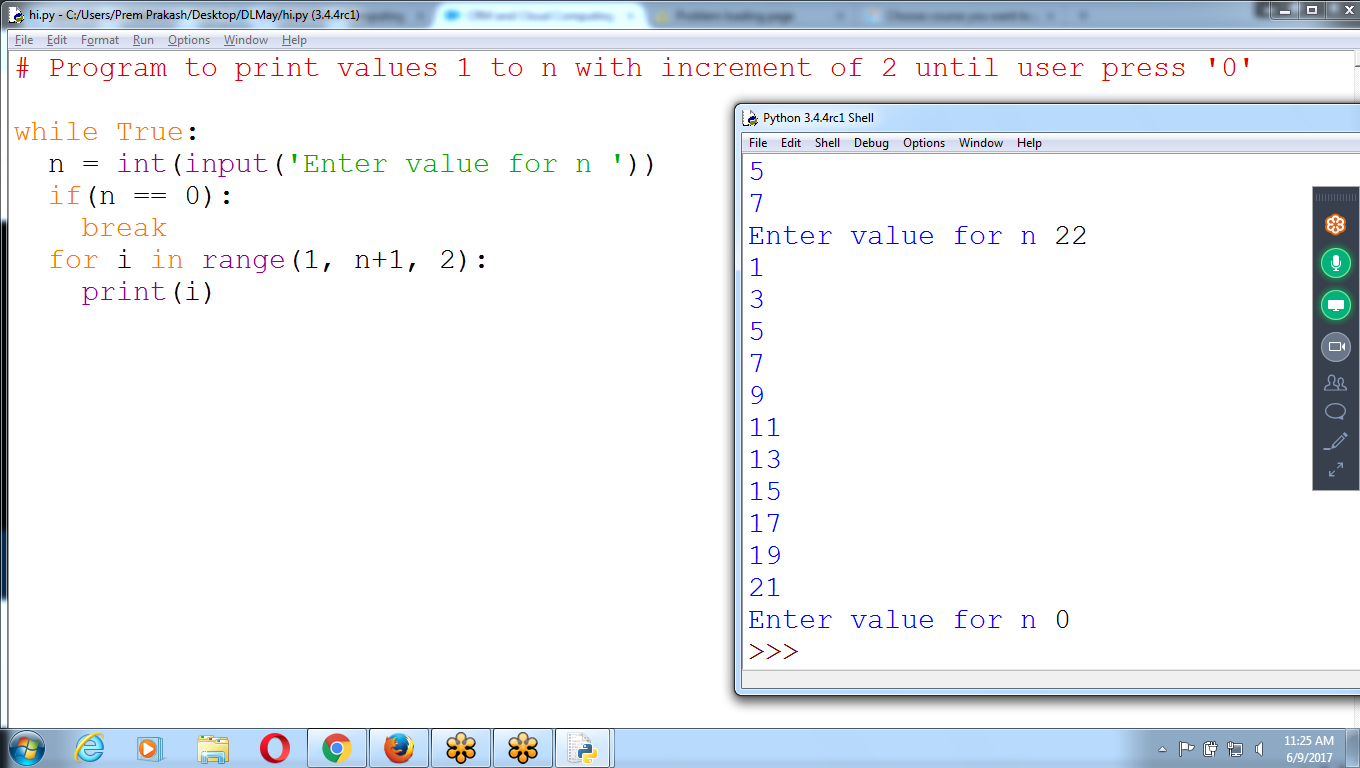
**Installation, Data types,**

**Decision Types ( if, elif, else)**

**Loops (for, while)**

**Functions (parameters)**

**Collections (List, Dictionary)**

****

**# Print 1 to n values, until user press '0'**

**while True:**

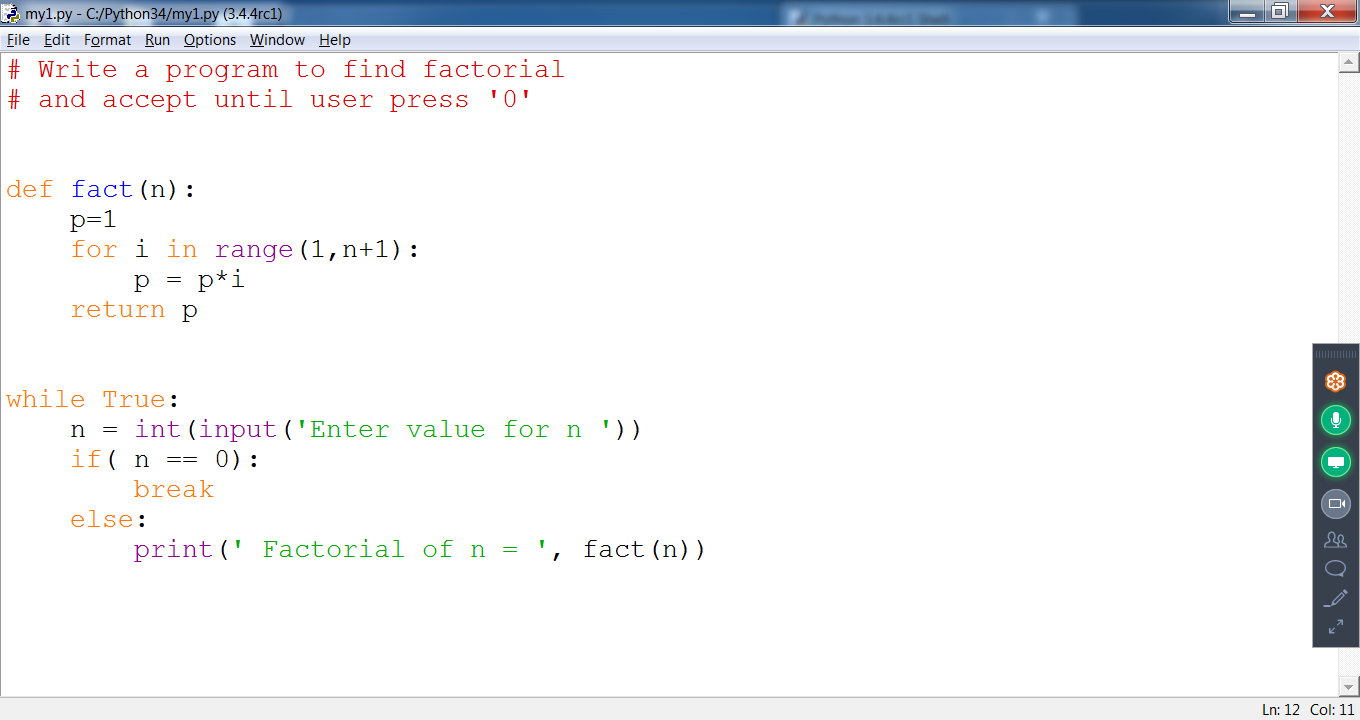
**n = int(input('Enter value for n '))**

**if( n == 0):**

**break**

**for i in range(1,n+1,2):**

**print(i)**

****

**# Write a program to find factorial**

**# and accept until user press '0'**

**def fact(n):**

**p=1**

**for i in range(1,n+1):**

**p = p\*i**

**return p**

**while True:**

**n = int(input('Enter value for n '))**

**if( n == 0):**

**break**

**else:**

**print(' Factorial of n = ', fact(n))**

**STRING, LISTS, TUPLES, DICTIONARIES**

>>> a = 24

>>> b = 4.5

>>> st = 'python'

>>> type(a)

<class 'int'>

>>> type(b)

<class 'float'>

>>> type(st)

<class 'str'>

>>>

# **The List Data Type**

* A *list* is a value that contains **multiple values** in an ordered sequence.
* A list value looks like this: ['cat', 'bat', 'rat', 'elephant', 123, ‘456’]. Just as string values are typed with quote characters to mark where the string begins and ends.
* A list begins with an opening square bracket and ends with a closing square bracket, [ ]. Values inside the list are also called *items*. Items are separated with commas (that is, they are *comma-delimited*).
* The value [] is an empty list that contains no values, as the empty string.

## **Negative Indexes**

* While indexes start at 0 and go up, you can also use negative integers for the index.
* The integer value **-1** refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on

>>> ls = ['java', 'python', 123, 'p123']

>>> print(ls)

['java', 'python', 123, 'p123']

>>> ls[0]

'java'

>>> ls[1]

'Python'

>>> ls[3]

'P123'

**>>> ls[-1]**

'p123'

**>>> ls[-4]**

'Java'

**>>> ls[6]**

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

ls[6]

IndexError: list index out of range

>>>

Lists can also contain other list values. The values in these lists of lists can be accessed using multiple indexes

>>> ls[0] = 'jeevan'

>>> ls[2]=['ramu', 'siva', 'ravan']

>>> ls

['jeevan', 'python', **['ramu', 'siva', 'ravan'],** 'p123']

>>>

>>> ls[2]

['ramu', 'siva', 'ravan']

**>>> ls[2][1]**

**'siva'**

>>>

## 

## 

>>> ls

['raju', ['python', 'java', 'php'], '123', 456, 'p@123\*&&\*']

>>> ls[-1]

'p@123\*&&\*'

>>> ls[-4]

['python', 'java', 'php']

>>> ls[-5]

'raju'

>>> ls[0]

'raju'

## 

## 

## **Getting Sublists with Slices**

* An index can get a single value from a list.
* A *slice* can get **several values from a list**, in the form of a **new list**
* A slice is typed between square brackets, like an index, but it has two integers separated by a colon
* a[2] is a list with an index (one integer).
* a[1:4] is a list with a slice (two integers)
* In a slice, the first integer is the index where the slice starts. The second integer is the index where the slice ends.
* A slice goes up to, **but will not include**, the value at the second index.
* A slice evaluates to a **new list value**

>>> ls

['raju', ['python', 'java', 'php'], '123', 456, 'p@123\*&&\*']

**>>> ls1 = ls[1:4] # 1,2,3, 4 not included**

>>> ls1

[['python', 'java', 'php'], '123', 456]

>>>

>>> ls[**:4**] **# 0 to 4 means 0,1,2,3**

['raju', ['python', 'java', 'php'], '123', 456]

>>> ls[**2:**] # 2 to **last element**

['123', 456, 'p@123\*&&\*']

>>>

>>> ls= ['raju', ['python', 'java', 'php'], '123', 456, 'p@123\*&&\*']

>>> ls[1]

['python', 'java', 'php']

>>> ls[1][1]

'Java'

>>> ls[2][2]

'3'

**>>> ls[1][1][2]**

'v'

>>>

* Can leave out one or both of the indexes on either side of the colon in the slice.
* Leaving out the first index is the same as **using 0**, or the beginning of the list.
* Leaving out the second index is the same as using the **length of the list**, which will slice to the end of the list

**Getting a List’s Length with len()**

The len() function will return the number of values that are in a list value passed to it, just like count the number of characters in a string value

>>> ls

['raju', ['python', 'java', 'php'], '123', 456, 'p@123\*&&\*']

>>>

>>> len(ls)

5

>>>

## **Changing Values in a List with Indexes**

Use an index of a list to change the value at that index.

>>> a[1] = 'General Purpose Language'

>>> a

['python', 'General Purpose Language', 'web', [10, 20, 30, 40], 'First\*']

>>>

## 

## 

## **List Concatenation and List Replication**

* The + operator can combine two lists to create a new list value in the same way it **combines two strings into a new string value**.
* The \* operator can also be used with a list and an integer value **to replicate** the list

**>>> [1,2,3] + ['a', 'b','c'] # List Concatenation**

[1, 2, 3, 'a', 'b', 'c']

>>> ['a','b','c'] **\***3 **# List Replica**

['a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c']

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

>>> a = a+['x','y','z']

>>> a

[1, 2, 3, 'x', 'y', 'z']

>>> lt = [1,2,3] + ['a','b','c']

>>> lt

[1, 2, 3, 'a', 'b', 'c']

>>> lt[3]\*5

'aaaaa'

>>>

## **Removing Values from Lists with del Statements**

## **Removing Values from Lists with remove()**

* The remove() method is passed the value to be removed from the list it is called on.
* If the value appears **multiple times** in the list, only the first instance of the value will be removed

>>> ls

['raju', ['python', 'java', 'php'], '123', 456, 'p@123\*&&\*']

**>>> ls.remove('123')**

>>> ls

['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*']

>>> ls.remove(1)

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

ls.remove(1)

ValueError: list.remove(x): x not in list

>>>

* The del statement will **delete values at an index** in a list.
* All of the values in the list after the deleted value will be **moved up one index**.

>>> ls

['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*']

>>> **del ls[1]**

>>> ls

['raju', 456, 'p@123\*&&\*']

>>>

>>> ls

['raju', 456, 'p@123\*&&\*']

>>> del ls[1:2] # Deletes element 1 , 2 is not included

>>> ls

['raju', 'p@123\*&&\*']

>>>

>>**> del lt[1:3]**

>>> lt

[1, 'b', 'c']

>>> **del lt**

>>> lt

Traceback (most recent call last):

File "<pyshell#33>", line 1, in <module>

lt

NameError: name 'lt' is not defined

>>>

>>> for i in range(4):

print(i)

0

1

2

3

>>> :

print(i)

0

1

2

3

4

>>>

>>> ls = ['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*']

**>>> for l in ls:**

print(l)

raju

['python', 'java', 'php']

456

p@123\*&&\*

>>>

## 

## **# Program to print values 1 to n with increment of 2 until user press '0'**

## 

## **while True:**

## **n = int(input('Enter value for n '))**

## **if(n == 0):**

## **break**

## **for i in range(1, n+1, 2):**

## **print(i)**

## **The in and not in Operators**

To determine whether a value is or isn’t in a list with the in and not in operators

>>> 'python' **in** ['c', 'java','.net','php']

False

>>> a = ['c', 'java','.net','php']

>>> 'java' in a

True

>>>

>>> ls

['raju', **['python', 'java', 'php'],** 456, 'p@123\*&&\*']

>>> 'python' in ls

False

>>> 'raju' in ls

True

>>> 'raju' not in ls

**False**

>>>

## 

# Write a Program input name until user press enter key

# and store onto list

ls=[]

while True:

name = input('Enter Name please ... ')

if(name == '' or name == 'bye'):

break

if( name not in ls):

ls.append(name)

print(ls)

# 

# 

# **Methods for LIST Type**

* A *method* is the same thing as a function, except it is “called on” a value
* Each data type has its own set of methods.
* The list data type, for example, has several useful methods for finding, adding, removing, and otherwise manipulating values in a list.

## **Finding a Value in a List with the index() Method**

* List values have an index() method that can be passed a value, and if that value exists in the list, the index of the value is returned.
* If the value isn’t in the list, then Python produces a ValueError error
* When there are duplicates of the value in the list, the index of its first appearance is returned

>>> ls

['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*']

**>>> ls.index('raju')**

**0**

>>> ls.index('python')

Traceback (most recent call last):

File "<pyshell#21>", line 1, in <module>

ls.index('python')

ValueError: 'python' is not in list

>>> ls.index(456)

2

>>>

## **Adding Values to Lists with the append() and insert() Methods**

* To add new values to a list, use the append() and insert() methods.
* **append()** method call adds the argument to the **end of the list**.
* The **insert()** method can insert a value at any index in the list. The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted.

>>> ls

['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*', 'c++']

>>> name = input('Enter name ')

Enter name kiran kumar

**>>> ls.append(name)**

>>> ls

['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*', 'c++', **'kiran kumar']**

>>>

**>>> ls**

**['raju', ['python', 'java', 'php'], 456, 'p@123\*&&\*', 'c++', 'kiran kumar']**

**>>> ls.insert(1,'django')**

**>>> ls**

**['raju', 'django', ['python', 'java', 'php'], 456, 'p@123\*&&\*', 'c++', 'kiran kumar']**

**>>>**

* The **return value of append() and insert()** is None
* Definitely wouldn’t want to store this as the new variable value, Rather, **the list is modified *in place***
* The append() and insert() methods are list methods and can be called only on list values, n**ot on other values such as strings or integers**

remove(), del, index, append(), insert()

## **Sorting the Values in a List with the sort() Method**

* Lists of number values or lists of strings can be sorted with the **sort()** method
* pass **True for the reverse keyword** argument to have sort() sort the values in reverse order
* Cannot sort lists that have **both number values *and* string** values
* Don’t try to capture the return value of sort()

>>> ls

['raju', 'django', **456,** 'p@123\*&&\*', 'c++', 'kiran kumar']

>>> ls.sort()

Traceback (most recent call last):

File "<pyshell#38>", line 1, in <module>

ls.sort()

TypeError: **unorderable types: int() < str()**

>>> ls[2] = '456'

>>> ls

['raju', 'django', '456', 'p@123\*&&\*', 'c++', 'kiran kumar']

>>> 456 + 456

912

>>> '456' + '456'

'456456'

>>>

>>> ls

['raju', 'django', '456', 'p@123\*&&\*', 'c++', 'kiran kumar']

>>> ls.sort()

>>> ls

['456', 'c++', 'django', 'kiran kumar', 'p@123\*&&\*', 'raju']

>>>

**Program :: Accept Names Until User press Enter Key or bye**

# Write a Program input name until user press enter key

# and store onto list

ls=[]

while True:

name = input('Enter Name please ... ')

if(name == '' or name == 'bye'):

break

if( name not in ls):

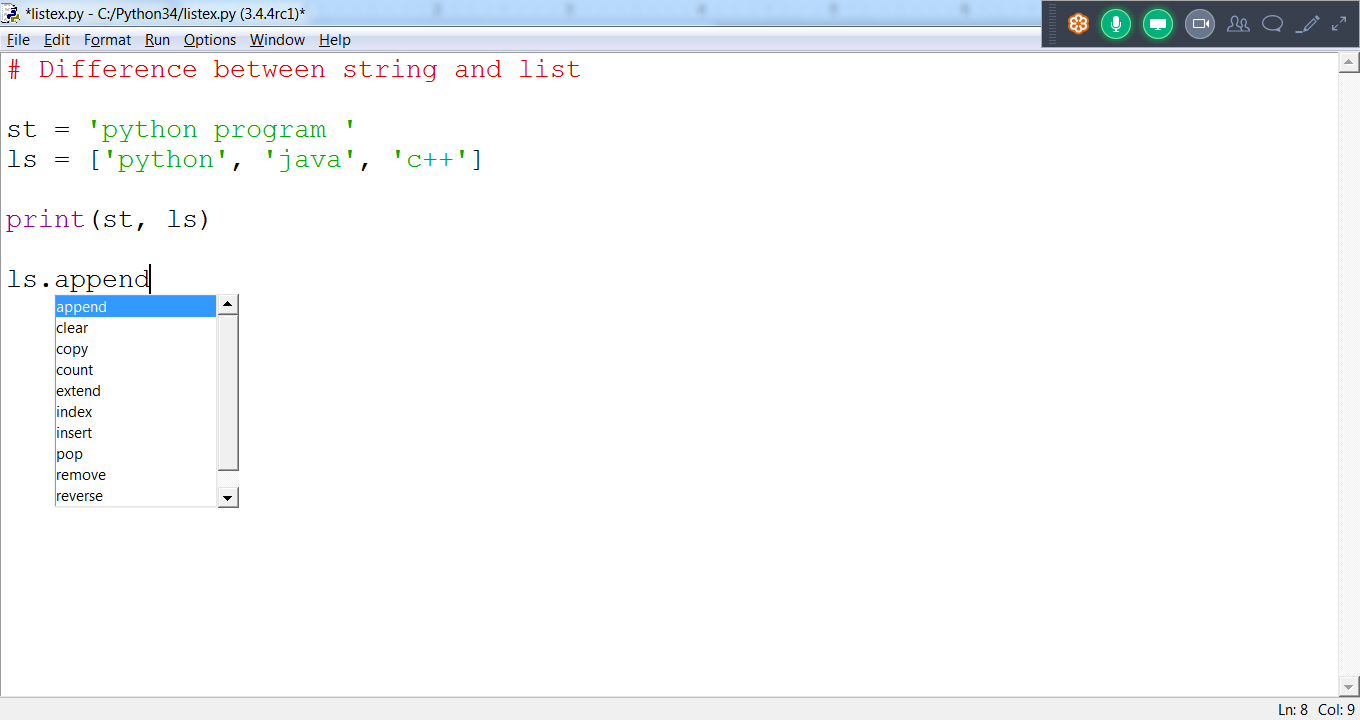
ls.append(name)

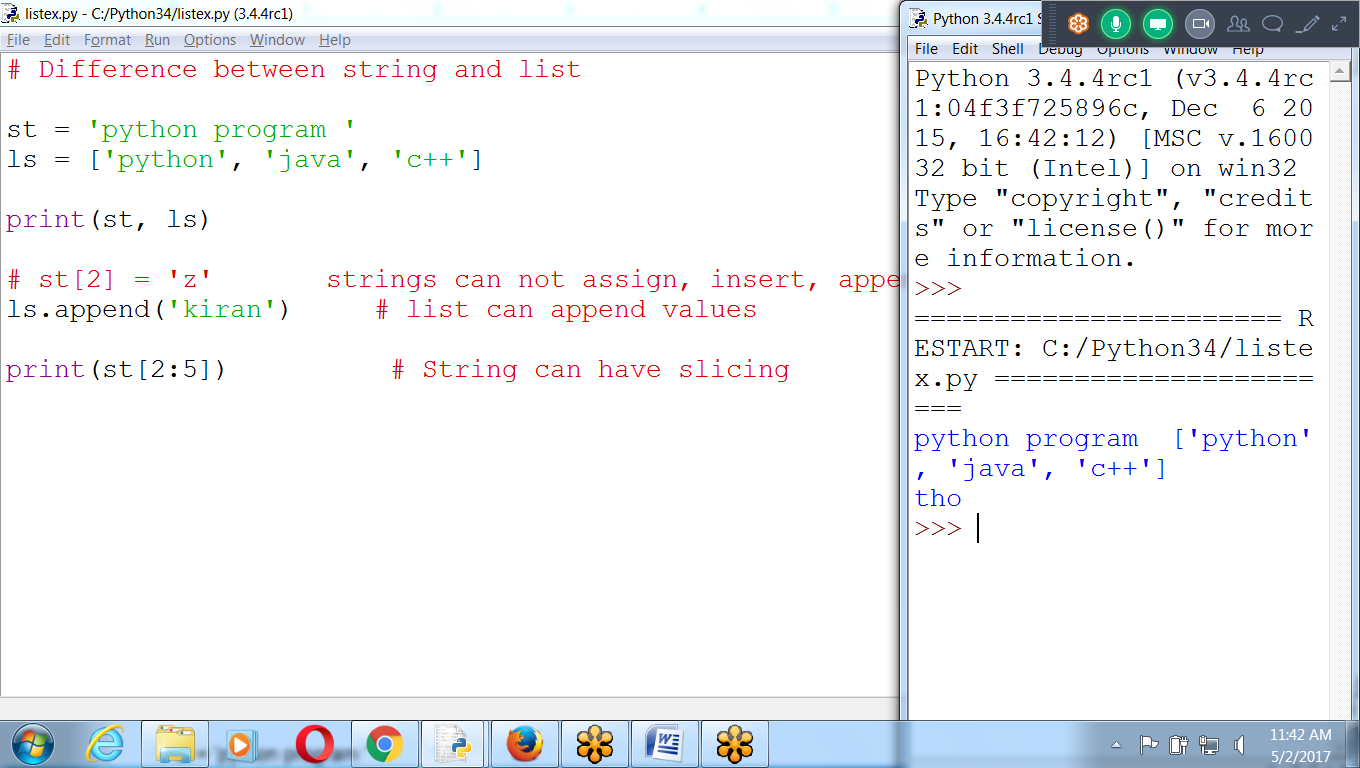
print(ls)

# **List-like Types: Strings**

* Strings and lists **are similar,** string to be a “list” of single text characters.
* We can do with lists can also be done with strings: **indexing; slicing;** and using them with for loops, with len(), and with the in and not in operators

List can be appended, delete





**# String methods**

st = 'python program '

print(' st = ',st)

print('st[0] = ', st[0])

print('st[2:5] = ', st[2:5])

print('g' in st)

print(' length of st = ', len(st))

for ch in st:

print(ch)

## **The Tuple Data Type**

* The *tuple* data type is almost identical to the list data type
* tuples are typed with **parentheses** and tuples are immutable
* Tuples cannot modified, appended, or removed.
* Don’t intend for that **sequence of values to change, contents don’t change**
* code using tuples **slightly faster** than code using lists.

>>> st = 'python program'

>>> ls = ['ramu', 'siva','sita']

>>> tp =**('ramu', 'siva','sita')**

>>> print(st,ls,tp)

python program ['ramu', 'siva', 'sita'] ('ramu', 'siva', 'sita')

>>>

>>> st = 'python program'

>>> ls = ['ramu', 'siva','sita']

>>> tp =('ramu', 'siva','sita')

>>> print(st,ls,tp)

python program ['ramu', 'siva', 'sita'] ('ramu', 'siva', 'sita')

>>> ls[2] = 'ravan'

>>> ls

['ramu', 'siva', 'ravan']

**>>> tp[2] = 'hello'**

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

tp[2] = 'hello'

TypeError: 'tuple' object does not support item assignment

>>> st[2] = 'k'

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

st[2] = 'k'

TypeError: 'str' object does not support item assignment

>>>

**Strings and Tuples are IMMUTABLE Object, can’t Modify**

**List are MUTABLE object, can insert, delete (can modify structure)**

# **The Dictionary Data Type in Python**

* Like a list, a dictionary is a collection of many values.
* Unlike indexes for lists, indexes for dictionaries can use many different data types, not just integers.
* Indexes for dictionaries are called **keys**, and a key with its associated value is called a key-value pair
* a dictionary is typed with braces, {}
* Dictionaries can still use integer values as keys, just like lists use integers for indexes, but they do not have to start at 0 and can be any number.

## **Dictionaries vs. Lists**

* Dictionaries items are unordered
* Order the **key-value pairs** are typed in a dictionary does not matter.
* Dictionaries are not ordered, they can’t be sliced like lists.

## **The keys(), values(), and items() Methods**

* Three dictionary methods that will return list-like values of the dictionary’s keys, values, or both keys and values: keys(), values(), and items()
* The values returned by these methods are not true lists
* They cannot be modified and do not have an append() method.
* can also use the multiple assignment trick in a for loop to assign the key and value to separate variables

>>> dic ={123:'raju', 78:'python', 'abc':123, 'a':['ravi','ramu']}

>>> dic

{'a': ['ravi', 'ramu'], 123: 'raju', 'abc': 123, 78: 'python'}

>>> for k in dic.keys():

print(k)

a

123

abc

78

>>> dic= {123 : 'python', 'l2':'java', 'l3':456}

>>> dic.keys()

dict\_keys(['l2', 123, 'l3'])

>>> dic.values()

dict\_values(['java', 'python', 456])

>>>

dic = {'l1': 'python', 'l2':'java', 'l3':'c lang'}

print(dic)

print(' ONly Keys are..')

for k in dic.keys():

print(k)

print(' Items are..')

for i in dic.items():

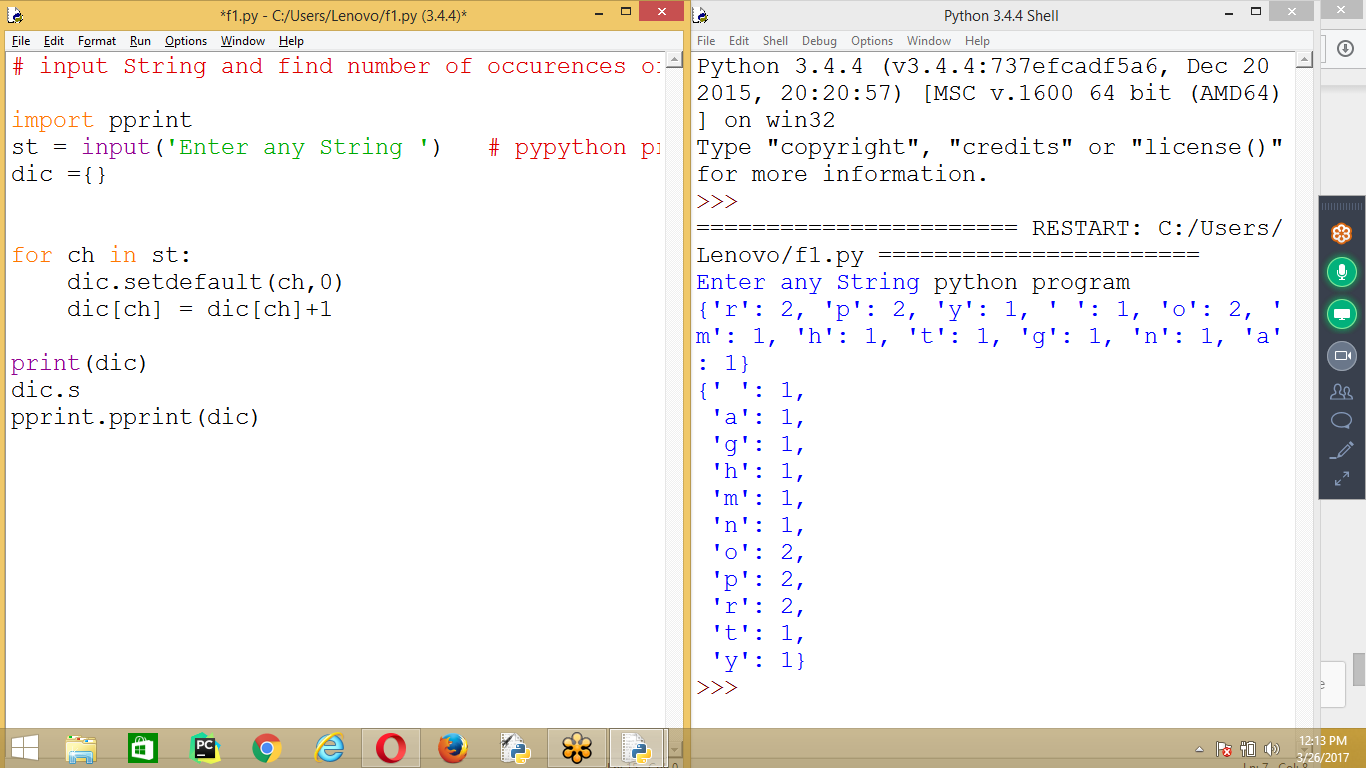
print(i)

print(' Values are..')

for v in dic.values():

print(v)

program that counts the number of occurrences of each letter in a string.



# input String and find number of occurrences of each character in a string

# in Sorting Order

import pprint

st = input('Enter any String ') # pypython progam

dic ={}

for ch in st:

dic.setdefault(ch,0)

dic[ch] = dic[ch]+1

print(dic)

pprint.pprint(dic)

**# Prog 3 Two Dimensional List**

ls = [

['ramu','cloud',1],

['siva','erp',2],

['raju','bigdata',3] ]

uid = input('Enter Username ')

pwd = input('Enter password ')

eid = int(input('Enter Employee id '))

**if [uid,pwd,eid] in ls:**

print('Grant Access to ', uid)

else:

print('can NOT Access ')

**Interchange of 4 values**

**>>> a=10**

**>>> b=20**

**>>> c=30**

**>>> d=40**

**>>> print(a,b,c,d)**

**10 20 30 40**

**>>> a,b,c,d = d,c,b,a # Interchange values**

**>>> print(a,b,c,d)**

**40 30 20 10**

**>>>**

EASY Learning

OOP

OPen Source

35 Hours : 45 Days

After Python:

Web Developer :: Django

AWS, Devops :: AUtomation Tools

Bigdata, Data science : Data scientist

Tableau, Networking, Testing

High Demand Required Skill Resource

GOOGLE, Facebook, youtube, dropbox, NASA

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 

**Using Given String check for Phone Number**

# Importing External Modules (re) for Regular Expressions

import re

str = input('Enter any string ')

reobj = re.compile('\d\d\d-\d\d\d-\d\d\d\d')

res = reobj.search(str)

print(res)

print(' Number Existing ', res.group())

# 

**Python Basics : 11.30 to 12.30 am**

**1 Installation, Expression, Data types, Variables,**

**Execution and Operators**

**2 Flow Controls (If, if-else) and Loops for, while, Range( Examples)**

**3 Functions and Collections( List, tuple, Dictionary)**

**PYTHON (30 Hours) : 9.30 to 10.30 am from Thursday**

**R Programing: 10.30 to 11.30 am**

**Python : Strings**

**Python : Regular Expressions**

**Python : Files**

**Python: Debugging, Exceptions**

**Python : Generators and Decorators**

**Python : Functions**

**Python : Collections (List, set , Dictionary)**

**python : OOPS**

**Advance Python ( 30 Hours)**

**Web, GUI, Database, Data Structures**

**Gaming, Multithreading, Numpy, scipy**

**Networking, Email, Scraping**

**Jupyter, Pycharm (EDITORS)**

**Python Developer :**

**Python (30 H) +**

**Advanced Python (30) + Python Projects**

**dJango (30 Hours) + Web Projects**

**Other Technologies :**

**Python (30 Hours) +**

**Data Science or AWS or**

**Devops or Big data or**

**Selenium or Open stack**

**Data Science : 3 Months (250 Hours)**

**Mon to Fri : 9.30 to 11.30**

**sat and sun :**

**Python + R Lang + Machine Learning ( 2 months)**

**Deep Learning + Natural Language Processing +**

**Predictive Analysis + Artificial Intelligence ( 2 Months)**