CH4: Basics on Python Language

In the previous section, we have discussed about how python can be used as your desk calculator. In this section we will discuss more about python, in term of a programing language just like C or C++. We will discuss basic element of a programming languages like decision control, loop control, etc. And try to make some small program using python.

**4.1. Mode of programming:**

There are two ways of writing a program in python:

1. Interactive mode programing, where you can write a program step by step in python shell and execute line by line.
2. Script mode programming: Where you can write the program in a file, called python script, and execute that script.

We have discussed the Interactive based programming in previous section and will concentrate more on python scripting in this section.

Let’s start with a simple program “**Write a program which takes two inputs and return the sum”**

In Interactive mode programing, the output would be like this:

>>> x =raw\_input("Enter first number: ")

Enter first number: 10

>>> y =raw\_input("Enter the second number: ")

Enter the second number: 20

>>> z = str(int(x)+int(y))

>>> print "Sum of "+str(x)+' and '+str(y) + 'is : '+str(z)

Sum of 10 and 20 is : 30

This program is doing only three steps:

1. Take the input x and y from user
2. Calculate the sum of x and y and store in z
3. Print z.

We can execute the similar program by writing some script. For that you need to open the notepad and write the following code and save it named by “add.py”.

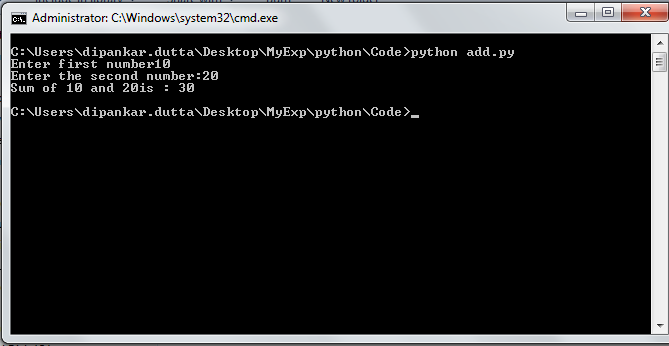
x = raw\_input("Enter first number")

y = raw\_input("Enter the second number:")

z = str(int(x)+int(y))

print "Sum of "+str(x)+' and '+str(y) + 'is : '+str(z)

Now execute this program by running this command: < **python add.py >**



**4.2. Python - As a Programing Language**

4.2.1 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). For Example:

>>> python = 10

>>> Python123 =10

>>> 123Python =10 # It can’t start with number

SyntaxError: invalid syntax

>>> \_abc =10

4.2.1 Python is Case sensitive: It means **python** and **Python** are not same in python.

>>> python = 10

>>> Python = 20

>>> print python # python having small p

10

>>> print Python # python having Cap P

20

>>> print pytHon

Traceback (most recent call last):

File "<pyshell#16>", line 1, in <module>

print pytHon

NameError: name 'pytHon' is not defined

>>>

**4.2.3. Python Naming convention:** It is always advisable to name the variable or a function name by the following standard:

* Class names start with an uppercase letter and all other identifiers with a lowercase letter. Ex: Class Python
* Function name start with small littler like getName(), addName()
* Starting an identifier with a single leading underscore indicates by convention that the identifier is meant to be private.
* Starting an identifier with two leading underscores indicates a strongly private identifier.
* If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

**4.2.4 Reserved Words:** There are some word, called reserved words, may not be used as constant or variable or any other identifier names. These are only used as a programming syntax. The following list shows the reserved words in Python.

|  |  |  |
| --- | --- | --- |
| 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 |

**4.2.5 Lines and Indentation:** Python doesn’t use opening and closing brackets. The blocks are indicated by indentation, which is rigidly enforced. It helps to write clear and more readable code. In the code below, you can see, we define a function hello() where the body is written 4 leading space. Inside if and else block have the line indentation of 8 spaces.

def hello():

a = 10

b = 20

if a > 10:

c = 10

d = 20

else:

e = 10;

f = 20

If you don’t make the indentation proper it will give error, like:

**-----------------------------------------------------------------------------------**

def hello():

a = 10

b = 20

if a > 10:

c = 10

d = 20 # Not indented properly

else:

e = 10;

f = 20

C:\Users\dipankar.dutta\Desktop\MyExp\python\Code>python add.py

File "add.py", line 6

d = 20

^

IndentationError: unindent does not match any outer indentation level

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

**4.2.6 Semicolons are not Compulsory:**  Unlike other languages, there is no need to add semicolon at the end of each statement, like both the below statements are error free in python.

a = b+c

a = b+c;

4.2.7 **Multiple statements in a same line:**  We can have multiple statements in a line by separated by semicolon. Like we can swap two variables like:

>>> a = 10; b = 20; a = a + b; b = a - b; a = a - b; print 'a:',a,'b:',b

a: 20 b: 10

>>>

**4.2.7 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:

python =" I \

Love \

Python \

"

print \

python

**4.2.8 Quotation in Python:** In python single ('), double (") and triple (''' or """) quotes are used 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. For example:

python = "I love python"

print python

python =' I love python'

print python

python = """I

Love

Python"""

print python

python 'I love python"

The last line throws an error like:

C:\Users\dipankar.dutta\Desktop\MyExp\python\Code>python add.py

File "add.py", line 9

python 'I love python"

^

SyntaxError: EOL while scanning string literal

**4.2.9 Comments in Python:** Python comments are simple as putting “#” at the begging. A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment and the Python interpreter ignores them. Example:

# This is first Comment

print 'hello python' # This is second comment

print “#this is not a comment”

**4.2.10. Basic input and output in python:** Python use raw\_input() function to take a user input and print is used to display the output. raw\_input() always returns string , hence if we want to take an integer, we need to do the typecast before using int.

Example:

x = raw\_input("Enter first number")

y = raw\_input("Enter the second number:")

z = str(int(x)+int(y))

print "Sum of "+str(x)+' and '+str(y) + 'is : '+str(z)

**4.3: Python Variables**

Variable are the nothing but the naming of some reserve storage which contain some value. Thus when we declare some variable, some memory is reserved for that variable.

For example: when we write x = 10, indicates we reserve a storage for 10 and name it as x

Unlined C or java, python is a type less language. That means we don’t need to specify types of a variable x can hold integer or a string. In fact, It just a pointer, hence it can point anything. In python all variables are references.

For example:

>>> x = 10 # Here we are make a storage which hold 10 and x is pointing to 10

>>> print x # It will print 10 as x is pointing to this location

10

>>> x = 'python' # here we create another space which hold”python” and x is pointing to “python “ now

>>> print x # it will print python as x is pointing to “python”

Python

**4.3.1 Python is Type-less:** Python doesn’t require specifying data type while declaration. Thus we simply assign different value easily like this

>>> myInt = 10 # Define an inetger

>>> myFloat = 10.5 # Define float

>>> myString = “python” # Define a string

>>> print myInt, myFloat, myString

**43.2 Multiple Assignment:** Python support multiple and chain assignment, for example we are assigning x , y, z to 1 and a,b,c to 1,2,3 respectively.

>>> x = y = z = 1 # chain Assignment

>>> print x,y,z # Print

1 1 1

>>> a,b,c = 1,2,3 # Multiple assignment

>>> print a,b,c # Print this

1 2 3

**4.3.3 Deleting a Python Variable.** We can also delete python variable using “**del”** keyword.

>>> x ="python" # Define x

>>> print x # Print x is successful

'python'

>>> del x # Now delete x

>>> print x # Now print fails

Traceback (most recent call last):

File "<pyshell#61>", line 1, in <module>

print x

NameError: name 'x' is not defined

>>>

**4.4 Python Standard Data Types:** Python provide 5 high level data type to be used in python program, used for different purpose. These are:

1. Number – basically integer of float
2. String – Indicate the string data
3. List – collection of python objects and variable
4. Tuple – like list but non-mutable
5. Dict – Dictionary which holds key-value pair

**4.4.1 Numbers:**  This kind of data type basically holds number object like integer or float. Python support 4 kinds of numbers, as bellow:

1. int - positive or negative whole numbers with no decimal point.
2. long - integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
3. float - represent real numbers and are written with a decimal point dividing the integer and fractional parts.
4. complex - are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 .

The following table shows the example of each of number type:

|  |  |  |  |
| --- | --- | --- | --- |
| Int | Long | float | complex |
| 10 | 51924361L | 0.0 | 3.14j |
| 100 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 080 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| -0490 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

**Type Conversion:**

It is possible to convert one type of number type into another type, the below example shows that:

>>> x =10.5; print int(x) # convert integer from float

10

>>> x =10 ; print float(x) # convert integer to float

10.0

>>> x = 10; print long(x) # convert interer to long

10

>>> x =10; y=10; print complex(x,y) # convert interger to complex

(10+10j)

>>>

**Mathematical functions:** Python provide a huge number of mathematical functions on number object. Some of the examples are given below:

>>> import math # Math module offer number of function

>>> x = -30; print abs(x) # Find the Absolute value

30

>>> x= 10.5; print math.ceil(x) # Find out the Ceiling value

11.0

>>> x = 10.5 ; print math.floor(x) # Find out the floor value

10.0

>>> x = 10; print pow(x,3) # return X to the power y

1000

>>> x = 10.123454321 ; print round(x,3) # Round the number upto 3 decimal point

10.123

>>> max (1,3,4,5,7,9) # Find the max of the number

9

>>> min (3,5,7,8,9) # Find the minimum of the collections

3

>>> cmp(10,11) # compare two number : return 0 if they are same

-1

>>> print math.sqrt(81), math.sqrt(30) # Print Squire root.

9.0 5.47722557505

>>>

There are lot of other function, can be found in Python documentation.

**4.4.2: Python Strings.**

Python strings are very popular data type. It is a collection of character enclose in single quote, double quote or triple quote.

Some of python String are:

>>> myStr1 = 'python'

>>> myStr2 = "python"

>>> myStr3 ="""Hello World"""

>>> print myStr1,myStr2,myStr3

python python Hello World

**4.4.2.1 String Manipulation Operator:** Python supports different string manipulation operator like +, \*,[] etc. Some examples are as below.

>>> x = 'hello'; y = 'world' # Define two string x and y

>>> print x + y # String Concatenation using + operator

helloworld

>>> print y\*3 # Self concatenation using \* Operator

worldworldworld

>>> print x[0] # get 0th character of ‘hello’

h

>>> print x[-1] # get -1th Character of ‘hello’

o

>>> print x[1:4]# get substring from 1st char to 4th char, but not taking 4thchar

ell

>>> print x[-3:-1] ]# get substring from -3st char to -1th char, but not taking -1th char

ell

>>> 'h' in x # is ‘h’ present in ‘hello’

True

>>> 'h' not in x # is ‘h’ not present in hello

False

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| + | Concatenation - Adds values on either side of the operator | a + b will give HelloPython |
| \* | Repetition - Creates new strings, concatenating multiple copies of the same string | a\*2 will give -HelloHello |
| [] | Slice - Gives the character from the given index | a[1] will give **e** |
| [ : ] | Range Slice - Gives the characters from the given range | a[1:4] will give **ell** |
| in | Membership - Returns true if a character exists in the given string | **H in a** will give 1 |
| not in | Membership - Returns true if a character does not exist in the given string | **M not in a** will give 1 |

**4.4.2.2 Python index and Stepping:**

Like c, python also use indexing for string : like x= “HELLO”. This x[0] is ‘h’, x[1] is e and so on. We will discuss more about indexing Stepping in list section. Same concept will be applied here.

**4.4.2.3 Python String formatting.**

Like printf() in C, python also provide a way to do string formatting using % operation. Some example is given below:

>>> x = 'python'

>>> y = 20

>>> print 'Welcome come to %s: version %d ' % (x,y)

Welcome come to python: version 20

>>>

As you can see %s and %d is replaced by value of x and y.

Some of the formatting style is listed below:

|  |  |
| --- | --- |
| Format Symbol | Conversion |
| %c | Character |
| %s | string conversion via str() prior to formatting |
| %i | signed decimal integer |
| %d | signed decimal integer |
| %u | unsigned decimal integer |
| %o | octal integer |
| %x | hexadecimal integer (lowercase letters) |
| %X | hexadecimal integer (UPPERcase letters) |
| %e | exponential notation (with lowercase 'e') |
| %E | exponential notation (with UPPERcase 'E') |
| %f | floating point real number |
| %g | the shorter of %f and %e |
| %G | the shorter of %f and %E |

**Python String manipulation Methods:** python support number of useful string manipulation methods discussed as below:

# Define a String

>>> x = 'HelloWorld'

>>> print x

HelloWorld

# Find out the length of it

>>> print len(x)

10

# do the captilization of it

>>> print str.capitalize(x)

Helloworld

# find the index of first match on ‘ow’

>>> print x.find('oW')

4

# find the index of first match from right to left

>>> print x.rfind('ow')

-1

# Find id x has a digit

>>> print x.isdigit()

False

# Check if x ends with ‘rld’

>>> print x.endswith('rld')

True

# could number of l in ‘helloworld’

>>> print x.count('l')

3

# find max alphabet in ‘helloworld’

>>> print max(x)

R

# reverse HelloWorld

>>> print x[::-1]

dlroWolleH

# replace all ‘l’ by ‘DD’

>>> print x.replace('l','DD')

HeDDDDoWorDDd

# Split the string into list based on ,

>>> print "abc,abc,bc,rr".split(',')

['abc', 'abc', 'bc', 'rr']

# Join the list of string with ‘:’ in between

>>> print ':'.join(['a','b','c','d'])

a:b:c:d

**4.4.3. Python List**

The list is a most versatile data type in Python which can be written as a list of comma-separated values between square brackets. This is similar kind of **array** in C/C++, but it is more powerful than an “array”. Unlike C, Lists are heterogeneous, thus it allow to store mixed data, for example, a list might contain integer, float, string, as well as an another list.

Below are the examples of a list.

>>> myList =[] # This is a null list, doesn’t contain anything

>>> print myList # It will print []

>>> myList1 = [1,2,3.5,'Python','Djnago']

>>> print myList1 # This will also print [1, 2, 3.5, 'Python', 'Djnago']

>>> myList2 = ['Hello','World',myList1]

>>> print myList2 # will print['Hello', 'World', [1, 2, 3.5, 'Python', 'Djnago']]

myList is a null list, it don’t contains anything. Mylist1 contains five different kinds of objects. MyList2 contains two string objects and myList1. myList2 is the example of nested list.

**Basic Operation on List:**

1. let’s Create an Empty List
2. Add two item in that list :’python’’django’
3. Get first element from that list
4. Update the first item in that list
5. Add another list with that list myList2=[1,2,3,4]
6. Print all element in that list
7. Delete 4th element from that list
8. Find the length of the list
9. Repeat 3 type that list
10. Check ‘perl; is the member of that list or not
11. Delete the whole list

Ans:

>>> myList = []

>>> myList.append('python')

>>> myList.append('djnago')

>>> print myList # Print ['python', 'djnago']

>>> print myList[0] # return 0th Element :python

>>> myList[0] = 'python1' # Assign 0th elements

>>> print myList # print mylist ['python1', 'djnago']

>>> myList = myList + [1,2,3,4] #Adding two list using + operation

>>> print myList # Added list['python1', 'djnago', 1, 2, 3, 4]

>>> for i in myList: # Iterate over the list

print i

python1

djnago

1

2

3

4

>>> del myList[3] # Delete 3th index of the list

>>> print myList

['python1', 'djnago', 1, 3, 4]

>>> print len(myList) # Find length of the list =>5

>>> myList =myList\*3 # repeat list for 3 times :

>>> print myList

['python1', 'djnago', 1, 3, 4, 'python1', 'djnago', 1, 3, 4, 'python1', 'djnago', 1, 3, 4]

>>> 'perl' in myList # Check membership of element =>False

>>> del myList # Delete the full list, Hence it is not accessible farther

>>> print myList # will return error

Traceback (most recent call last):

File "<pyshell#31>", line 1, in <module>

print myList

NameError: name 'myList' is not defined

>>>

Basic Operation is listed in table as below:

|  |  |  |
| --- | --- | --- |
| Python Expression | Results | Description |
| len([1, 2, 3]) | 3 | Length |
| [1, 2, 3] + [4, 5, 6] | [1, 2, 3, 4, 5, 6] | Concatenation |
| ['Hi!'] \* 4 | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition |
| 3 in [1, 2, 3] | True | Membership |
| for x in [1, 2, 3]: print x, | 1 2 3 | Iteration |

**List Indexing, Slicing and Stepping**

Indexing of the list starts from 0, left to right. Hence first element in the list is the 0th index elements. Indexing is used to random access the list element. myList[3] will return 4th element in the list.

Example

**>>> myList=[0,1,2,3,4,5]**

**>>> myList[5] # Return 5**

**>>> mylist[0] # return 0**

Negative indexing is a new concept in python. It’s start from -1, from right to left, that means, last element have an index of -1, second last element have an index of -2 and so on.

Example:

**>>> myList =[-5,-4,-3,-2,-1]**

**>>> myList[-1] # return-1**

**>>> myList[-4] # return -4**

**>>>**

Slicing is the ability to create a list (in the abstract data type sense) from another list by cutting pieces of that other list.

Example:

**>>> myList = [0,1,2,3,4,5]**

**>>> myList[2:4] # Return second to 3rd index: [2, 3]**

**>>> myList[2:] # return 2nd to rest [2, 3, 4, 5]**

**>>> myList[:3] # return begning to 2nd index [0, 1, 2]**

**>>> myList[:] # return Full List [0, 1, 2, 3, 4, 5]**

**>>> myList[300:] # reyurn Null list []**

**>>>**

**>>> myList =[ -5,-4,-3,-2,-1]**

**>>> myList[-4:-2] Retunr -4th Index to -3 index [-4, -3]**

**>>> myList[-4:] # return -4th index to rest[-4, -3, -2, -1]**

**>>> myList[:-3] # return beg to -4 index[-5, -4]**

**>>>**

**Stepping/ Sliding** allows you to specify an increment between the elements to cut from your list.  If you wanted to slice every 2 elements, instead of specifying an increment of 1, you specify 2.

Syntax:

**l[start:end:step] # slices from start to end but slide by the step value**

**l[::] # slices through the entire list i.e. does a copy of the list**

Example:

**>>> myList = [0,1,2,3,4,5,6,7,8,9,10]**

**>>> myList[0:11:1] # Print 0th to 10th index with step of 1 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

**>>> myList[0:11:2] print 0th index to 10 th index with step of 2 [0, 2, 4, 6, 8, 10]**

**>>> myList[::2] #Same as before only Fiest two argument is missing[0, 2, 4, 6, 8, 10]**

**>>> myList[::3] Print whole list with step 3 [0, 3, 6, 9]**

**>>> myList[::-1] # print Whole list with step 1 from Right to Left, Hence Reverse**

**[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]**

**>>> myList[::-2] # print Whole list with step 2 from Right to left**

**[10, 8, 6, 4, 2, 0]**

**>>> myList[::-3] # Print whole list with step 3 from R2L [10, 7, 4, 1]**

**>>> myList[1:7:2] # print list from 1st index to 6th index with step 2 [1, 3, 5]**

**List Build In functions:**

Let discuss some of the build in function with example.

1. Find length of the list, Max and Min elements
2. Convert a String to list of character
3. Remove an Object from List
4. Append an item at the end of list
5. Remove an Object from First location from a list
6. Reverse that list
7. Sort that list

Ans:

**>>> myList = [4,5,6,1,9,0]**

**>>> print len(myList) # Print the length 6**

**>>> print max(myList) # Print the Maximum element 9**

**>>> print min(myList) # print the minimum Element 0**

**>>> print list('python') # Break the String into a list of char**

**['p', 'y', 't', 'h', 'o', 'n']**

**>>> myList.remove(4) # Remove 4 thus myList is [5, 6, 1, 9, 0]**

**>>> myList.append(4) # Add 4 at end Hence myList is [5, 6, 1, 9, 0,4]**

**>>> myList.pop(0) # remove 0th elemnt and return 5 hence list is [6, 1, 9, 0, 4]**

**>>> myList.reverse() # reverse , Now myList is [4, 0, 9, 1, 6]**

**>>> myList.sort() # Sort it now my list id [0, 1, 4, 6, 9]**

**4.4.4 Python Tuple.**

Tuple are similar as list data-type. Only difference is that tuples are immutable, whereas list is mutable.

A data-type is said to be a mutable if we can change the data structure by adding new elements or removing elements. Lists are mutable, as we can add or remove elements from a list, but tuples are not mutable. Once it is declared, we cannot change it.

Creating a tuple is as simple as putting different comma-separated values and optionally you can put these comma-separated values between parentheses also. For example:

>>> myTuple1 = () # is an empty tuple

>>> myTuple2 = (1,) # typle having one element, Note that the Comma is is required

>>> myTuple3 = (1,2,3,4) # tuple of 4 4lements

>>> mytuple4 = ([1,2,3],'python',(1,2,3)) # tuple having list and tuple in it

>>> print myTuple1,myTuple2,myTuple3,mytuple4

() (1,) (1, 2, 3, 4) ([1, 2, 3], 'python', (1, 2, 3))

**Tuple’s are immutable.**

As we define a tuple, we cannot change it. Like, we can’t add any element to it, even if we can’t modify any value. Thus following operation is invalid:

**>>> mytuple = (1,2,3)**

**>>> mytuple = mytuple +(3,4,5) # Invalid**

**>>> mytuple[0]=6 # invalid**

**If we want to modify tuple, the we can create a new tuple and assign to it.**

**>>> mytuple1 = mytuple +(3,4,5) # it will create a new one**

**>>> mytuple2 = (6,)+mytuple[1:] # it will make (6,1,2,3)**

**Operation of tuple:** Like list, tuple also supports +, \*, indexing, slicing and stepping operation.

>>> mytuple1 = (1,2,3); mytuple2 = (4,5,6) # Define two tuple

>>> print mytuple1 + mytuple2 # Add two tuple and print (1, 2, 3, 4, 5, 6)

>>> print mytuple1 \* 3 # Repeat 3 times=>(1, 2, 3, 1, 2, 3, 1, 2, 3)

>>> print mytuple1[0] # get First Elemnet => 1

>>> print mytuple1[-1] # get last Elemts => 3

>>> print mytuple1[1:3] # Do slicing =>(2, 3)

>>> print mytuple1[-3:-1] # Do Slicing over negetie slicing => (1, 2)

>>> print len (mytuple1) # Find out length 3

>>> 4 in mytuple1 # The the member ship of 4 => False

>>> mytuple1[0] = 0 # Tuple Assigmet is Invalid hence it raise the Error

Traceback (most recent call last):

File "<pyshell#34>", line 1, in <module>

mytuple1[0] = 0

TypeError: 'tuple' object does not support item assignment

>>> mylist1 = list (mytuple1) # Convert tuple to list

>>> mylist1

[1, 2, 3]

>>> mytuple = tuple(mylist1) # Convert list to tuple

>>> mytuple

(1, 2, 3)

>>> print tuple('python') # generate tuple of character of a string

('p', 'y', 't', 'h', 'o', 'n')

**4.4.4 Python Dictionary:**

Dictionaries are useful when we want to fast lookup, for example if we want to store the information of the student in a class, we can store the information of student in two ways:

1. List of tuple :

studentbase1=[

(1,'xyz','kolkata'),

(2,'ram','khragpur'),

(3,'hari','banaglore')

]

1. Dictionary :

studentbase2 ={

1:('xyz','kolkata'),

2:('ram','khragpur'),

3: ('hari','bangalore')

}

The main drawback of the first data structure is that, it is slower if you want to get the info of roll no 1. It need to traverse whole list, compare with roll numbers and return it. Where-as, in second representation, we will create a HashTable having key as roll number and value as student’s info. Thus lookup is faster than Case 1, as iteration over list is not required.

Python dictionaries are also known as associative arrays or hash tables. The general syntax of a dictionary is as follows:

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

>>> dict1 = {} # empty Dict

>>> dict2 = {1:1,2:2,3:3}

>>> dict3 = {'ram':[1,2,3],'sam':'hellopython'}

>>> print dict1,dict2,dict3

{} {1: 1, 2: 2, 3: 3} {'ram': [1, 2, 3], 'sam': 'hellopython'}

Two important properties of dict are:

1. Unique Keys: Keys are unique within a dictionary while values may not be.
2. Immutable Keys: The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Thus following are invalid:

>>> dict1 = {'a':1,'b':2,'a':10} # Here are two same keys:a, but last one is stored, Python Interpreter doesn’t raised any error

>>> dict1

{'a': 10, 'b': 2}

>>> dict2 ={[1,2,3] : '233' } # List can’t be used as a key, It will raised an error

Traceback (most recent call last):

File "<pyshell#59>", line 1, in <module>

dict2 ={[1,2,3] : '233' }

TypeError: unhashable type: 'list'

>>>

**Manipulation of Python Dictionary**

Python Dict support all kind of hash table operation, like insertion, deletion , update, lookup etc. The following example illustrates this:

1. Create an empty student dict
2. Insert two entry < roll: name> 1: ‘ram’ and 2: ;jadu
3. Get the name of student having roll no 1
4. Change the name of the student of roll 1.
5. Delete the information of roll 1.
6. Add some more student
7. Get list of keys and values
8. Convert dict to list of tuple
9. Convert a tuple to list
10. Delete the full dictionary

Ans:

>>> student = {} # create empty dict

>>> student[1] = 'ram' # Adding a Student

>>> student[2] = 'jadu' # Inserting another student

>>> print student # print >{1: 'ram', 2: 'jadu'}

>>> print student[1] # get student name of roll no 1 =>ram

>>> student[1] = 'Hari' # update the name of student having roll 1

>>> print student # print again

{1: 'Hari', 2: 'jadu'}

>>> del student[1] # Delete student 1

>>> student

{2: 'jadu'}

>>> student[3] = 'Dipankar' # Add few more students

>>> student[4] = 'python'

>>> student[5] = 'django'

>>> student.keys() # get all roll numbers

[2, 3, 4, 5]

>>> student.values() # get all values

['jadu', 'Dipankar', 'python', 'django']

>>> tt = student.items() # convert dict to list of tuple [ <key>,<val>, …]

[(2, 'jadu'), (3, 'Dipankar'), (4, 'python'), (5, 'django')]

>>> s = dict(tt) # Convert List of tuple to dict

{2: 'jadu', 3: 'Dipankar', 4: 'python', 5: 'django'}

>>> 2 in student # find out the if key 2 is presnt in that dict or not

True

>>> student[10] # get an Not existing student: it will raise an KeyError

**Traceback (most recent call last):**

**File "<pyshell#88>", line 1, in <module>**

**student[10]**

**KeyError: 10**

>>> student.get(10,'Not fount') # Get an not exising student, if not found returns an objects, But don’t raise an exception.

'Not found'

**4.4.6 Python Sets:**

Set is unordered collection of distinguish elements. Thus the set has two important properties

1. It is an un-order collection, hence, the order of elements does not matter.
2. All the elements in the set are distinguishable there should not be same elements twice.

In python we can define set using “set” keyword. Just like mathematics, set also support number of operation like insertion, union, difference etc. Below examples, illustrate the operation on set.

1. Convert a list to set by removing duplicates.
2. Find unique element in a list
3. Add / remove elements in a set.
4. Check membership of an elements in a set
5. Create two set {1,2,3,4,5} and {3,4,5,6,7}
6. Find insertion, union, OR and difference of the set

Ans:

>>> set1 =set([1,2,3,4,5]) # Create a Set from List => set([1, 2, 3, 4, 5])

>>> set2 = set((1,2,3,4,5)) # Create a set from tuple => set([1, 2, 3, 4, 5])

>>> set3 = set('python') # Creae a set from String =>set(['h', 'o', 'n', 'p', 't', 'y'])

>>> print list(set([1,2,3,4,5,4,3,2,1])) # Remove Duplicates from List=>

[1, 2, 3, 4, 5]

>>> set1.add(6) # add 6 to set1 => set([1, 2, 3, 4, 5, 6])

>>> set1.remove(5) # Remove 5 from set1 =>Set([1, 2, 3, 4, 6])

# Create two set

>>> set1 =set([1,2,3,4,5])

>>> set2 = set([3,4,5,6,7])

>>> set1 & set2 # Find Intersection of set1 and set2 =>set([3, 4, 5])

>>> set1 | set2 # Find Union of set1 and set 2= >set([1, 2, 3, 4, 5, 6, 7])

>>> set1 -set2 # find Diff of set1 to set2 => set([1, 2])

>>> set2 -set1 # Find diff of set2 to set 1 =>set([6, 7])

>>> set1 ^ set2 # Find symmetric difference

set([1, 2, 6, 7])

>>> 1 in set1 # is 1 belongs to set 1? True

>>> 10 in set1 # is 10 belongs to set1 ? False

>>> print list(set1) # Convert set to list => [1, 2, 3, 4, 5]

**4.5 Python Control Blocks:**

Control blocks are the basics of any programming languages, which basically control the flow of the execution of a program. There are two basic Control blocks in python

* **Decision control -** which allow, branching in code depends on some conditions.
* **Loop control –** which allow execution of same steps multiple times.

In this section we will discussion more of these control.

4.5.1. *Decision control Statements:*  Python Support 3 kind of decision control:

1. If – Else Statements:

The basic example is as below:

x = 1

if x == 1:

print ' It;s 1'

else:

print ' it;s not 1'

We assign x to value 1 and checking the condition, if it is true the else block going to execute, otherwise, else block going to execute.

If –ElIf-Else Example:

x = 1

if x < 0:

print 'Less than Zero'

elif x == 0:

print 'Equals to Zero'

else:

print 'Greter than Zero'

Nested If Else block

We can nest one if-else block into another if-else block. For example we want to find out Maximum of three numbers.

print 'Enter 3 number:'

x=int(raw\_input())

y=int(raw\_input());

z=int(raw\_input());

if x > y:

if x >z:

print 'Max: ',x

else:

print 'Max: ',z

else:

if y > z :

print 'Max: ',y

else:

print 'Max: ',z

Output :

>>>Enter 3 number:

1

2

3

Max: 3

>>>Enter 3 number:

3

2

1

Max: 3

4.5.2. *Loop control Statements:* A loop statement allows us to execute a statement or group of statements multiple times. Python Support 3 kind of decision control

***4.5.2.1 While Loop:*** For example **Find out the sum of 1 to 10 ?**

i = 0;

sum =0

while i <= 10:

sum = sum +i

i = i+1

print 'Sum of 1 to 10 is :', sum

Output :

Sum of 1 to 10 is : 55

***4.5.2.2 For –in Loop* :** We can do the same calculation using for loop as below:

data = [1,2,3,4,5,6,7,8]

sum =0

for i in data: # it will iterate for each value of data

sum= sum +i

print ' Sum of 1 to 10 is :' ,sum

Output :

Sum of 1 to 10 is : 55

***4.5.2.3 Range() Function:***

Range() is a function which generate arithmetic progression. Like range(10) will generate a list from 0 to 9 . Thus range(N) generate a list containing number [0,1,2… N-1]

Example:

>>> print range(10) # Generate 0 to 9

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> print range(5,10) # generate 5 to 9

[5, 6, 7, 8, 9]

>>> print range(0,10,2) # generate 0 to 9 with Step of 2

[0, 2, 4, 6, 8]

>>> print range(0,10,3) # generate 0 to with step of 3

[0, 3, 6, 9]

>>> range(10,0,-1) # Generate 10 to 1

[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

***4.5.2.4 Break and Continue Statement***

Like C, Break is a statement to breakout the loop and continue is a statement to continue the loop for next iteration without executing rest of the body of loop.

For example: the following code show how we can check for a number is prime or not. To test it, we try to divide it from 2 to N-1. If it is divisible by any number ( say i) we can conclude that the number is not prime. Hence it’s meaning less to continue for testing i+1 to N-1. We can do that using loop.

i = None

N =int(raw\_input('Enter the Number:'))

for i in range(2,N):

if N%i == 0:

print 'Composite'

break

if i == N-1:

print 'Prime'

Output :

Enter the Number:10

Composite

>>>

Enter the Number:19

Prime

>>>

Now suppose, we want to find the sum of all number from 1 to N. the code is as simple as:

sum =0

N =int(raw\_input('Enter the Number:'))

for i in range(1,N+1):

sum = sum + i

print 'Sum: ',sum

Output :

>>>

Enter the Number:10

Sum: 55

>>>

Now if we want to find out the sum of all even numbers, we need to skip summing odd number, which can be done like this:

sum =0

N =int(raw\_input('Enter the Number:'))

for i in range(1,N+1):

if i%2 != 0:

continue # Donot do sum but contine for next

sum = sum + i

print 'Even Sum: ',sum

>>>

Enter the Number:10

Even Sum: 30

>>>

***4.5.3.5 For-Else:***

If you see the **CheckPrime** program given above, to decide a number is prime, we need to test that the loop is fell through and none of <2 to N-1> can divide N. For that we need to do a check *i==N-1 at the end of the loop . A*better way to check this *loop through condition* by putting an ELSE block after this for loop.

The modified code is:

N =int(raw\_input('Enter the Number:'))

for i in range(2,N):

if N%i == 0:

print 'Composite'

break

else: #This else part will execute if the loop is fall through without break

print 'Prime'

Output :

>>>Enter the Number:11

Prime

>>> Enter the Number:81

Composite

>>>

***4.5.2.6 PASS Keyword.***

How can I write an empty body for loop or if else block? The following code raises an error:

for i in range(10):

Output:

File "max\_of\_three.py", line 3

^

IndentationError: expected an indented block

That means we can’t have empty body for or if block. But sometime it requires. Pass keyword is used as a place holder for something to do in near feature or to avoid a syntax error for if or for loop. The below code is perfectly valid:

for i in range(10):

pass # Pass says Don’t do anything, We can add more code later on.

***4.5.2.7 Enumerate() and Zip() Function***

Some time, when we iterate over list by a for loop, we need the value as well the index. One way to do that, we can use a variable i and increment in the body of the loop. A better way for doing that using enumerate() function, which return index and value both :

>>> myList = ['python','java','c','c++']

>>> enumerate(myList)

<enumerate object at 0x02A33D78>

>>> for i,v in enumerate(myList): # i is for Index and v is for Vlaue

print i,' : ',v

Ouptput:

0 : python

1 : java

2 : c

3 : c++

>>>

Zip is used when we want to iterate more than one list at the same time. For example, a student database is maintained by two lists: one contains the roll number and another contains name. We would like to iterate both on same time and print student data:

>>> roll = [1,2,3]

>>> name = ['ram','sam','jadu']

>>> for r, n in zip(roll,name):

print 'Roll:',r,' Name: ',n

**output**

**-------**

Roll: 1 Name: ram

Roll: 2 Name: sam

Roll: 3 Name: jadu