CHAPTER 1: Introduction

Python Name - Famous Tv Show Monty python's flying circus.

Python is

- General-purpose
- interpreted
- interactive
- high-level programming language

The main advantage of python is open source under GNU General Public License. It is an interpreted language means the code is processed at runtime, similar to PERL and PHP. Python also supports object-oriented similar to other high level programming languages like C++, java that encapsulate code within the object. For beginners to learn programming, python is correct choice.

History:

Python is introduced by Guido Van Rossum during 1985 to 1990 at National Research Institute for Mathematics and Computer Science in the Netherlands. It is derived from many other languages including ABC, modula-3, C, C++, Unix etc.

Features:

- Easy to learn
- Easy to read
- Easy to maintain
- Broad standard library
- Interactive mode

- Portable
- Extendable
- Databases
- GUI Programming
- Scalable

Python supports functional and Structured programming methods as well as OOP. It is also Used as a Scripting Language or can be compiled to byte code for building large applications. It provides high level dynamic datatype and supports dynamic type checking. Python supports automatic garbage collections and easily integrated with other languages like C, C++, Java.

Use Case:

- YouTube: Originally it is written in Python and MySQL
- Dropbox: web-based file hosting service
- Yahoomaps uses python
- Google: Many components of google search engine written in python
- YUM: Package management utility in Linux OS
- Zope Corp Blue Bream (a powerful web application server)

What you can do with python?

- Shell scripting
- Gaming programming
- Windows development
- Testing code
- Web development

Special Features:

- Interpreter (Learning made easy)
- Supports Object Oriented Programming
- Other libraries can be attached in python
- Python libraries can be attached with other languages
- Extensive Library
- Open Source

Versions of Python

- CPython (Original)
- Cython (Compatible for c program)
- Jython (JVM based Code)
- IronPython (C# .Net)

Mode of Python Programming:

1. Interactive mode
2. Script mode

1. Interactive mode:

You can write python code using python command line interpreter

Goto command prompt

```
c:> python
```

- Command prompt

`>>>print ("Hello World!")` - Version 3.5 and above

If you are using python version 2.x, you can specify string without brackets

`>>> print "Hello World!"`- Version 2.7 and lower

Use Python as a calculator

```
>>> 25+653+12.5
  690.5
```

To exit from python prompt use exit function

`>>> exit()` or control+Z

2. Script mode:

Use notepad or notepad++ to write scripts

Open new notepad file and add the below statements

```
print ("Hello World")
a = 10
b = 20
c = a+b
print ("The result of c is: ", c)
```

Now save the file as Sample.py

Change the directory to file location

```
> cd E:\python\example>
```

`> dir` : It shows all the files in the current directory

`> type sample.py` : To display file content

`> cls` : To clear screen

`> python sample.py` : To execute the script file

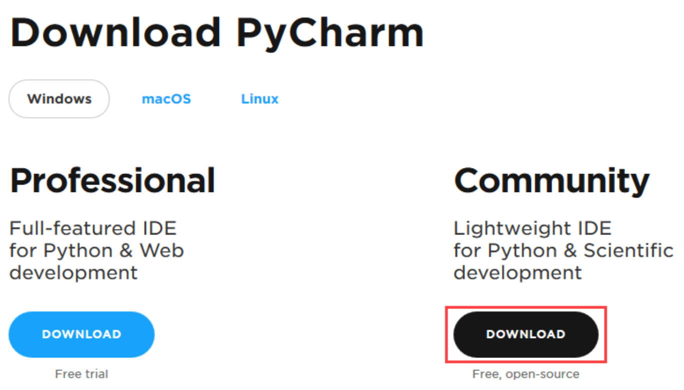
Pycharm:

PyCharm is a cross-platform editor developed by JetBrains. Pycharm provides all the tools you need for productive Python development.

Installing Pycharm

To download PyCharm visit the website

https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.


Download PyCharm
Windows
macOS
Linux
Professional
Full-featured IDE for Python & Web development
DOWNLOAD
Free trial
Community
Lightweight IDE for Python & Scientific development
DOWNLOAD
Free, open-source

Features:

Be More Productive

Save time while PyCharm takes care of the routine. Focus on the bigger things and embrace the keyboard-centric approach to get the most of PyCharm's many productivity features

Get Smart Assistance

PyCharm knows everything about your code. Rely on it for intelligent code completion, on-the-fly error checking and quick-fixes, easy project navigation, and much more.

Boost Code Quality

Write neat and maintainable code while the IDE helps you keep control of the quality with PEP8 checks, testing assistance, smart refactoring, and a host of inspections.

Simply All You Need

PyCharm is designed by programmers, for programmers, to provide all the tools you need for productive Python development.

Chapter 2: Data Types

Memory Operations

- No declaration
- Store values in variable name
- Retrieve value
- Delete variable (del)
- Constants / arithmetic operations decide variable types

Example:

```
>>> 10
>>> a = 10
>>> del(a)
```

Types:

- Int
- Float
- String
- Boolean

Numbers:

- int (10)
- long (0122L) - in python3 no long datatype
- float (15.20)
- Complex (3.4j) - j might be lower case or upper case
- Type conversion
  - int()
  - float()

- long()
- complex()

Example:

`a = int(10)`
`print(a)`
`10`
`b =`
`float("10.56")`
	`print(b)`
	`10.56` -- automatically converts string into float
	`a + b`

Chapter 3: Operators

Operators are the constructs, which can manipulate the value of operands

Example:

```
>>> 4 * 5 = 20
```

Types of Operators:

- Arithmetic Operators
- Relational/Comparison Operators
- Assignment Operators
- Bitwise Operators
- Logical Operators
- Membership Operators
- Identity Operators

Arithmetic Operators

Addition (+) → Add two operands and unary plus

Subtraction (-) → Subtract right operand from the left or unary minus

Multiply (*) → Multiply two operands

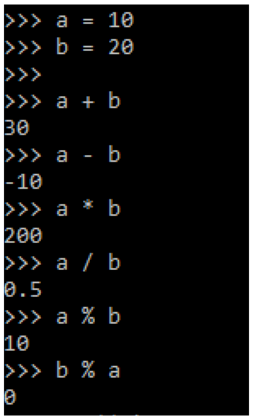
Divide (/) → Divide left operand by the right one (always results into float)

Modulus (%) → remainder of the division of left operand by the right

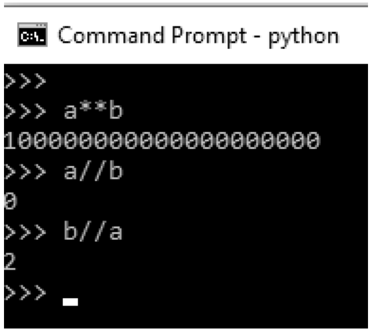
Floor Division(//) → division that results into whole number adjusted to the left in the number line

Exponent (**) → left operand raised to the power of right operand

Example:



```
>>> a = 10
>>> b = 20
>>>
>>> a + b
30
>>> a - b
-10
>>> a * b
200
>>> a / b
0.5
>>> a % b
10
>>> b % a
0
```



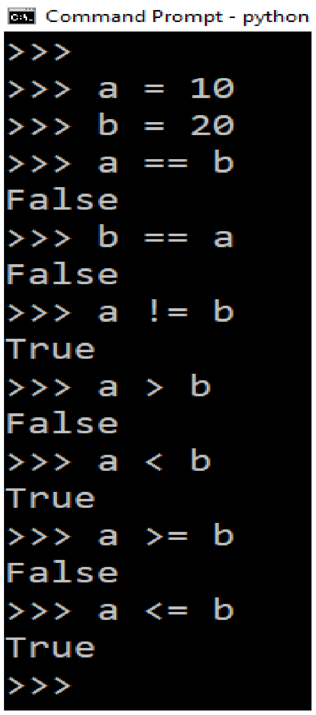
```
>>>
>>> a**b
100000000000000000000
>>> a//b
0
>>> b//a
2
>>>
```

Comparison Operator

These Operators compare the values on either sides of them and decide the relation among them.

- Equal (==)
- Not equal (!=)
- Greater than (>)
- Less than (<)
- Greater than or equal to (>=)
- Less than or equal to (<=)

Example:


Command Prompt - python

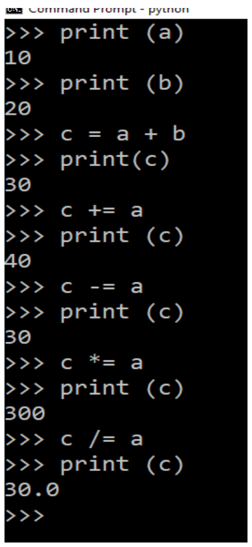
```
>>>
>>> a = 10
>>> b = 20
>>> a == b
False
>>> b == a
False
>>> a != b
True
>>> a > b
False
>>> a < b
True
>>> a >= b
False
>>> a <= b
True
>>>
```

Assignment Operator

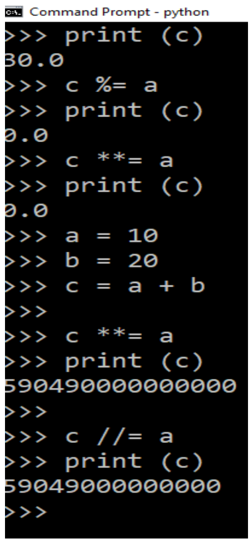
An Assignment Operator is the operator used to assign a new value to a variable.

- Assign (=)
- Add AND (+=)
- Multiply AND (*=)
- Subtract AND (-=)
- Divide AND (/=)
- Modulus AND (%=)
- Exponent AND (**=)

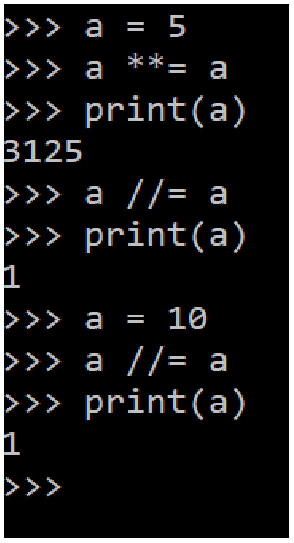
Example:



```
>>> print (a)
10
>>> print (b)
20
>>> c = a + b
>>> print(c)
30
>>> c += a
>>> print (c)
40
>>> c -= a
>>> print (c)
30
>>> c *= a
>>> print (c)
300
>>> c /= a
>>> print (c)
30.0
>>>
```



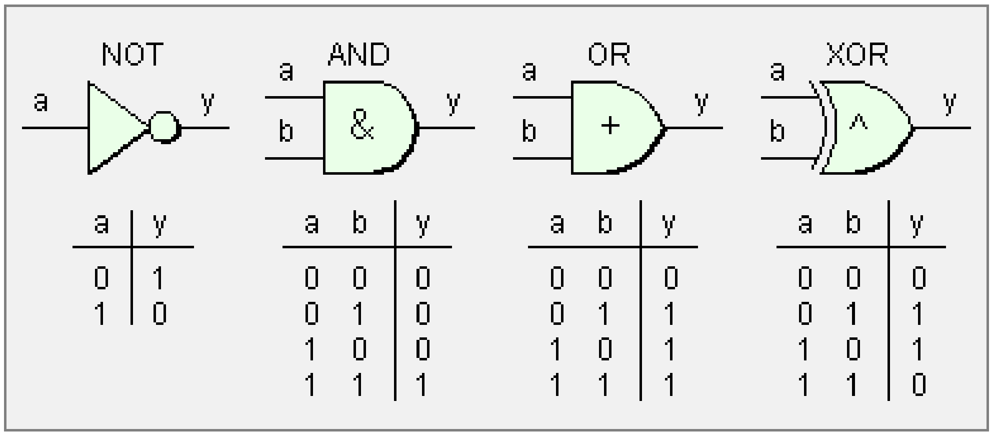
```
>>> print (c)
30.0
>>> c %= a
>>> print (c)
0.0
>>> c **= a
>>> print (c)
0.0
>>> a = 10
>>> b = 20
>>> c = a + b
>>>
>>> c **= a
>>> print (c)
590490000000000
>>>
>>> c //= a
>>> print (c)
59049000000000
>>>
```



```
>>> a = 5
>>> a **= a
>>> print(a)
3125
>>> a //= a
>>> print(a)
1
>>> a = 10
>>> a //= a
>>> print(a)
1
>>>
```

Bitwise Operator

Bitwise operator works on bits and performs bit-by-bit operation bin() can be used to obtain binary representation of an integer number.



a	y
0	1
1	0

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

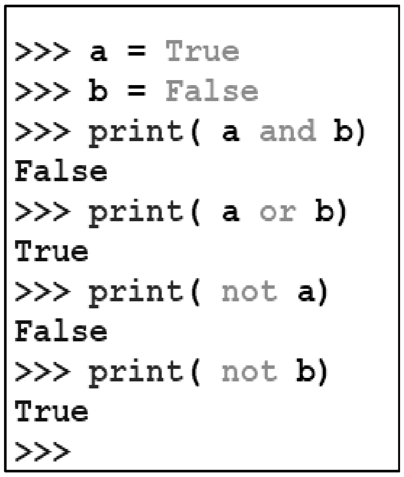
a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

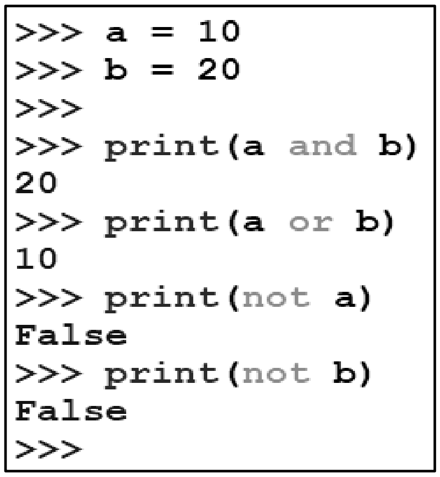
Operator	Description	Example
&	Operator copies a bit, to the result, if it exists in both operands	(a & b) (means 0000 1100)
|	It copies a bit, if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^	It copies the bit, if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)

~	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<<	The left operand's value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>>	The left operand's value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111

Logical Operator



```
>>> a = True
>>> b = False
>>> print( a and b)
False
>>> print( a or b)
True
>>> print( not a)
False
>>> print( not b)
True
>>>
```



```
>>> a = 10
>>> b = 20
>>>
>>> print(a and b)
20
>>> print(a or b)
10
>>> print(not a)
False
>>> print(not b)
False
>>>
```

Membership Operator

These Operators are used to test whether a value or a variable is found in a sequence (Lists, Tuples, Sets, Strings, Dictionaries).

Operator	Description	Example
in	True if value/variable is found in the sequence	`x = list[1,2,3,4,5]` `5 in x`
not in	True if value/variable is not found in the sequence	`6 not in x`

Chapter 4: Sequences

What are sequences?

Sequences are containers with items that are accessible by indexing or slicing. The built-in len() function takes any container as an argument and returns the number of items in the container.

Sequence Operations:

- Concatenation - Combine multiple sequences together
- Repetition - To repeat particular sequence multiple times
- Membership Testing - To check the item is belonging to the particular seq
- Slicing - To slice particular range
- Indexing - Index value for each item (Index starting from 0)

Types of Sequences in Python:

1. Lists
2. Strings
3. Dictionaries
4. Tuples
5. Sets

1. List:

- List is the most versatile data type in python
- It can be written as a list of comma-separated values between square brackets
- Items in a list need not be a same data type

- Similar to arrays in c (but arrays can store similar datatype)

Example:

```
Fruit = ['mango', 'apple', 'orange']
```

Operations:

- ist.append(elem) - To store the new element after the last one
- list.insert(index, elem) - To define where you want to store the value
- list.extend(list2)
- list.index(elem)
- list.remove(elem)
- list.sort()
- list.reverse()

Example 1:

```
Subjects = ['physics', 'Chemistry', 'Maths']
Games = ['Football', 'Cricket', 'Tennis']
Subjects.append('History')      #append operation
print(Subjects)
Subjects.insert(1,'History')  # TO insert into the second
position
print(Subjects)
Subjects.extend(Games)
print(Subjects)
Subjects.remove('Chemistry')
print(Subjects)
Subjects.reverse()          # it prints the elements in the reverse
order
print(Subjects)
print(Subjects + Games)         # Similar to
Subjects.extend(Games)
# Repeat the list twice
```

```
print(Subject * 2)
```

Example 2:

```
list = [ 'abcd', 786, 2.23, 'john', 70.2]
tinylist = [123, 'john']
print (list)          # Prints complete list
['abcd', 786, 2.23, 'john', 70.2]
print (list[0])          # Prints first element of the list
abcd
print (list[1:3])   # Prints elements starting from 2nd till 3rd
[786, 2.23]
print (list[2:])    # Prints elements starting from 3rd element
[2.23, 'john', 70.2]
print (tinylist * 2)     # Prints list two times
[123, 'john', 123, 'john']
print (list + tinylist) # Prints concatenated lists
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
del list[2]
print(list)
list.append("rahul")
print(list)
```

2. Strings:

- Single quote 'Hello World'
- Double quote "Hello World"
- Accessing String
  - o Whole string
  - o Sub string
- Escape Characters '\r', '\n', '\t' etc.
- Escape Characters - '\xnn' for hexa decimal

Example:

```
    Fruit = 'Mango'
```

Operations:

slicing	- string[range]
Updating	- string[range] + 'x'
Concatenation	- String 1 + String 2
Repetition	- String 1 * 2
Membership	- In, Not in
Reverse	- String[:-1]

stg = 'Keep the blue flag flying high'

print(stg.__len__())	- To print length of the characters
print(stg.index('b'))	- To print index value of character 'b'
print(stg.count('f'))	- To check number of times character 'f' repeated
print(stg[0:4])	- To print first 4 characters (4 is ignored)
print(stg[::-1])	- To reverse the string
print(stg.upper())	- To print in upper case
print(stg * 2)	- To print the statements twice
print(stg + 'XXXX')	- To concatenate two strings

Membership testing:

if 'P' in stg:

```
    print('It is an element')
```

Example:

```
    name = 'My name is XXXX'
    print (name)
```

My name is XXXX

```
    name
```

'My name is XXXX' -- it represents variable type

```
a = 10
a
```

10

```
b = float(10)
b
```

10.0

```
c = 10L
c
```

10L

`type (c)` - to display variable type

Substring:

`name [0]` - it prints 1st character from the string

'M'

`name[1]` - it prints last character from the string

'i'

`name[2]` - It prints empty string

''

`name[0:2]` - 0-inclusive, 2-exclusive

'My'

```
name[0:3]
```

'My'

```
name[3:7]
```

'name'

```
name[-4:-1]
```

‘XXX'

```
name[-4:]
```

'XXXX'

```
b = '\x10'
b
```

'\x10'

```
print(b)
```

```
b = 'Hello World\x11'
b
```
'Hello World\x11'

3. Dictionaries:

- Unordered collection of key-value pair
- It is generally used when we have huge amount of data
- Denoted by {}

Example:

```
d = {1:'value','key':2}
print(type(d))
print("d[1]=", d[1])
```

Operations:

Length

del

membership testing

Demo:

```
Student = {'Name':'XXXX', 'Age':33}
print(Student)
```

`print(Student['Name'])` - To print value of 'Name' key

`Student['Gender'] = 'Male'` - To add one more key-value pair to existing dic

`print(Student)`

`Student.pop('Name')` -To remove name key value pair

`Student.clear()` -To clear all key value in a dict or del a dict

`Student['Name'] = 'Santhosh'` - To change the name value

4. Tuples:

- A tuple is a sequence of immutable Python objects
- The tuples cannot be changed
- Tuples are faster than list

```
fruits = ('Mango','Apple', 'Grapes')
```

Operations:

Index	-Tuple.index()
Sclicing	-Tuple[range]
Concatenation	-Tuple1+Tuple2
Repetition	-Tuple * 2
Count	-Tuple.count(elem)

Example:

```
Cricket = ('Dhoni', 'Sachin', 'Virot')
Football = ('Hazard', 'Lampard', 'Terry')
Cricket.append('Dravid')
print(Football)                    - Error because it is updatable
# indexing
print(Cricket[1])
print(Cricket.index('Virot')) - To print particular element
index
# Slicing
print(Football[0:2])               - 0 - Starting, 2 - means actually
(2-1)
# Concatenation
FoodCric = Football + Cricket
print(FoodCric)
# Repetition
print(Cricket * 2)
# count
print(Football.count('Hazard'))
```

Question:

```
Football = [('Dhoni', 'Sachin', 'Virot'),('Hazard',
'Lampard', 'Terry')]
```

- In the above example how the indexing will be created
- Football is a List which contains 2 tuples
- Football list index is 0 and 1
- 0 - tuples index (0, 1, 2)
- 1 - tuple index (0,1,2)

Now i want o print 'Lampard'

```
print(Football[1][1])
```

Now i want to print 'Virot'

```
print(Football[0][2])
```

Example:

```
    Points = [(1,2),(3,4),(6,7)]
    Points.append((5,6))
    print(Points)
    Points.remove((1,2))
    print(Points)
```

5. Sets:

- A set contains an unordered collection of unique and immutable objects(unchanged).
- Sets are denoted by '{}'

Example:

```
    Fruit = {'Mango','Apple','Grapes'}
```

Operations:

Slicing

Add element

clear

copy

Difference

Discard

Remove

Intersection

`Set1 = {1,2,3,4,5,6,6}` - If we print this, '6' is printed only once

```
Set2 = {5,6,7,8,9,10}
print(Set1)
Set1 = {1,2,3,4,5,6}
```

`Set1.discard(4)` - To discard element 4

`print(Set1)`

`print(Set1 | Set2)` - Union Operation(Adding)

`print(Set1 & Set2)` - Intersection (prints common elements

`print(Set1 - Set2)` - Difference

`print(Set1 ^ Set2)` - Symmetric difference (Except common elements in bothsets)

Can we convert list into set?

```
list1 = [1,2,3,4,5,6,5,6,7,8]
print(list1)
Set3 = set(list1)
print(Set3)
```

If you want to convert list into tuple:

```
list1 = [1,2,3,4,5,6,5,6,7,8]
tup = tuple(list1)
print(tup)
```

Chapter 5: Control and Loop Statements

Why to use loop?

If a software developer develops a software module for payroll processing that needs to compute the salaries and the bonus of all the employees.

Example:

salary + bonus = Total salary -> emp1

salary + bonus = Total salary -> emp2

salary + bonus = Total salary -> emp3

Write logic to calculate total salary of all employees

What are loops?

- loops allow the execution of the statement or a group of statement multiple times.
- In order to enter the loop certain conditions are defined in the beginning
- Once the condition becomes false the loop stops and the control moves out of the loop.
- pre-test & Post-test loops
- If the condition is checked before the loop enters
- Condition is checked after the loops ends
- In python, there is no post-test loops

Types of Loops

1. While
2. For
3. Nested

1. While:

- Indefinite or conditional loops.
- It will keep iterating until certain conditions are met
- There is no guarantee ahead of time regarding how many times the loop will iterate.

Syntax:

```
while expression:
     <statements>
```

Example:

```
count = 0
while count < 9:
     print("Number: ", count)
     count = count + 1;
print("Print the Numbers")
```

When you don't know how many iterations are required, use while loop.

2. For Loop:

- Repeats the group of statements a specific number of times
- For loops provides the syntax where the following information is provided:
  - o Boolean condition
  - o Initialization of the counting variable
  - o Incrementation of counting variable

Syntax:

```
for <variable> in <range>:
     <statement 1;>
```

<statement 2;>

<statement n;>

Example 1:

```
Fruits = ["Mango", "Apple", "Orangle"]
for fruit in Fruits:
    print ("Current fruit is: ", fruit)
print("Good bye")
```

Example 2:

```
num = int(input("Number: "))
factorial = 1
if num < 0:
    print("must be positive")
elif num == 0:
    print("factorial = 1")
else:
    for i in range(1, num+1):
        factorial = factorial * i
print(factorial)
```

3. Nested Loop:

- Allows use of loop inside another loop
- Use for loop inside a for loop
- Use while loop inside a while loop

Example: Simulate bank ATM

ENTER 4 digit pin -> check a balance

-> Make a withdrawal

-> Pay In

-> Return Card

```
print("Welcome to ATM")
```

```
restart = ('Y')
chances = 3
balance = 67.14
while chances >= 0:
    pin = int(input('Please enter your 4 digit pin:'))
    if pin == (1234):
        print('You entered your Pin Correctly \n')
        while restart not in ('n', 'no', 'No','N'):
             print("Please press 1 for your Balance \n")
             print("Please press 2 for your Withdrawl \n")
             print("Please press 3 for your Pay in \n")
             print("Please press 4 for your Return Card \n")
             option = int (input ('What would you like to
choose?'))
            if option == 1:
                print('Your balance is Rs.', balance,'\n')
                restart = input('Would you like to go back?')
                if restart in ('n', 'no', 'No','N'):
                   print('Thank you')
                   break
    elif option == 2:
        option2 = ('Y')
        Withdrawl = float(input('How much would you like to
                      withdraw? \n
                      Rs.10/Rs.20/Rs.40/Rs.60/Rs.80/Rs.100')
             if Withdrawl in [10,20,40,60,80,100]:
                balance = balance - Withdrawl
                print("\n your balance is now Rs.", balance)
                if restart in ('n', 'no', 'No','N'):
                       print('Thank you')
                       break
             elif Withdrawl != [10,20,40,60,80,100]:
                   print("Invalid Amount, please Re-try\n")
                   restart = ('Y')
             elif Withdrawl == 1:
                    Withdrawl = float(input('Please enter
                                desired amount:'))
    elif option == 3:
                Pay_in = float(input("How much would you like
                          to pay in?"))
                balance = balance + Pay_in
                print ("Your Balance is now Rs.", balance)
                restart = input("Would you like to go back?")
                if restart in ('n', 'no', 'No','N'):
                   print('Thank you')
                   break
    elif option == 4:
                print("Please wait while your card is
returned...\n")
                print("Thank you for your service")
                break
    else:
             print('Please enter a correct number. \n')
             restart = ('Y')
    elif pin != (1234):
         print('Incorrect Password')
         chances = chances - 1
         if chances == 0:
             print ('\n No more tries')
             break
```

Example 2:

```
     count = 1
     for i in range(10):
         print (str(i) * i)
          for j in range(0, i):
             count = count +1
```

Output:

```
1
22
333
4444
55555
666666
7777777
88888888
999999999
```

Chapter 6: Functions

- Functions are used to perform particular task
- Functions takes some input and produce output or change
- The function is a piece of code that you can reuse
- You can implement your own functions, in many cases use built-in functions

Built-in functions:

Python has many built-in function, you should know what task that function performs

```
L1 = [1,2,3,4,5]
len(L1)          - It writes number of elements in a list
5
sum(L1)          - prints sum of L1
15
print(L1)
[1, 2, 3, 4, 5]
L2 = [5,3,4,2,1]
L2.sort()
print(L2)
[1, 2, 3, 4, 5]      - the input is changed
```

User-defined functions:

Demo 1

```
def add1(a):
-----  b = a + 1     # indents
-----  return b
```

def	- keyword to define a function
add1	- function name
a	- function argument (formal parameter)
return	- keyword to return the output

# call the function

```
add1(5)
6
def add1(a):
...     b = a+1
...     return b
...
add1(5)
6
```

# Save the output of a function in variable c

```
c = add1(10)
print(c)
11
```

# Document String - To create help document

```
def add1(a):
...     """
...     add 1 to a
...     """
...     b = a + 1
...     return b
...
help(add1)
```

Help on function add1 in module __main__:

```
add1(a)
    add 1 to a
```

# Passing - Multiple Parameters

- A function can have multiple parameters

```
def mul(a, b):
...     c = a * b
...     return c
...
 mul(2,3)
6
mul(10,3.15)
```

```
31.5
mul(2,"XXXX") - 2 * "XXXX"
'XXXXXXXX'
```

# No return statement & argument

```
def mj():
...     print("XXXX")
...
 mj()
XXXX
```

# Empty block

```
def nowork():
...
...
File "<stdin>", line 3
^
```

IndentationError: expected an indented block

- use pass to define empty block

```
def nowork():
...     pass
...
 print(nowork)
none
```

- In function if the return statement is not called, python automatically written none

// Assumed

```
def nowork():
...     pass
...return none // automatically returned
```

# Usually functions perform more than one task

```
def add1(a):
...     b = a + 1
...     print(a, "plus 1 equals", b)
...     return b
...
```

```
add1(2)
```

2 plus 1 equals 3 -- Output of Print

3 -- Output of return

# Use for loop within the function

In Python, the enumerate() function is used to iterate through a list while keeping track of the list items' indices.

Example:

```
pets = ('Dogs', 'Cats', 'Turtles', 'Rabbits')
```

Then you'll need this line of code:

```
for i, pet in enumerate(pets):
     print i, pet
```

Your output should be as follows:

```
0 Dogs
1 Cats
2 Turtles
3 Rabbits
```

- use loops within the function

```
def printStuff(Stuff):
...     for i,s in enumerate(Stuff):
...             print("Album", i, "rating is", s)
...
album_rating = [10.0,8.5,9.5]
printStuff(album_rating)
Album 0 rating is 10.0
Album 1 rating is 8.5
Album 2 rating is 9.5
```

# Collecting Arguments

```
def Ast(*names):
...     for name in names:
...             print(name)
...
Ast("XXXX","Santhosh","Monica")
XXXX
```

```
Santhosh
Monica
Ast("Danie","Felix")
Danie
Felix
Ast("Dolly",2)
Dolly
2
```

# Scope Variable: Local vs Global

```
def locvar():
...     Date = 1984 -- Local
...     return(Date)
...
print(locvar())
1984
Date = 1985         -- Global
print(Date)
1985
```

Chapter 7: FILES

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. We can control what kind of operations we can perform on a file with the *mode* parameter of `open`function.

Opening and Closing Files

We use `open()` function to open a file in read or write mode. To return a file object, we use open() function along with two arguments, that accepts file name and the mode, whether to read or write.

Syntax:

```
fileobject = open(filename, mode)
```

Here is a list of opening modes:

`r`	Read-only mode, file must exist
`w`	Write-only mode, creates or overwrites an existing file
`a`	Write-only mode, write always append to the end
`r+`	Read/write mode
`w+`	Read/write mode, creates or overwrites an existing file
`a+`	Read/write mode, write will append to end
`rb`	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
`wb`	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
`ab`	Opens a file for appending in binary format. The file pointer is at

	the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
`rb+`	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
`wb+`	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
`ab+`	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing

The file object attributes: Once a file is opened and you have one *file* object, we can get various information related to that file. A list of all attributes related to file object is here,

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example:

```
# Open a file
fileo = open("abc.txt", "wb")
print "Name of the file: ", fileo.name
print "Closed or not ", fileo.closed
print "Opening mode: ", fileo.mode
```

This produces the below result:

```
Name of the file: abc.txt
Closed or not: False
```

```
Opening mode: wb
```

The close method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close method to close a file.

Syntax:`fileobject.close();`

Example:

```
# Open a file
fileo = open("abc.txt", "wb")
print "Name of the file: ", fileo.name
# Close opened file
fileo.close()
```

Result: `Name of the file: abc.txt`

Sample program:

```
import os.path
import sys
f1=input("enter a source file").strip()
f2=input("enter a target file").strip()
if os.path.isfile(f2):
    print(f2 +"already exists")
infile=open(f1,"r")
outfile=open(f2,"w")
countlines=countcharacters=0
for line in infile:
    countlines+=1
    countcharacters+=len(line)
```

```
        outfile.write(line)
    print(countlines, "lines and",countcharacters,"characters
    copied")
    print("Contents copied from file 1 to file 2")
    infile.close()
    outfile.close()
```

Writing and Reading files

The write method writes any string to an open file. It is important to note that Python strings can have binary data and not just text. The write method does not add a newline character ' \n ' to the end of the string.

Syntax: `fileobject.write(string);`

Here, passed parameter is the content to be written into the opened file.

Example:

```
# Open a file
fileo = open("abc.txt", "wb")
fileo.write("Python is a great language!!\n");
# Close opend file
fileo.close()
```

Result:

```
Python is a great language!!
```

The read method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax: `fileObject.read([count]);`

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

Example:

```
# Open a file
fileo = open("abc.txt", "r+")
str = fileo.read(10);
print "Read String is : ", str
# Close opend file
fileo.close()
```

Result:

```
Read String is : Python is
```