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
- Built-in functions
- Type conversions
- Exercise Programs

Introduction

There are 5000+ programming langauges in the world and we are still counting... Python is in the top 5 programming langauges as of 2019. 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.

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.

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..

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.

```
Example:-     High-level programming languages: Python, Java, c#, Rube etc..
              Low-level Programming languages: C, Assembly programming langu
ages etc..
```

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 of translation.

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 first line of code.

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, matplotlib
- Devops Automation using boto
- IoT apps using raspberry pi API
- Deeplearning using google's Tensorflow API, PyTorch, thaenos etc

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 is 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.

Note:

The biggest difference is, Programming requires designing whihc is not necessary for scripting As we design, programs are maintained for longer time, where as scripts are short-lived and use and throw code files.

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.

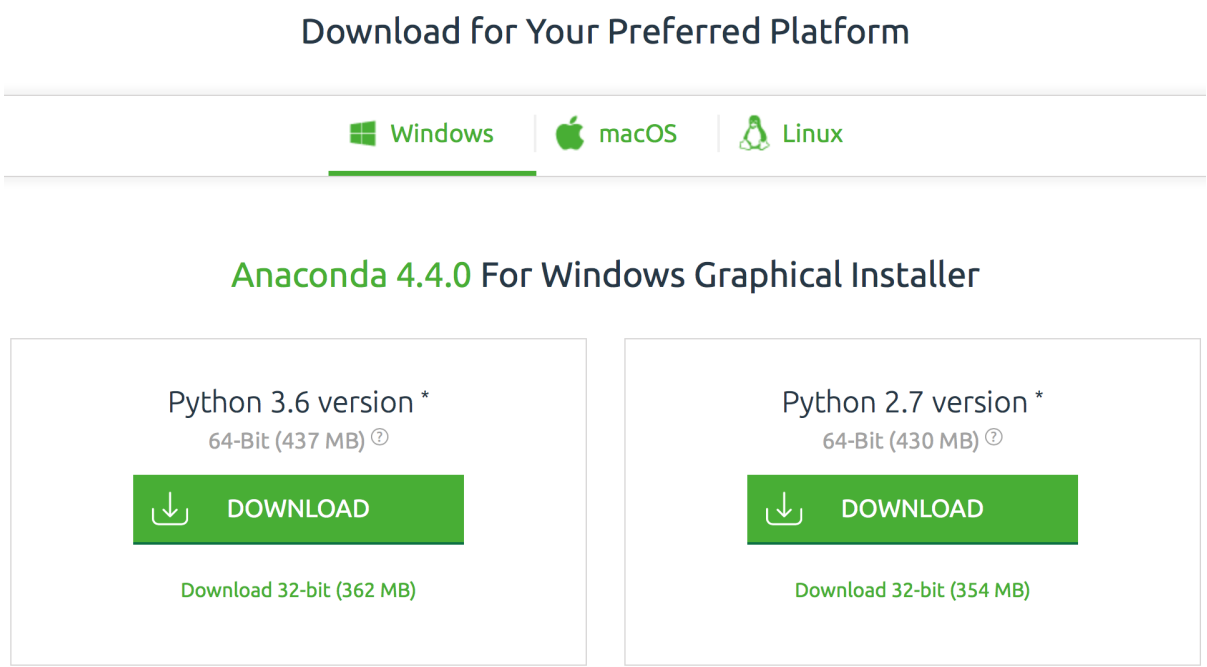
Software Installation

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

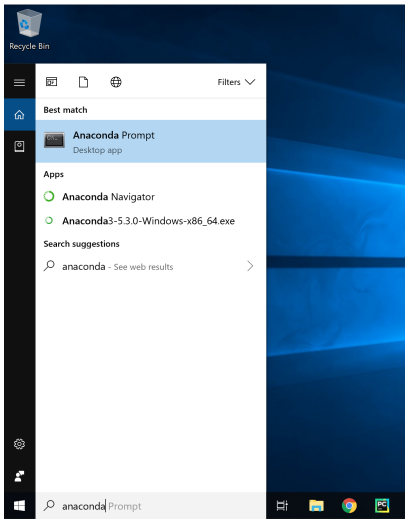
Anaconda installation:

Download and install python 3.7 www.anaconda.com (http://www.anaconda.com) (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 (http://www.python.org), you need to install required packages seperately.

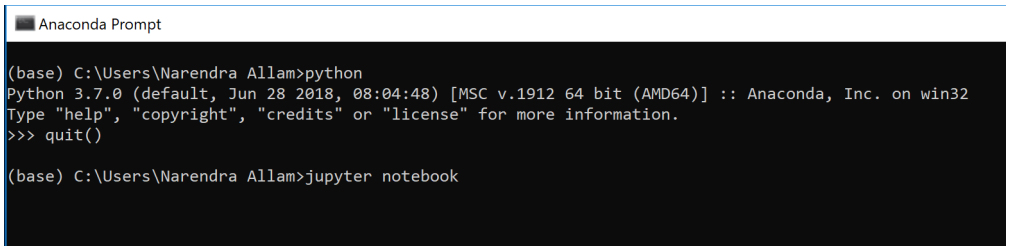
- Download python 3.7 or later version, from https://www.anaconda.com/download/ (https://www.anaconda.com/download/)



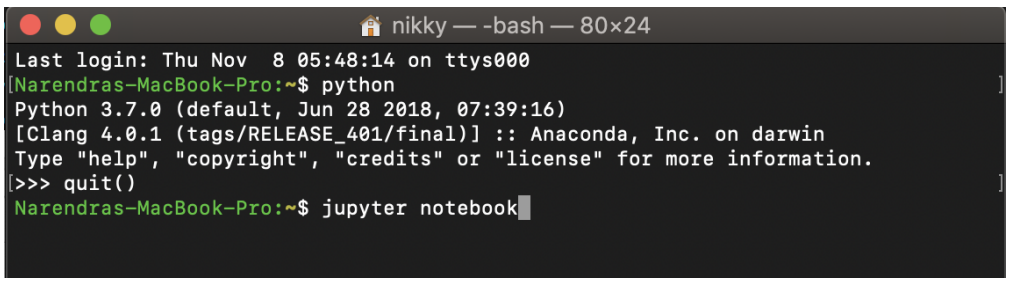
- Just double click on the installer and follow the wizard to complete the installation.
- Open **Anaconda Prompt**(command prompt) on Windows or **terminal**(shell) for Linux and Mac Users.
- type 'python' command and hit enter, you should see the below screen.



WINDOWS:

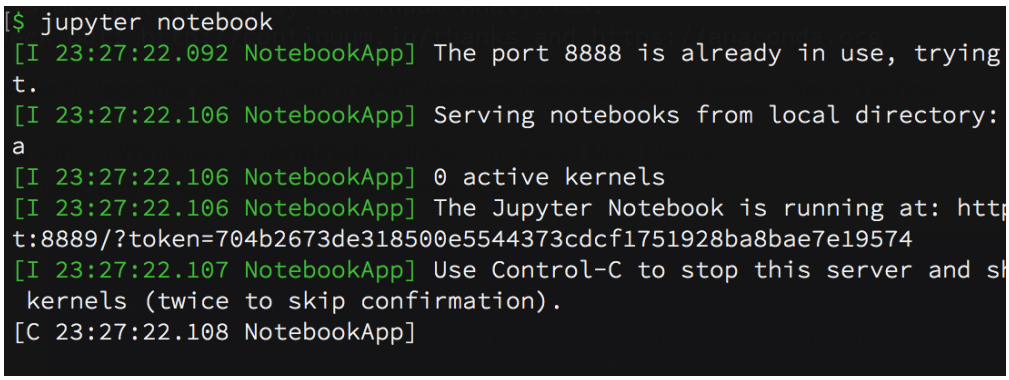


MAC:

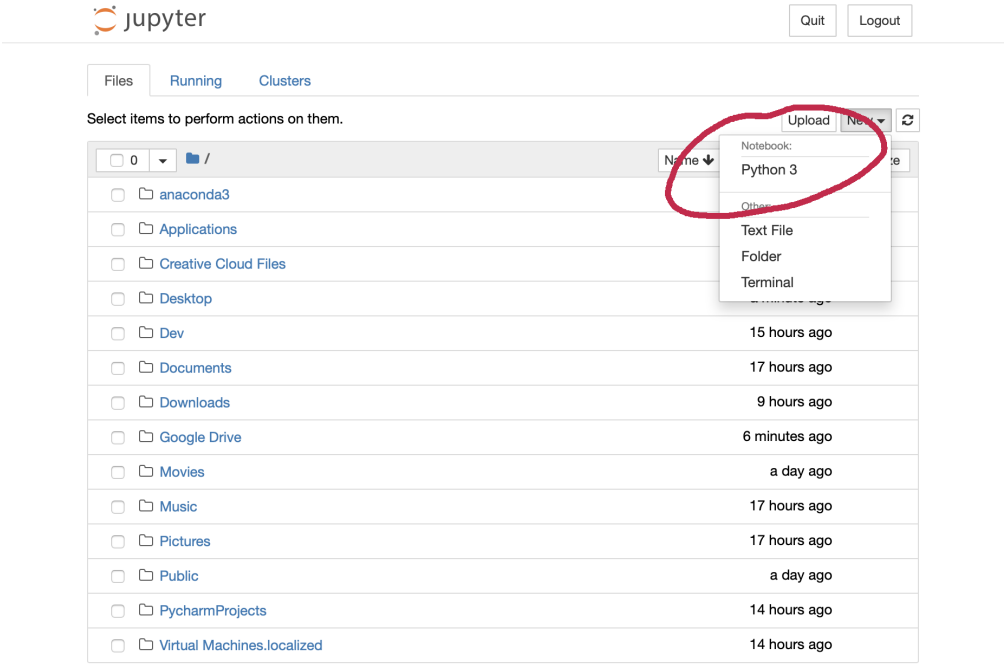


This is python shell, we can use this for python programming, but we have a better option Jupyter notebook. Come out of this shell by typing _'ctrl + D'_.

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



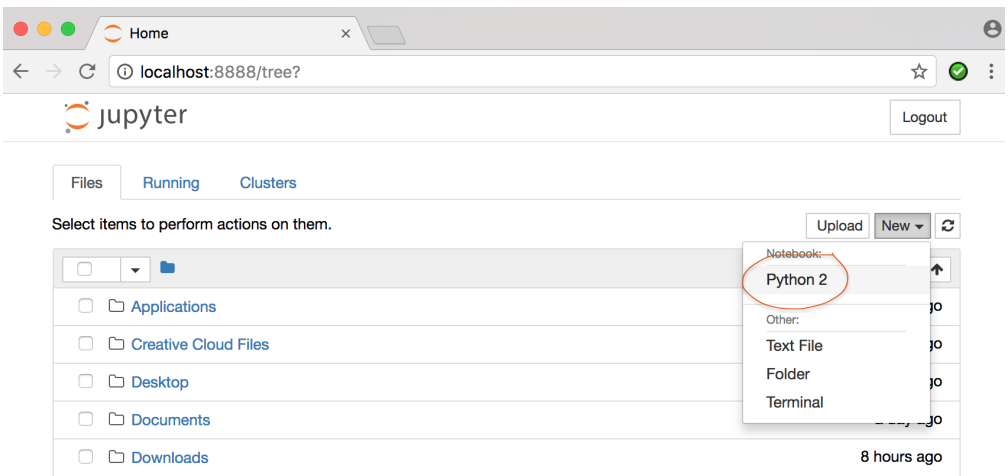
- Some text scrolling , but wait until you see your default browser with the following screen.



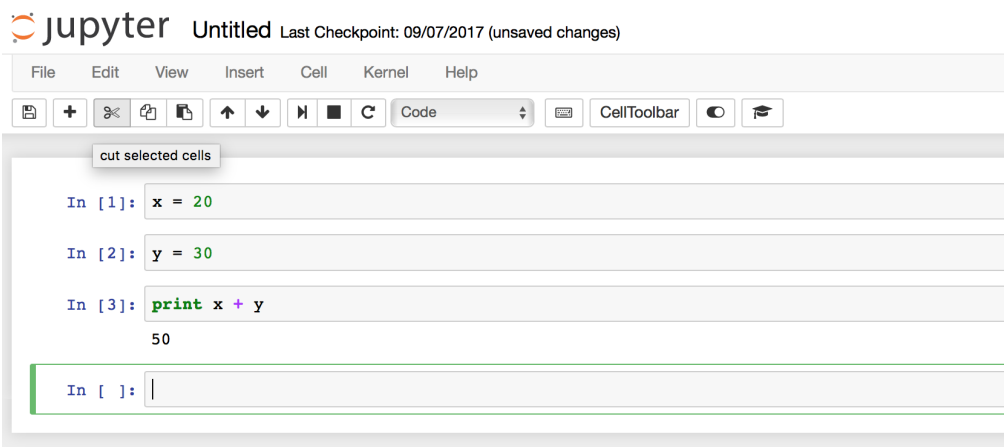
- Do not close cmd/shell, which should be running in the bachground, just minimize it.

Creating a notebook:

- Open New button on the right side and click on python 2 or python 3, then '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.



Values and Variables

Python shell can be used as a calculator!

In [1]:

```
100 * 67.89
```

Out[1]:

```
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 [2]:

```
1450 / 60 / 3
```

Out[2]:

```
8.055555555555555
```

Why, result is 8.05555555555?

Order of evaluation matters? Of course, yes. We are supposed to calculate 60/3 first then the result 20, divides 1450. By default order of evaluation 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 [3]:

```
1450 / (60 / 3)
```

Out[3]:

```
72.5
```

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

In [4]:

```
1450.0 / (60 / 3)
```

Out[4]:

```
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 [5]:

```
import math
math.ceil(1450.0 / (60 / 3))
```

Out[5]:

```
73
```

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 [6]:

```
(3 + 4 + 5)/ 2.0
```

Out[6]:

```
6.0
```

In [7]:

```
(6*(6-3)*(6-4)*(6-5))**0.5
```

Out[7]:

```
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 [8]:

```
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: print() is an output function in python3, and is used to display output on the conaolew or shell.

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

Primitive data Types:

There are 5 primitive data types in python.

- int
- float
- str
- bool
- complex

Dynamically typed
In python variables change their data types dynamically.

In [9]:

```
1 x = 20
2 print(type(x))
```

```
<class 'int'>
```

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

In [10]:

```
1 x = 3.5
```

In [11]:

```
1 type(x)
```

Out[11]:

```
float
```

now 'x' has become a float.

In [12]:

```
1 x =  'Apple'
```

In [13]:

```
1 type(x)
```

Out[13]:

```
str
```

now 'x' has become a str, as python is **'Dynamically typed'** language. Variables can change their data type when a different value is assigned.

In [14]:

```
1 x = True
```

In [15]:

```
1 type(x)
```

Out[15]:

```
bool
```

In [16]:

```
x = 2 + 3j
```

In [17]:

```
type(x)
```

Out[17]:

```
complex
```

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

Program: Average of three numbers

In [18]:

```
a = 2
b = 4
c = 5

s = a + b + c
avg = s / 3
print('average =', avg)
```

```
average = 3.6666666666666665
```

Reading from console

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

syntax:

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

In [19]:

```
x = input("Enter x value: ")
```

```
Enter x value: 12
```

In [20]:

```
y = input("Enter y value: ")
```

```
Enter y value: 25
```

In [21]:

```
print(x)
```

```
12
```

In [22]:

```
print(y)
```

```
25
```

Note:

- print is the statement/function used to print values on the screen.
- in python 2.x, raw_input() - reads input from the console and 'print' is a statement used to print values on console
- in python 3.x, it is input(), we don't have raw_input() in python 3.x and 'print()' is a function used to print values on console

As explained above, '*print*' is the statement/function used to print values on console but when shell is used, '*print*' statement is not required, we can directly get the output by typing the variable name. When running as a script '*print*' is mandatory

In [23]:

```
print(1234, 'John', True, 456.78, 2 + 3j)
```

```
1234 John True 456.78 (2+3j)
```

In [24]:

```
x + y
```

Out[24]:

```
'1225'
```

Note: By default 'raw_input()' function reads values as strings.We should explicitly 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 [25]:

```
x = int(input("Enter x Value: "))
y = int(input("Enter y Value: "))
s = x + y
print("Sum = ", s)
```

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

Type of the data we are reading from keyboard is 'str'

In [26]:

```
x = input()
type(x)
```

```
56
```

Out[26]:

```
str
```

In [27]:

```
x = -100
y = 300
z = x + y
print(x, 'plus', y, 'is', z, 'and', x, 'is -ve')
```

```
-100 plus 300 is 200 and -100 is -ve
```

This is old style of printing.

New style of printing is,

In [28]:

```
x = -100
y = 300
z = x + y
print('{} plus {} is {} and {} is -ve'.format(x, y, z, x))
```

```
-100 plus 300 is 200 and -100 is -ve
```

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

or

In [29]:

```
x = -100
y = 300
z = x + y
print('{0} plus {1} is {2} and {0} is -ve'.format(x, y, z))
```

```
-100 plus 300 is 200 and -100 is -ve
```

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

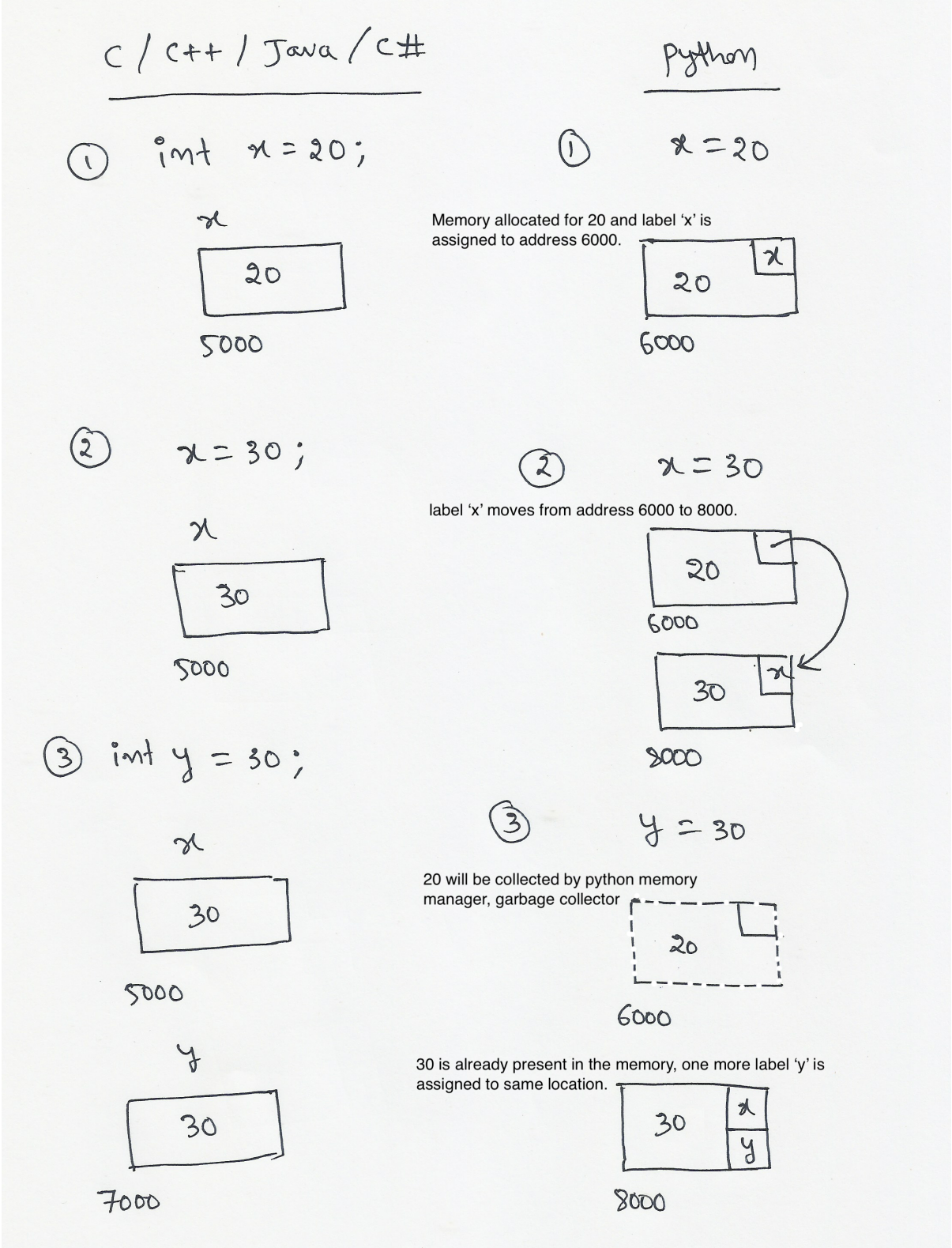
In [30]:

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

```
300 plus -100 is 200 and -100 is even
```

Python memory model

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



In [31]:

```
1 x = 20
```

In [32]:

```
1 print (id(x))
```

```
10965504
```

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

In [33]:

```
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 label assigned to location, python's memory manager '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 [34]:

```
print(id(x))
```

```
10965824
```

we can see, x location changing

In [35]:

```
y = 30
print(id(y))
```

```
10965824
```

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 [36]:

```
x = -5
y = -5
print (id(x), id(y))

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

```
10964704 10964704
10973056 10973056
```

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

In [37]:

```
x = -6
y = -6
print (id(x), id(y))

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

```
140183578270576 140183578270640
140183578271024 140183578271120
```

ids are equal before for same strings

In [38]:

```
x = 'Apple'
y = 'Apple'
print (id(x), id(y))
```

```
140183577803496 140183577803496
```

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 Operators

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 [39]:

```
a, b = 7, 3
```

In [40]:

```
a + b
```

Out[40]:

```
10
```

In [41]:

```
a / b
```

Out[41]:

```
2.3333333333333335
```

In [42]:

```
7 / 2 # Real number division
```

Out[42]:

```
3.5
```

In [43]:

```
7 // 2  # Integer or Floor division
```

Out[43]:

```
3
```

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

In [44]:

```
7 // 2.0
```

Out[44]:

```
3.0
```

% - Modules or remainder operator

In [45]:

```
7 % 4
```

Out[45]:

```
3
```

In [46]:

```
4 % 7
```

Out[46]:

```
4
```

** - Exponential Operator

In [47]:

```
9 ** 2   # square of nine
```

Out[47]:

```
81
```

In [48]:

```
9 ** 0.5  # square root of nine
```

Out[48]:

```
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 [49]:

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

Out[49]:

```
False
```

In [50]:

```
x < y
```

Out[50]:

```
True
```

In [51]:

```
x <= y
```

Out[51]:

```
True
```

In [52]:

```
x != y
```

Out[52]:

```
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.

In [53]:

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

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

In [54]:

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

Out[54]:

```
False
```

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

In [55]:

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

Out[55]:

```
False
```

In [56]:

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

Out[56]:

```
True
```

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

In [57]:

```
not x > 30
```

Out[57]:

```
True
```

In [58]:

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

Out[58]:

```
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

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

'=' is assignment operator. After the execution of above statements, 20 is the value assigned to variable 'x'. 'x' refers to '20', until we assign a new value to x.

'+=' 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". So x value becomes '30'.

In [59]:

```
x = 7
y = 4
x %= y # equivalent to x = x%y
print (x)
```

3

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

```
x = 0b10101 # 0b prefix is used to represent binary numbers in python
x = 0o4526  # 0o prefix is used to represent octal numbers
x = 0xAB2F  # 0x prefix is used to represent hexadecimal numbers
```

Left shifting

In [60]:

```
x = 20
print (x << 3)
```

```
160
```

In [61]:

```
x
```

Out[61]:

```
20
```

In this 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 [62]:

```
x = 160
x >> 3
```

Out[62]:

```
20
```

In [63]:

```
print (x)
```

```
160
```

In this 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

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

Number System conversion functions

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

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

Note: All the above functions returns strings not numbers.

In [64]:

```
bin(2345) # converting decimal to binary
```

Out[64]:

```
'0b100100101001'
```

In [65]:

```
hex(0o234) # converting octal to hexadecimal
```

Out[65]:

```
'0x9c'
```

In [66]:

```
oct(0xea) # converting hexadecimal to octal
```

Out[66]:

```
'0o352'
```

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 [67]:

```
import sys

x = 2**1234567
y = 20

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

```
164636 28
```

"The language which knew Infinity"

We can represent infinity in python

In [68]:

```
x = float("inf")
```

'x' is holding infinity here

In [69]:

```
y = float("-inf")
```

'y' is -ve infinity

In [70]:

```
2 ** 1234567 < x
```

Out[70]:

```
True
```

anything is less than +infinity

In [71]:

```
y < 2 ** 1234567
```

Out[71]:

```
True
```

-ve Infinity is less than everything,

Infinity equals to infinity

In [72]:

```
x == x
```

Out[72]:

```
True
```

Do you really thing python stores infinity in memory????? No, 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 come back to our bitwise operators...

Bitwise AND : &

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

In [73]:

```
x = 0b1010
y = 0b1100
bin(x & y)
```

Out[73]:

```
'0b1000'
```

Bitwise OR : |

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

In [74]:

```
x = 0b1010
y = 0b1100
bin(x | y)
```

Out[74]:

```
'0b1110'
```

Complement : ~

~ toggles ones to zeros, zeros to ones

In [75]:

```
x = 1
print (~x)
```

```
-2
```

In [76]:

```
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 [77]:

```
x = 0b1010
y = 0b1100
bin(x ^ y)
```

Out[77]:

```
'0b110'
```

Bitwise short-hand assignment operators

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

e.g, 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 [78]:

```
s1 = 'Hello world!'
s2 = 'Hell'
print (s2 in s1)
print ("World" in s1)
print ("xyz" not in s1)
```

```
True
False
True
```

In [79]:

```
print (' ' in '+*$%^ @')
```

```
True
```

Identity operators

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

is, is not

In [80]:

```
s1 = "Hello"
s2 = "Polo"
s3 = "Hello"
```

In [81]:

```
print (id(s1), id(s2), id(s3))
```

```
140183577480920 140183577480528 140183577480920
```

In [82]:

```
print (s1 is s3)
```

```
True
```

In [83]:

```
print (s1 is s2)
```

```
False
```

In [84]:

```
x = 2.3
y = 2.3
print (id(x), id(y))
```

```
140183578663624 140183578663480
```

In [85]:

```
x = 2.3
y = x
print (id(x), id(y))
print(x is y)
```

```
140183578663672 140183578663672
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 [86]:

```
import keyword
print (keyword.kwlist)
# help("keywords")
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'co
ntinue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'fr
om', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'no
t', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Knowing python version

In [87]:

```
import sys
print(sys.version)
```

```
3.6.6 (default, Sep 12 2018, 18:26:19)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]]
```

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 given 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.