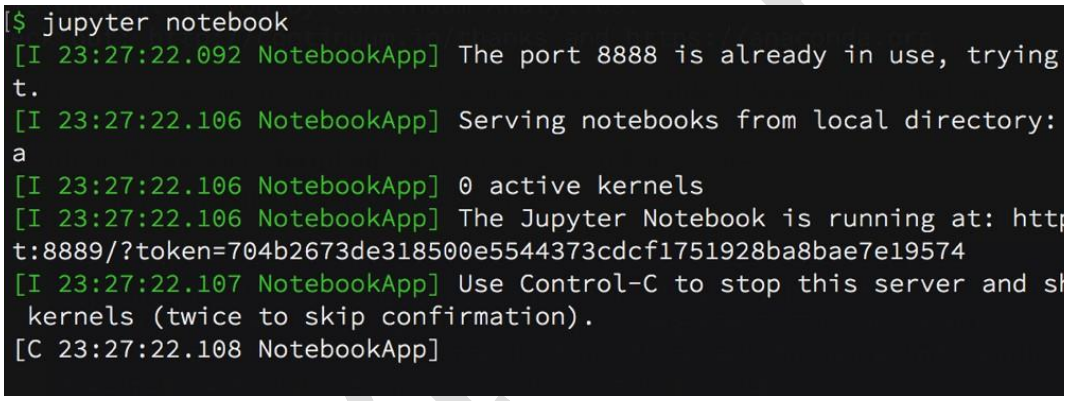
- type 'python' command and hit enter, you should see the below screen.



```
[$ python
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default, Dec 20 2016, 23:05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> 
```

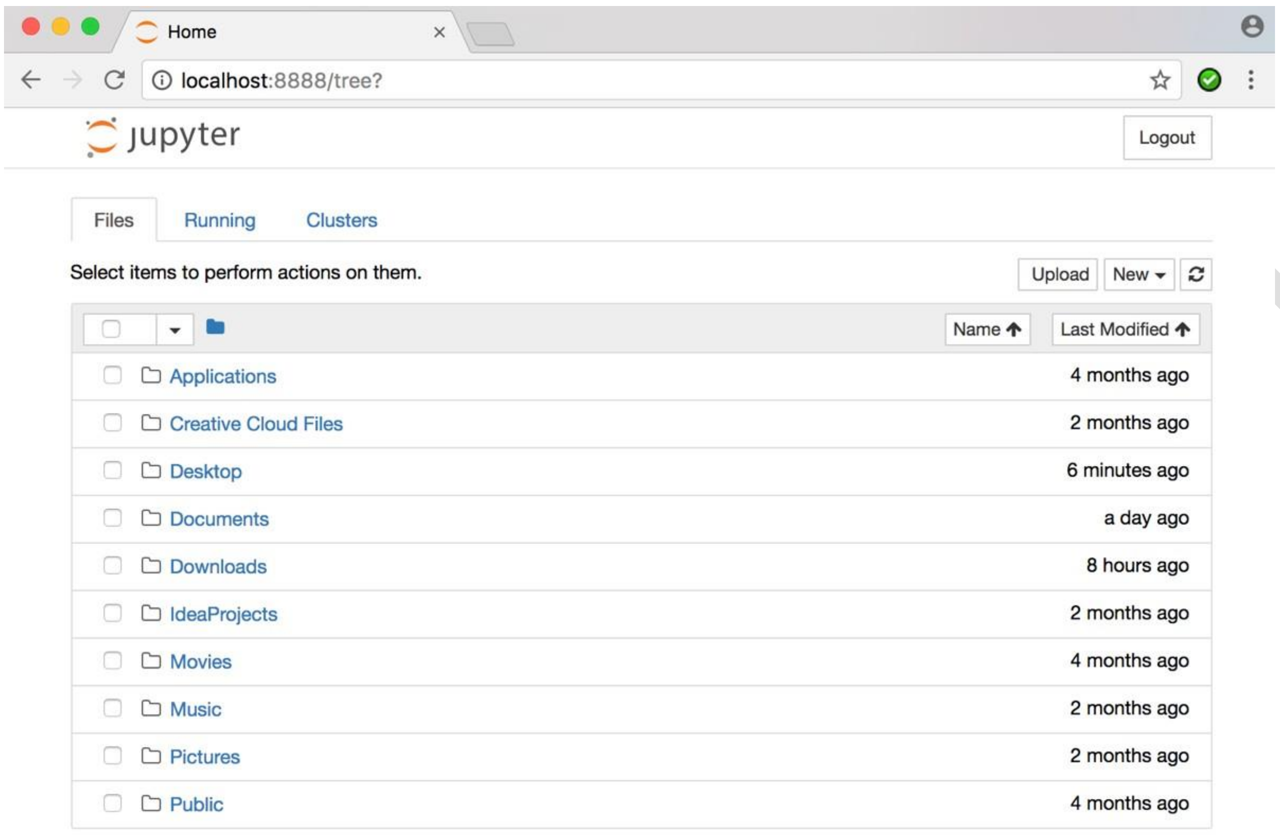
This is python shell, we can use this for python programming, but we have a better option Jupyter notebook. Come out of this shell by typing *'ctrl + D'*.

- Now type command *jupyter notebook*, and hit enter



```
[$ jupyter notebook
 [I 23:27:22.092 NotebookApp] The port 8888 is already in use, trying
 t.
 [I 23:27:22.106 NotebookApp] Serving notebooks from local directory:
 a
 [I 23:27:22.106 NotebookApp] 0 active kernels
 [I 23:27:22.106 NotebookApp] The Jupyter Notebook is running at: htt
 t:8889/?token=704b2673de318500e5544373cdcf1751928ba8bae7e19574
 [I 23:27:22.107 NotebookApp] Use Control-C to stop this server and s
  kernels (twice to skip confirmation).
 [C 23:27:22.108 NotebookApp]
```

- Some text scrolling , but wait until you see your default browser with the following screen.



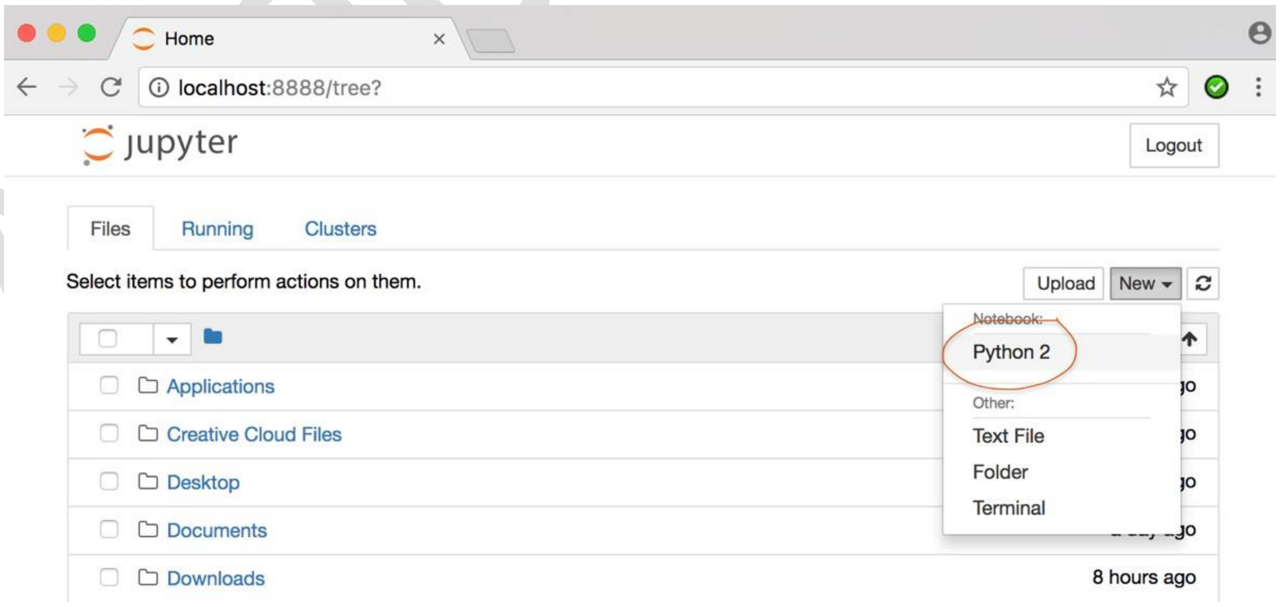
Select items to perform actions on them.

Name	Last Modified
Applications	4 months ago
Creative Cloud Files	2 months ago
Desktop	6 minutes ago
Documents	a day ago
Downloads	8 hours ago
IdeaProjects	2 months ago
Movies	4 months ago
Music	2 months ago
Pictures	2 months ago
Public	4 months ago

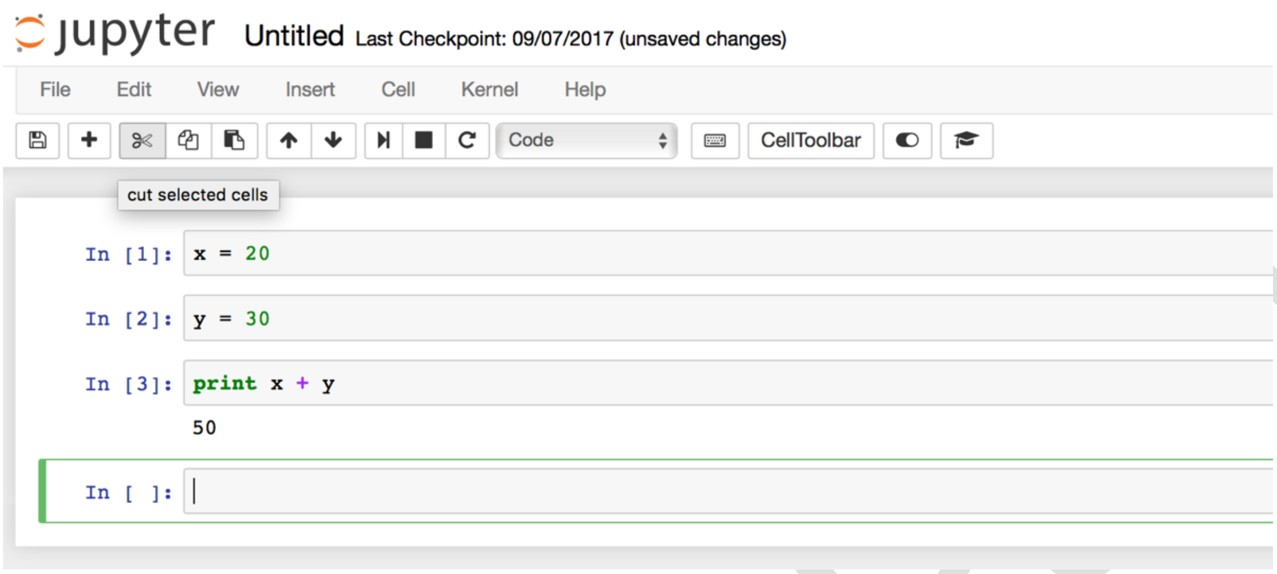
- Do not close cmd/shell, which should be running in the bachground, just minimize it.

Creating a notebook:

- Open New button on the right side and click on python 2, 'Untitled' is created now, we can rename it just clicking on 'Untitled' word on the top.



- We can start programming here, write python code and *'Shift + Enter'* to execute each cell. We can type multiple lines in the same cell.


jupyter Untitled Last Checkpoint: 09/07/2017 (unsaved changes)
File Edit View Insert Cell Kernel Help
Code
CellToolbar
cut selected cells

```
In [1]: x = 20

In [2]: y = 30

In [3]: print x + y
        50

In [ ]:
```

Values and Variables

Python shell can be used as a calculator!

```
   In[]  234 + 123
Output:  357

   In[]  100 * 67.89
Output:  6789.0
```

Program: Birds and Coconut
There is a small bird which can carry one third of its weight. If the bird weight is 60gms and a coconut is 1450gms
how many such birds are required to carry the coconut.

```
   In[]  1450 / 60 / 3
Output:  8
```

Why, result is 8?

Order of evalutation matters? Ofcourse, yes. We are supposed to calculate 60/3 first then the result 20, devides 1450. By default order of evalution is left to right for most of the operators, except assignment operators, it is right to left.

Paranthesis is used to change order of evaluation.

```
In[] 1450 / (60 / 3)
```

```
Output: 72
```

There must be atleast one real number to get accurate results.

```
In[] 1450.0 / (60 / 3)
```

```
Output: 72.5
```

How do we round the numbers to next whole number?

We can use built-in functions. There is a function **ceil()** in module(set of functions) called **'math'** in python. We have to import math module and we can use functions in it.

```
In[] import math
     math.ceil(1450.0 / (60 / 3))
```

```
Output: 73.0
```

There are so many functions in math module, we will discuss in detail in modules topic.

Program: Area of a triangle when sides given

```
In[] (3 + 4 + 5)/ 2.0
```

```
Output:  6.0
```

```
In[] (6*(6-3)*(6-4)*(6-5))**0.5
```

```
Output:  6.0
```

what if we need to calculate are for sides, 7, 8, 9 ???

Using variables

- Varaibles are place holders for values.
- Variables are for identification.
- Variables make things reusable.

If we define variables, we do not need to change entire expression, instead change the values assigned to variables.

```
In[]  a = 3
      b = 4
      c = 5
      s = (a + b + c)/ 2.0

      area = (s * (s - a) * (s - b) * (s - c)) **  0.5
      print 'area: ', area
```

```
area:  6.0
```

Note: To find square roor we can also use math.sqrt() function.

Primitive data Types:

There are 5 primitive data types in python.</br>

- int
- float
- str
- bool
- complex

Dynamically typed
In python variables change their data types dynamically.

x = 20

```
In[]  type(x)
```

```
-------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
<ipython-input-10-c6ec0f91e335> in <module>()
----> 1 type(x)

NameError: name 'x' is not defined
```

type() is a built-in function which tells us the data type of a variable.

```
In[]  x = 3.5
```

```
In[] type(x)
Output: float
```

now 'x' has become a float.

```
In[] x =  'Apple'
```

```
In[] type(x)
Output: str
```

now 'x' has become a str.

```
In[] x = True
```

```
In[] type(x)
Output: bool
```

```
In[] x = 2 + 3j
```

```
In[] type(x)
Output: complex
```

'x' can change its type dynamically, because of python memory model and labeling system.

Program: Average of three numbers

```
In[] a = 2
     b = 4
     c = 5

     s = a + b + c

     avg = s / 3.0

     print 'average =',(avg)

     average = 3.66666666667
```

Reading from console

- raw_input() is the function which is used to read the values from the keyboard. Prompt string is optional

syntax:

```
val = raw_input("Prompt string")
```

```
In[] x = raw_input("Enter value: ")
```

```
Enter value: 20
```

```
In[] y = raw_input("Enter value: ")
```

```
Enter value: 30
```

```
In[] print x
```

```
20
```

Note: *'print'* is the statment used to print values on console but shell is used, *'print'* statment is not required, we can directly get the ouput by typing the variable name. When running as a script *'print'* is mandatory

```
In[] print y
```

```
30
```

Note:
in python 2.x raw_input() - reads input from the console
in python 3.x it is input(), we dont have raw_input() in python 3.x

```
In[] x + y
```

```
Output: '2030'
```

Note: By default 'raw_input()' function reads values as strings.We should explicitely convert them to the target type.

Conversion functions

In python, we can change one type into other if it is legitimate. We have conversion functions for all types in python.

- int()
- float()
- bool()
- str()
- complex()

lets apply int() conversion function on the out put of raw_input()

```
In[] x = int(raw_input("Enter X Value: "))
     y = int(raw_input("Enter y Value: "))
     s = x + y
     print "Sum = ", s
```

```
Enter X Value: 20
Enter y Value: 30
Sum = 50
```

```
In[] x = raw_input()
     type(x)
```

```
Output: str
```

print statement

- print is the statement used to print values on the screen.

syntax:

```
print val1, val2, ...
```

```
In[] print 1234, 'John', True, 456.78
```

```
1234 John True 456.78
```

New style of printing,

```
In[] x = 4
     y = 5
     z = x + y
     print '{} plus {} is {}'.format(x, y, z)

     4 plus 5 is 9
```

'{}' is the place holder for a value we can add any text in between as above

or

```
In[] print '{0} plus {1} is {2}'.format(x, y, z)

     4 plus 5 is 9
```

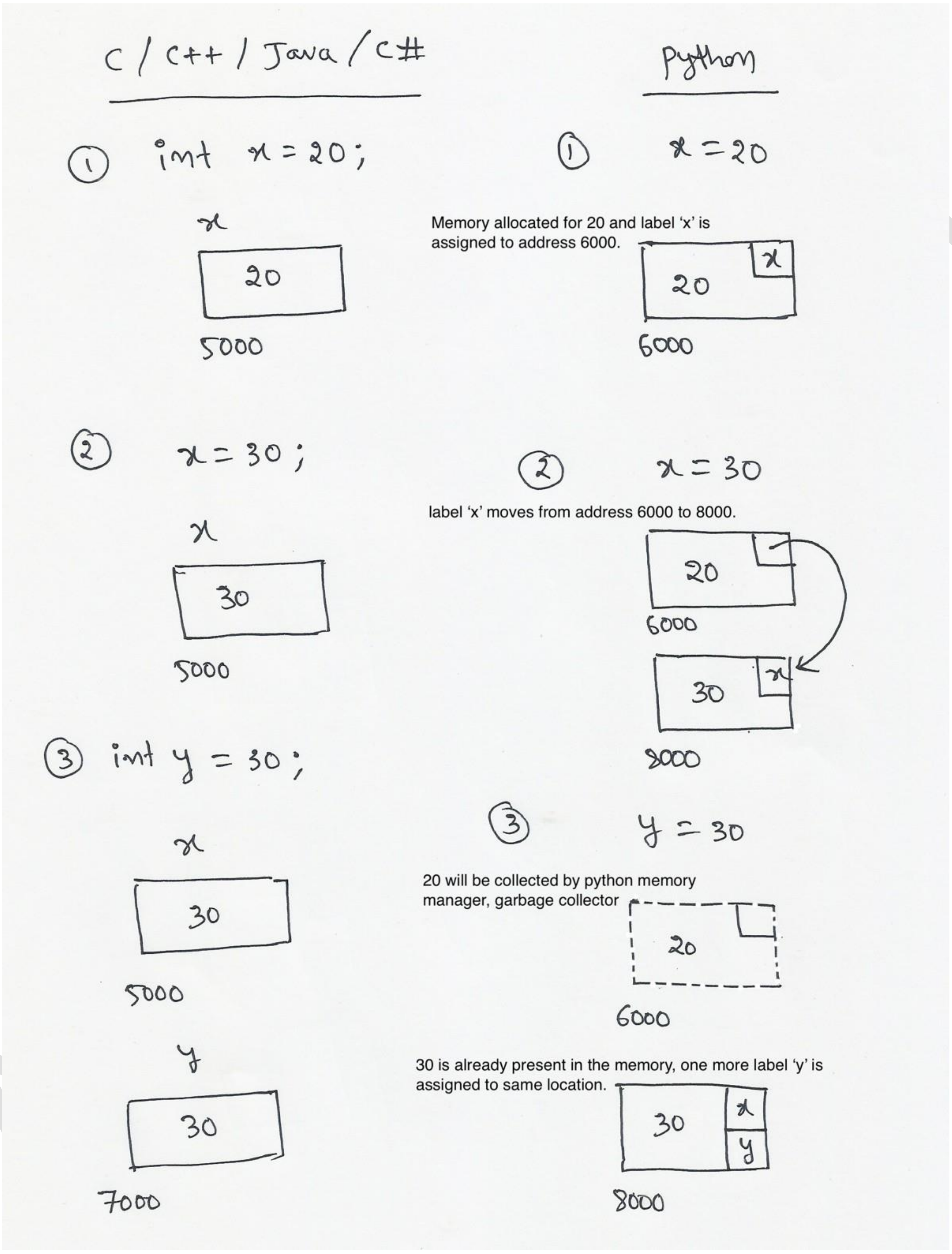
we can reorder the arguments using their positional values(indices) in the format function.
x index is 0,
y index is 1,
z index is 2 and so on...
And we can also use them multiple times, as below

```
In[] print '{1} plus {0} is {2} and {0} is even'.format(x, y,  z)

     5 plus 4 is 9 and 4 is even
```

Python memory model

Everything is an object in python. Comparision with other langauges.


C / C++ / Java / C#
Python
① int x = 20;
x
20
5000
① x = 20
Memory allocated for 20 and label 'x' is assigned to address 6000.
20
x
6000
② x = 30;
x
30
5000
② x = 30
label 'x' moves from address 6000 to 8000.
20
6000
30
x
8000
③ int y = 30;
x
30
5000
y
30
7000
③ y = 30
20 will be collected by python memory manager, garbage collector
20
6000
30 is already present in the memory, one more label 'y' is assigned to same location.
30
x
y
8000

In[]
```
x = 20
```

In[]
```
print id(x)
```

```
140245518555440
```

id() function returns identification, logical address of the object

In[]
```
x = 30
```

If we execute above statement in other programming languages like C/C++, Java and C#, x's old value(20) will be replaced by 30, as 'x' owns a memory location and always can hold one value. In python value '20' owns the location 'x' is just a label to it. When we assign 30, x will be moved to 30's location by leaving 20's location. If there is no lebel assigned to location, python's memory manger 'gc'(garbage collector), collects the memory

Labeling: Python treats object names as lables. When we assign a new value to a label(variable name), it allocates space and constructs an object for new value and adds a label to it, but it do not replace the exisitng value.

In[]
```
print id(x)
```

```
140245518555200
```

we can see, x location changing

In[]
```
y = 30
print id(y)
```

```
140245518555200
```

Surprise! x and y are pointing same object. Python pools frequently used objects.

Integer pooling:

Typical Python program spends much of its time in allocating and deallocating integers, these operations should be very fast. Therefore python uses a dedicated allocation scheme with a much lower overhead (in space and time). Python generally pools integers between -5 to +256, in pre-allocated object list. This is also applicable for characters(**str**) afterall ASCII codes are integers.

Note:

- Pooling is not applicable for **float** values.
- ids are equal for integers ranging from -5 to + 256

```
In[]  x = -5
      y = -5
      print id(x), id(y)

      x = 256
      y = 256
      print id(x), id(y)
```

```
140245518556040 140245518556040
140245518561632 140245518561632
```

ids might not be equal before -5 and afterv + 256

```
In[]  x = -6
      y = -6
      print id(x), id(y)

      x = 257
      y = 257
      print id(x), id(y)
```

```
140245521341632 140245521339256
140245521276440 140245521341144
```

ids are equal before for same strings

```
In[]  x = 'Apple'
      y = 'Apple'
      print id(x), id(y)
```

```
4472141504 4472141504
```

Operators

An operator performs an action on operands.

```
x + y
```

In the below expression + is the operator, x and y are operands. Python supports 7 types of operators,

- Arithmetic
- Relational
- Logical
- Assignment
- Bitwise
- Membership
- Identity

Arithmetic

Operator	Description
+ Addition	Adds values on either side of the operator.
- Subtraction	Subtracts right hand operand from left hand operand.
* Multiplication	Multiplies values on either side of the operator
/ Division	Divides left hand operand by right hand operand
% Modulus	Divides left hand operand by right hand operand and returns remainder
** Exponent	Performs exponential (power) calculation on operators
//	Integer Division or Floor Division - Rounds the quotient towards -ve infinity

Let's start with

In[]
```
a = 7
b = 3
```

In[]
```
a + b
```

```
Output: 10
```

In[]
```
a - b
```

```
Output: 4
```

```
In[]  a * b
Output: 21

In[]  a / b
Output: 2

In[]  7 / 2
Output: 3

In[]  7 / 2.0
Output: 3.5
```

Note: There must be atleast one real number to get real number in any python expression.

```
In[]  7 // 2
Output:  3

In[]  7 // 2.0
Output:  3.0
```

// - (integer division) always gives integer part of the result

```
In[]  7 // 2
Output: 3

In[]  7 // 2.0
Output: 3.0
```

% - Modules or remainder operator

```
In[]  7 % 4
Output:  3

In[]  4 % 7
Output:  4
```

** - Exponential Operator

```
In[]  9 ** 2
Output: 81
```

```
In[]  9 ** 0.5
Output: 3.0
```

Relational operators

Relational operators produce boolean values(True or False)

Operator	**Description**
==	If the values of two operands are equal, then the condition becomes True.
!=	If values of two operands are not equal, then condition becomes True.
>	If the value of left operand is greater than the value of right operand, then condition becomes True.
<	If the value of left operand is less than the value of right operand, then condition becomes True.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes True.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes True.

```
In[]  x = 20
      y = 30

      print x == y

      False
```

```
In[]  x < y
Output: True
```

```
In[]  x <= y
```

```
Output: True
```

```
In[]  x > y
```

```
Output: False
```

```
In[]  x >= y
```

```
Output: False
```

```
In[]  x != y
```

```
Output: True
```

```
In[]  x <> y
```

```
Output:  True
```

Logical Operators

Logical operators are used to combine multiple relational expressions into one.

Operator	**Description**
and Logical AND	If both the operands are true then condition becomes true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.
not Logical NOT	Used to reverse the logical state of its operand.

Lets take

```
x = 20
y = 30
```

```
In[]  x = 20
      y = 30
```

and: gives **True** when all relational expressions are **True** remaining cases it gives **False**

```
In[]  (x > 20) and (y%6 == 0)
```

```
Output: False
```

or: gives **False** when all relational expressions are **False** remaining cases it gives **True**

```
In[]  (x > 30) or (y%7 == 0)
```

```
Output: False
```

```
In[]  (x > 30) or (y%6 == 0)
```

```
Output: True
```

not: can be applied on any boolean expression, which converts a **True** to **False** and **False** to **True**

```
In[]  not x > 30
```

```
Output: True
```

```
In[]  (not x > 30) or (y%7 == 0)
```

```
Output: True
```

Short-hand assignment operators

Operator	**Description**
=	Assigns values from right side operands to left side operand
+= Add AND	It adds right operand to the left operand and assign the result to left operand
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand
/= Divide AND	It divides left operand with the right operand and assign the result to left operand
%= Modulus AND	It takes modulus using two operands and assign the result to left operand
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand
//= Floor Division	It performs floor division on operators and assign value to the left operand

'=' is assignment operator.

```
x = 20
```

in the bove statment, 20 is the value assigned to variable 'x'. 'x' refers to '20', untill we assign a new value.

```
x = 20
x += 10
```

After the execution of above two statments, x value becomes '30'. '+=' is short hand assignment operator, which adds right side value '10' to 'x', and stores the result back in 'x'. That means, "x += 10' is equivalent to "x = x + 10".

let's see another example

```
In[]  x = 7
      y = 4
      x %= y
      print x
```

```
3
```

another one,

```
In[]  x = 7
      y = 4
      x //= y
      print x
```

```
1
```

Bitwise Operators

Bitwise operators are used to manipulate values at bit and byte level.

Operator	**Description**
& Binary AND	Operator copies a bit to the result if it exists in both operands
| Binary OR	It copies a bit if it exists in either operand.
^ Binary XOR	It copies the bit if it is set in one operand but not both.
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.

Before starting bitwise operators, we need to undertsand how numbers are represented in various number systems in python.

Number Prefix

The dfault number system everybody uses is decimal number system. We can also represent numbers directly in other number systems in python code.

```
x = 0b10101
x = 0o4526
x = 0xAB2F
```

```
In[]  x = 0b10100
      print x

      20

In[]  x = 0o4526
      print x

      2390

In[]  x = 0xAB2F
      print x

      43823
```

Left shifting

```
In[]  x = 20
      x << 3

Output: 160

In[]  x

Output: 20
```

In the above example we are shifting bits of 20 (i.e 0b10100) 3 times to the left side, which becomes 0b10100000.
Three 0s are added to the right side.

Note: When x is shifted n times to the left side, its value becoms, x * 2^n

Right Shifting

```
In[]  x = 160
      x >> 3
```

```
Output: 20
```

```
In[]  print x
```

```
160
```

In the above example we are shifting bits of 160 (i.e 0b10100000) 3 times to the right side, which becomes 0b10100.
Three bits are discarded from right side.

Note: When x is shifted n times to the right side, its value becoms, x / 2^n

When printing on console or shell numbers are displayed in decimal.

number system conversion functions

We can get string reprenation of a number in any hexadecimal, octal and binary number systems using the fllowing functions,

- bin()
- hex()
- oct()

Note: All the above functions returns strings not numbers.

```
In[]  bin(2345)
```

```
Output: '0b100100101001'
```

```
In[]  hex(0o234)
```

```
Output: '0x9c'
```

```
In[]  oct(0xea)
```

```
Output: '0352'
```

Side Track: No limit for number size.

Python can handle huge numbers, as, it is not having any limitation on the number size.

Note: *sys.getsizeof()* function gives the number of bytes allocated to a value. We have to import sys module to use this function.

```
In[]  import sys

      x = 2**1234567
      y = 20

      print sys.getsizeof(x), sys.getsizeof(y)
```

```
164636 24
```

"The language which knew Infinity"

We can represent infinity in python

```
In[]  x = float("inf")
```

'x' is holding infinity here

```
In[]  y = float("-inf")
```

'y' is -ve infinity

```
In[]  2 ** 1234567 < x
```

```
Output: True
```

anything is lessthan +infinity

```
In[]  y < 2 ** 1234567
```

```
Output: True
```

-ve Infinity is lessthan everything,

Infinity equals to infinity

```
In[]  x == x
```

```
Output: True
```

Do you really thing python stores infinity in memory????? Nohh, actually, it is playing with your mind. It simply gives you **True**, when infinity is on the right side of less than symbol!!! same for -ve infinity.

Let's comback to our bitwise operators...

Bitwise AND : &

AND (&) results 1 if both the bits are One else zero.

```
In[]  x = 0b1010
      y = 0b1100
      bin(x & y)
```

```
Output: '0b1000'
```

Bitwise OR : |

OR(|) results 0 if both the bits are Zero else One.

```
In[]  x = 0b1010
      y = 0b1100
      bin(x | y)
```

```
Output: '0b1110'
```

Complement : ~

~ toggles ones to zeros, zeros to ones

```
In[]  x = 1
      print ~x
```

```
-2
```

```
In[]  x = 15
      print ~x
```

```
-16
```

In python, signed integers are represented in 2's complement integer representation.

In 16 bits, 1 is represented as 0000 0000 0000 0001. Inverted, we get 1111 1111 1111 1110, which is -2.

Similarly, 15 is 0000 0000 0000 1111. Inverted, you get 1111 1111 1111 0000, which is -16.

Bitwise EX-OR, Exclusive OR: ^

EX-OR(^) - results 0, if both are either zeros or ones, results one, if one is zero and other is one.

```
In[]  x = 0b1010
      y = 0b1100
      bin(x ^ y)
```

```
Output: '0b110'
```

Bitwise short-hand assignment operators

```
<<=, >>=, &=, |=, ^=
```

, x <<= n is equivalent of x = x << n

Membership operators

in , not in

Checks the membership of the item/sub string in the container/string

```
In[]  s1 = 'Hello world!'
      s2 = 'Hell'

      s2 in s1
```

```
Output: True
```

```
In[]  ' ' in '+*$%^ @'
```

```
Output: True
```

```
In[]  "World" in s1
```

```
Output: False
```

```
In[] "xyz" in s1
```

```
Output: False
```

```
In[] "xyz" not in s1
```

```
Output: True
```

Identity operators

Identity operators checks the id() equality. Results to True, if both varibles are having reference of same object else False.

is, is not

```
In[] s1 = "Hello"
     s2 = "Polo"
     s3 = "Hello"
```

```
In[] print id(s1), id(s2), id(s3)

     4472242016 4472239520 4472242016
```

```
In[] print s1 is s3

     True
```

```
In[] print s1 is s2

     False
```

```
In[] print s1 is not s2

     True
```

```
In[] x = 25
     y = 25
     print id(x), id(y)

     140245518555320 140245518555320
```

```
In[] x is y
```

```
Output: True
```

```
In[]  x = 2.3
      y = 2.3
      print id(x), id(y)

      140245518173280 140245518173184

In[]  x is y

Output: False

In[]  c1 = 2 + 3j

In[]  c2 = 2 + 3j

In[]  print id(c1), id(c2)

      4471943600 4471943856

In[]  c1 is c2

Output: False

In[]  c1 == c2

Output: True
```

Keywords in Python

Every language has a set of keywords. In python, to know the list of all keywords, just import module 'keyword' and use the kwlist variable to see all the keywords.

```
In[]  import keyword
      print keyword.kwlist

      ['and', 'as', 'assert', 'break', 'class', 'continue', 'def',  'del'
      , 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from',  'glo
      bal', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass',  '
      print', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Interview Questions

Output ?

```
In[]  print 1450 / 60 / 3
```

```
8
```

```
In[]  x = 25
      y = 25
      print id(x) == id(y)
```

```
True
```

```
In[]  x = raw_input("Enter value: ")
      y = raw_input("Enter value: ")
      print x + y
```

```
Enter value: 20
Enter value: 30
2030
```

Exrecise Programs

1. Write a program to calcualte compound interest when principle, rate and number of periods are given.
2. Write a program to convert gven seconds to hours and minutes
3. Given coordinates (x1, y1), (x2, y2) find the distance between two points
4. Read name, address, email and phone number of a person through keyboard and print the details.

Notes

- print is a keyword in python 2 and a function in python 3
- raw_input() function replaced with input() function in python 3
- In python 2, / operator gives 'int' value, if both the operands are 'int'. In python 3 it gives real value.
- Python pools integer values, 'int' and 'str' types, except float values.
- float('inf') is + infinity, float('-inf') is - infinity.