

# PYTHON APPLICATION PROGRAMMING

17CS664



IA MARKS: 40  
EXAM MARKS : 60  
CREDITS: 03

By.  
Ravi Kumar B N  
Assistant Professor, Dept. of CSE



**EASY TO READ AND SIMPLE TO IMPLEMENT.**

# WHAT IS PYTHON...?

- Python is an interpreted, high-level, general-purpose(designed to be used for writing software in the widest variety of application domains) programming language.

# FEATURES

High level language

Expressive – More readable

Interpreted Language

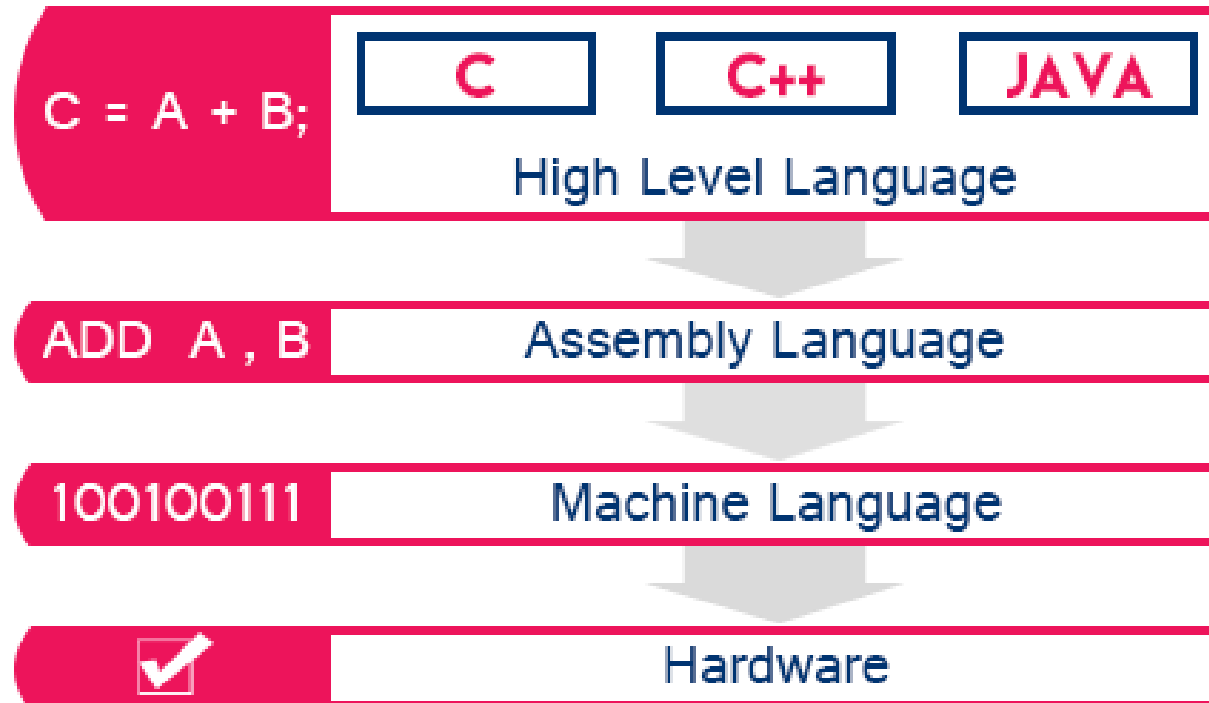
Cross-platform/ Portable

Large Standard Library

Everything is object in python

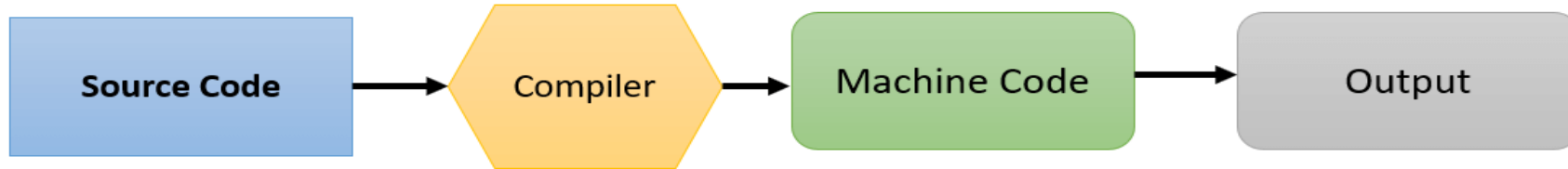
open source

# HIGH LEVEL LANGUAGE



# COMPILER V/S INTERPRETER

## How Compiler Works



## How Interpreter Works



- Interpreted code is translated to machine instructions step by step while the program is being executed.
- Compiled code has been translated before program execution.

# PORTABLE

- Python runs virtually every major platform used today.
- As long as you have a compatible python interpreter installed, python programs will run in exactly the same manner, irrespective of platform.

# IT'S POWERFUL

- Dynamic typing – type of a variable is interpreted at run time.
- Built-in types and tools
- Library utilities
- Third party utilities (e.g. NumPy, SciPy)
- Automatic memory management



Numpy- library, support for large, multidimensional arrays and matrices, along with a large collection of high level mathematical functions to operate on the arrays.

Scipy- Contains modules for optimization, linear algebra signals, image processing.

# PYTHON IS OBJECT-ORIENTED

Python is an object-oriented programming language. It allows us to develop applications using an object oriented approach. In python, we can easily create and use classes and objects.

Everything is object in python.

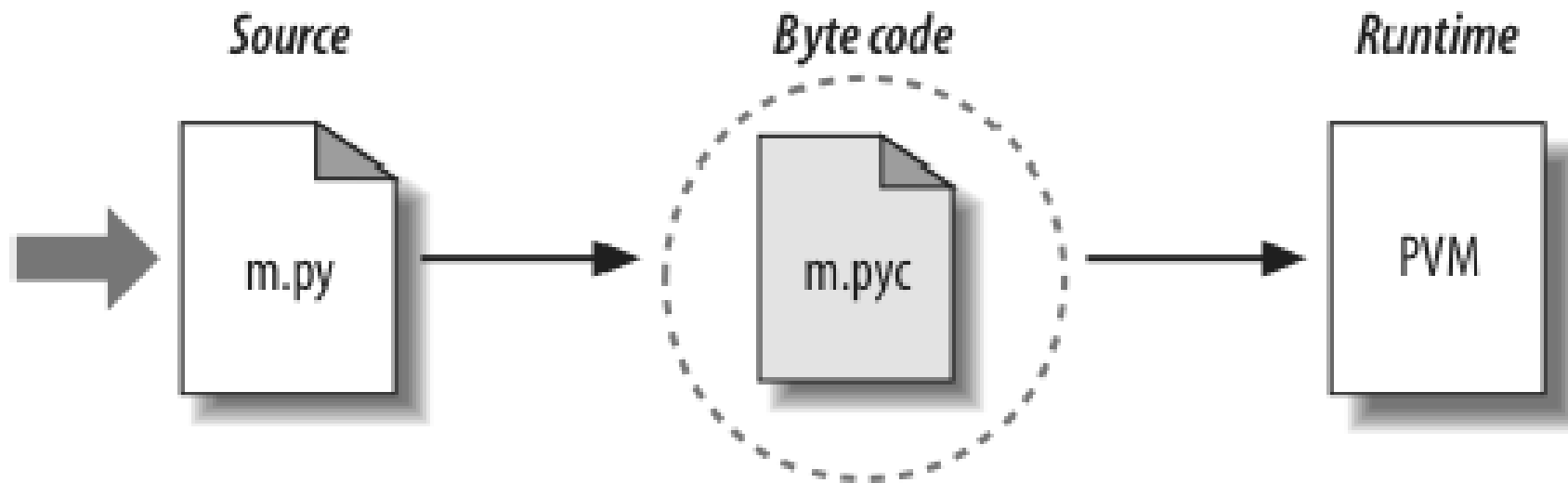
structure supports such concepts - polymorphism, operation overloading, and inheritance.

## **OPEN SOURCE (IT'S FREE )**

Downloading and installing python is free and easy.

source code is easily accessible.

# HOW PYTHON RUNS?



# IDLE

IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python.

The Python installer for Windows contains the IDLE module by default.

IDLE can be used to execute a single statement just like Python Shell and also to create, modify and execute Python scripts.

# DIFFERENT IDE AVAILABLE

- PyCharm
- Spyder
- Pydev
- IDLE
- Wing
- Eric Python
- Rodeo
- Thonny
- Jupyter Notebook
- Visual Studio



# WHY DO PEOPLE USE PYTHON...?

## *Fastest growing Language*

- In terms of number of developers using, number of libraries we have, number of companies using and number of areas we can implement it.
- Python's syntax is easy to learn, so both non-programmers and programmers can start programming right away.
- Python looks more like a readable

# WHAT CAN I DO WITH PYTHON...?

- System programming
- Graphical User Interface Programming
- Internet Scripting
- Component Integration
- Database Programming
- Gaming, Images, XML , Robot and more



# WHY OTHER BRANCHES HAS TO STUDY?

**ECE** - Python can be used for signal processing tasks using NumPy and SciPy. To implement digital signal processing algorithms

**EEE**- controlling and automating test equipment. Python was originally created for text parsing so it's amazingly useful to sift through huge amounts of text data to extract useful information - digital oscilloscope-It can output waveforms as raw CSV data.

**ME**- Mechanical and automobile industries use python to automate tasks. To write scripts and then import them to a CFD software to test numerous designs. To perform numerical analysis.

**CIV**- the applications of data science in civil engineering: Population forecasting for urban planning, water supply & sewerage system. Risk assessment and mitigation such as prediction of floods, earthquakes, cyclones.

# WHO USES PYTHON TODAY...

Python is being applied in real revenue-generating products by real companies.  
For instance:

- Google makes extensive use of Python in its web search system.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.
- ESRI uses Python as an end-user customization tool for its popular GIS mapping products.
- The YouTube video sharing service is largely written in Python

# HISTORY

- Invented in the Netherlands, early 90s by **Guido van Rossum**
- Python was conceived in the late 1980s and its implementation was started in December 1989
- Guido Van Rossum is fan of '**Monty Python's Flying Circus**', this is a famous TV show in Netherlands
- Named after Monty Python
- Open sourced from the beginning



# TEXT BOOKS

1. “Python for Everybody: Exploring Data Using Python 3” Charles R. Severance
2. “Think Python: How to Think Like a Computer Scientist” Allen B. Downey

# **MODULE 2 – PART 3**

## **FILES**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# INTRODUCTION

- ✓ File is a named location on the system storage which records data for later access.  
It enables persistent storage in a non-volatile memory i.e. Hard disk.
- ✓ It is required to work with files for either writing to a file or read data from it. It is essential to store the files permanently in secondary storage.
- ✓ In python, file processing takes place in the following order.
  - Open a file that returns a file handle.
  - Use the handle to perform read or write action.
  - Close the file handle.

# TYPES OF FILES & OPERATIONS ON FILES

## Types of Files

1. Text Files - All doc files /excel files etc
2. Binary Files - Audio files, Video Files, images etc

## Operations on File:

- Open
- Close
- Read
- Write

# OPEN A FILE IN PYTHON

✓ To read or write to a file, you need to open it first. To open a file in Python, use its built-in `open()` function. This function returns a file object, i.e., a handle. You can use it to read or modify the file.

✓ **`open()` file method :**

**`file_object = open("file_name", "access_mode")`**

**`file_object`** – File handler that points to the particular location as a reference to an object

**`file_name`**- Name of the file

**`access_mode`**- Read/write/ append mode. By default, it is set to read-only <r>.

Ex: `file1 = open("app.log", "w")`



- ✓ Python stores a file in the form of bytes on the disk, so you need to decode them in strings before reading. And, similarly, encode them while writing texts to the file. This is done automatically by Python Interpreter.
- ✓ If the open is successful, the operating system returns us a file handle. The file handle is not the actual data contained in the file, but instead it is a “handle” that we can use to read the data. You are given a handle if the requested file exists and you have the proper permissions to read the file.

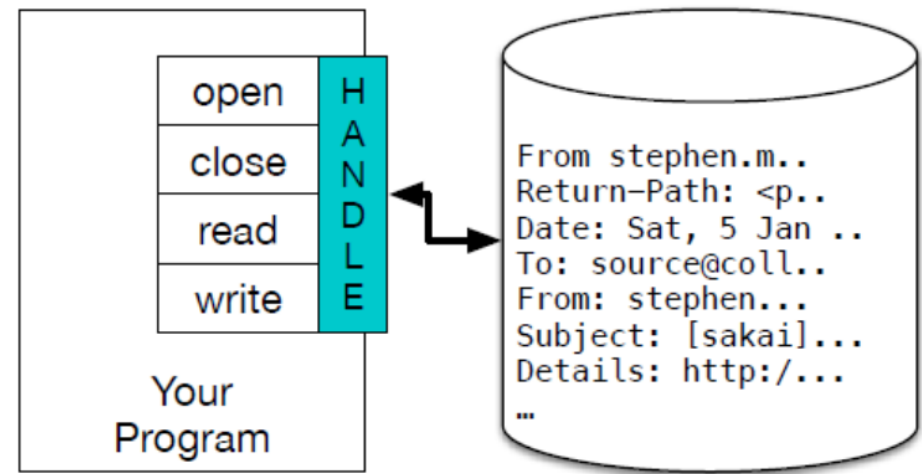


Figure 7.2: A File Handle

- ✓ **If the file does not exist, open will fail with a traceback and you will not get a handle to access the contents**

```
• >>> file1 = open('stuff.txt')  
• Traceback (most recent call last):  
• File "<stdin>", line 1, in <module>  
• FileNotFoundError: [Errno 2] No such file or directory
```

- ✓ **A file opening may cause an error due to some of the reasons as listed below**
  - File may not exist in the specified path (when we try to read a file)
  - File may exist, but we may not have a permission to read/write a file
  - File might have got corrupted and may not be in an opening state

# FILE MODES

Modes	Description
<b>r</b>	Opens a file only for reading
<b>rb</b>	Opens a file only for reading but in a binary format
<b>w</b>	Opens a file only for writing; overwrites the file if the file exists
<b>wb</b>	Opens a file only for writing but in a binary format
<b>a</b>	Opens a file for appending. It does not overwrite the file, just adds the data in the file, and if file is not created, then it creates a new file
<b>ab</b>	Opens a file for appending in a binary format
<b>r+</b>	Opens a file only for reading and writing
<b>w+</b>	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
<b>a+</b>	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode.

# TEXT FILES & LINES

A text file can be thought of as a sequence of lines, much like a Python string can be thought of as a sequence of characters. For example, this is a sample of a text file which records mail activity from various individuals in an open source project development team:

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakail] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

To break the file into lines, there is a special character that represents the “end of the line” called the newline character. In the text file each line can be separated using escape character `\n`

```
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
```

# READING THE FILES

When we successfully open a file to read the data from it, the `open()` function returns the file handle (or an object reference to file object) which will be pointing to the first character in the file.

**There are different ways in which we can read the files in python.**

- `read()` //reads all the content
- `read(n)` // reads only the first n characters
- `readline()` // reads single line
- `readlines()` //reads all lines

Where , n is the number of bytes to be read

Note: If the file is too large to fit in main memory, you should write your program to read the file in chunks using a for or while loop.

# SAMPLE PROGRAMS

## readexample.py

```
f1=open("1.txt","r")

# to read first n bytes
print("---first four
characters---")
print(f1.read(4))

# to read a first line
print("---first line---")
print(f1.readline())

# to read entire file
print("---Entire File---")
print(f1.read())
```

### Output

```
---first four characters---
Coro
---first line---
navirus Can Be Stopped Only by
Harsh Steps

---Entire File---
Stay at home
wear a facemask
Clean your hands often
Monitor your symptoms
```

## countlines.py

```
f1 = open("1.txt","r")
count = 0
for line in f1:
    count = count + 1
print('Line Count:', count)
```

### Output

Line Count: 5

**Note:** When the file is read using a for loop in this manner, Python takes care of splitting the data in the file into separate lines using the newline character.

## Readlineexample.py

```
f1=open("1.txt") #by default read mode
#to read line wise
print(f1.readline())
print(f1.readline())
```

### Output

```
Coronavirus Can Be Stopped Only by Harsh Steps
Stay at home
```

## Countchars.py

*#finds the length of the file*

```
f1 = open('1.txt')
ch = f1.read()
print(len(ch))
```

### Output

120

**Note:**In the above code it counts the number of characters along with newline character(\n)

## 1.txt

```
Coronavirus Can Be Stopped Only by Harsh
Steps
Stay at home
wear a facemask
Clean your hands often
Monitor your symptoms
```

# WRITING THE FILES:

- ✓ To write a data into a file, we need to use the mode 'w' in open( ) function.
- ✓ The write( ) method is used to write data into a file.

```
>>> fhand=open("mynewfile.txt","w")  
>>> print(fhand)
```

```
<_io.TextIOWrapper name='mynewfile.txt' mode='w' encoding='cp1252'>
```

- ✓ We have two methods for writing data into a file as shown below

1. write(string)
2. writelines(list)

If the file specified already exists, then the old contents will be erased and it will be ready to write new data into it. If the file does not exist, then a new file with the given name will be created.

**write( ) method:** It returns number of characters successfully written into a file. The file object also keeps track of position in the file.

For example,

**Writexample.py**

```
fhand=open("2.txt",'w')  
s="hello how are you?"  
print(fhand.write(s))
```

**Output:**

18

**writelines( ) method:** Example: This code adds the list of contents into the file including \n

**Writelist.py**

```
food = ["Citrus\n", "Garlic\n", "Almond\n", "Ginger\n"]  
my_file = open("immunity.txt", "w")  
my_file.writelines(food)
```

**Output**

```
It creates a file immunity.txt  
Citrus  
Garlic  
Almond  
Ginger
```



# Example for read binary 'rb' and write binary 'wb'

## Imagecopy.py

```
f1=open("bird.jpg", 'rb')  
f2=open("birdcopy.jpg", 'wb')  
for i in f1:  
    print(f2.write(i))
```

bird.jpg #input file



birdcopy.jpg #output file



# SEARCHING THROUGH A FILE

- ✓ When you are searching through data in a file, it is a very common pattern to read through a file, ignoring most of the lines and only processing lines which meet a particular condition.
- ✓ Most of the times, we would like to read a file to search for some specific data within it. This can be achieved by using some string methods while reading a file.
- ✓ For example, we may be interested in printing only the line which starts with a specific character.

# SEARCH EXAMPLE

## Search1.py

```
fhand = open('mbox.txt')
for line in fhand:
    line = line.rstrip() #strips whitespace from right side of a string
    if line.startswith('From:'):
        print(line)
```

## Or

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    # Skip 'uninteresting lines'
    if not line.startswith('From:'):
        continue
    # Process our 'interesting' line
    print(line)
```

## Output

```
From: stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
From: louis@media.berkeley.edu
From: zqian@umich.edu Fri Jan 4 16:10:39 2008
```

## Search2.py

Note: find lines where the search string is anywhere in the line. Find() method returns either position of a string or -1

```
fhand = open('mbox.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1: continue
    print(line)
```

## Output

```
From: stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

## mbox.txt

```
From: stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
From: louis@media.berkeley.edu
Subject: [sakai] svn commit:
From: zqian@umich.edu Fri Jan 4 16:10:39 2008
Return-Path: <postmaster@collab.sakaiproject.org>
```

# LETTING THE USER CHOOSE THE FILE NAME

In a real time programming, it is always better to ask the user to enter a name of the file which he/she would like to open, instead of hard-coding the name of a file inside the program.

## Fileuser.py

```
fname=input("Enter a file name:")
f1=open(fname)
count =0
for line in f1:
    count+=1
    print("Line Number ",count, ":", line)
print("Total lines=",count)
f1.close()
```

## Output:

```
Enter a file name:1.txt
Line Number  1 : Coronavirus Can Be Stopped Only by Harsh Steps
Line Number  2 : Stay at home
Line Number  3 : wear a facemask
Line Number  4 : Clean your hands often
Line Number  5 : Monitor your symptoms
Total lines= 5
```

In this program, the user input filename is received through variable fname, and the same has been used as an argument to open() method. Now, if the user input is 1.txt (discussed before), then the result would be **Total lines=5**

Everything goes well, if the user gives a proper file name as input. But, what if the input filename cannot be opened (Due to some reason like – file doesn't exists, file permission denied etc)?

Obviously, Python throws an error. The programmer need to handle such run- time errors as discussed in the next section.

# USING TRY ,EXCEPT AND OPEN

When you try opening the file which doesn't exist or if a file name is not valid, then the interpreter throws you an error. Assume that the open call might fail and add recovery code when the open fails as follows:

## Tryfile.py

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()
count = 0
for line in fhand:
    if line.startswith('From:'):
        count = count + 1
print('count=', count)
```

## Output1:

```
Enter the file name: mbox.txt
count= 3
```

## Output2:

```
Enter the file name: newmbox.txt
File cannot be opened: newmbox.txt
```

In the above program, the command to open a file is kept within try block. If the specified file cannot be opened due to any reason, then an error message is displayed saying File cannot be opened, and the program is terminated. If the file could able to open successfully, then we will proceed further to perform required task using that file.

# CLOSE A FILE IN PYTHON

- ✓ It's always the best practice to close a file when your work gets finished. However, Python runs a garbage collector to clean up the unused objects. While closing a file, the system frees up all resources allocated to it
- ✓ The most basic way is to call the Python `close()` method.

**Filepointer.close()**

Example:

```
f = open("app.log")  
# do file operations.  
f.close()
```

# PROBLEMS WITH WHITE SPACE- REPR()

- ✓ When we are reading and writing files, we might run into problems with white space. These errors can be hard to debug because spaces, tabs and newlines are normally invisible.

```
>>> s = '1 2 \t 3 \n 4'
>>> print(s)
1 2 3
4
```

- ✓ The built in function repr( ) can take any object as an argument and returns a string representation of the object. For strings, it represents whitespace, characters with backslash sequences:

```
>>> print(s)
'1 2 \t 3 \n 4'
```

- ✓ This helps for debugging

## EXERCISE PROBLEMS:

- 1) WAP to copy all the lines from one file to another file (file1.txt to file2.txt) where the line begins with vowels and also demonstrate the computational faults in the program
- 2) Write a program to count the number of occurrences of a given word(accept the input from user) in a file.  
(hint : Can use strip() and count() methods for each word in a line)
- 3) Input decimal number and convert it to binary number and write it in another file until user enters 0



**THANK  
YOU**

# **MODULE 3 – PART 2**

## **DICTIONARY**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# BASICS

- ✓ **Dictionary** : It is like a list, but In a list, the index positions have to be integers; in a dictionary, **the indices can be any type**.
- ✓ It is mapping between set of indices (which are called keys) and a set of values. Each key maps to a value. **The association of a key and a value is called a key-value pair or sometimes an item.**
- ✓ **dict( )**: The function dict creates a new dictionary with no items.

build a empty dictionary that maps from English to Spanish words,

```
>>> eng2sp = dict( )  
>>> print(eng2sp)  
{ }
```

The curly brackets, { }, represent an empty dictionary.

- ✓ To add items to the dictionary, you can use square brackets:

```
>>> eng2sp['one'] = 'uno'
```

This line creates an item that maps from the key 'one' to the value 'uno'.

- ✓ you can create a new dictionary with three items:  
`>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}`
- ✓ The **order of items in a dictionary is unpredictable** But that's not a problem because we use the keys to look up the corresponding values:

```
>>> print(eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
>>> print(eng2sp['two'])
'dos'
```

The key 'two' always maps to the value 'dos' so the order of the items doesn't matter.

- ✓ **len()**: The len function works on dictionaries; it returns the number of key-value pairs:  
`>>> len(eng2sp)`  
3
- ✓ The **in operator** works on dictionaries; it tells whether something appears as a key in the dictionary  
`>>> 'one' in eng2sp`  
True  
`>>> 'uno' in eng2sp`  
False

✓ To see whether something appears as a value in a dictionary, we can use the method `values`, which returns the values as a list, and then use the `in` operator:

```
>>> vals = eng2sp.values( )  
>>> 'uno' in vals  
True
```

Note: The `in` operator uses different search algorithms for lists and dictionaries.

For **lists**, it uses a **linear search algorithm**. As the list gets longer, the **search time gets longer in direct proportion to the length of the list**.

For **dictionaries**, Python uses an algorithm called a **hash table** that has a remarkable property—the `in` operator takes about the **same amount of time no matter how many items there are in a dictionary**

# DICTIONARY AS A SET OF COUNTERS

Given a string and we want to count how many times each letter appears.

You could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, you would add an item to the dictionary. After that you would increment the value of an existing item

The for loop traverses the string. Each time through the loop, if the character `c` is not in the dictionary, we create a new item with key `c` and the initial value 1 (since we have seen this letter once). If `c` is already in the dictionary we increment `d[c]`.

```
#count how many times each letter appears.
```

```
word = 'life is beautiful'
```

```
d = dict()
```

```
for c in word:
```

```
    if c not in d:
```

```
        d[c] = 1
```

```
    else:
```

```
        d[c] = d[c] + 1
```

```
print(d)
```

```
Output:
```

```
{'l': 2, 'i': 3, 'f': 2, 'e': 2, ' ': 2, 's': 1, 'b': 1, 'a': 1, 'u': 2, 't': 1}
```

**Note:** It computes a histogram, which is a statistical term for a set of counters (or frequencies).

- ✓ **get( )**: it takes a key and a default value. If the key appears in the dictionary, get() returns the corresponding value, otherwise it returns the default value.

For example: >>> counts = { 'ECE' : 29 , 'ISE' : 30, 'EEE': 18, 'ME': 3}

```
>>> print(counts.get('ECE', 0))
```

```
29
```

```
>>> print(counts.get('CSE', 0))
```

```
0
```

**#count how many times each letter appears using get method.**

```
word = 'life is beautiful'
```

```
d = dict()
```

```
for c in word:
```

```
    d[c] = d.get(c,0) + 1
```

```
print(d)
```

**Output:**

```
{ 'l': 2, 'i': 3, 'f': 2, 'e': 2, ' ': 2, 's': 1, 'b': 1, 'a': 1, 'u': 2, 't': 1 }
```

# BUILT IN FUNCTIONS

Function	Examples
<b>len():</b> Returns the number of items(key value pairs) in the dictionary	len(dict1)
<b>clear():</b> To remove all elements from the dictionary	d.clear()
<b>get():</b> To get the value associated with the key	d.get(key,defaultvalue) d.get('c', 0)
<b>pop():</b> It removes the entry associated with the specified key and returns the corresponding value	d.pop(key)
<b>keys():</b> It returns all keys associated with dictionary	d.keys()
<b>update():</b> All items present in the dictionary x will be added to dictionary d	d.update(x)
<b>del</b> operator	del d[key] del d



# DICTIONARIES AND FILES

```
#program to read through the lines of the file
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1
print(counts)
```

## Output:

Enter the file name: sample1.txt

```
{'When': 1, 'we': 2, 'run': 1, 'the': 2, 'program':
1, 'see': 1, 'a': 1, 'raw': 1, 'dump': 1, 'of': 2, 'all':
1, 'counts': 1, 'in': 1, 'unsorted': 1, 'hash': 1,
'order': 1}
```

## #sample1.txt

When we run the program, we see a raw dump of all of the counts in unsorted hash order

- ✓ We have two for loops. The outer loop is reading the lines of the file and the inner loop is iterating through each of the words on that particular line.

# LOOPING AND DICTIONARIES

- ✓ If we use a dictionary as the sequence in a **for statement**, it **traverses the keys of the dictionary**.

This loop prints each key and the corresponding value:

```
>>> counts = { 'ECE' : 29 , 'ISE' : 30, 'EEE': 18, 'ME': 3}  
>>> for key in counts:  
    print (key, counts[key])
```

ECE 29

ISE 30

EEE 18

ME 3

- ✓ If we wanted to find all the entries in a dictionary with a value above ten, we could write the following code:

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}  
>>> for key in counts:  
    if counts[key] > 10 :  
        print(key, counts[key])
```

jan 100

annie 42

# TO PRINT THE KEYS IN ALPHABETICAL ORDER

```
dict1= { 'chuck' : 1 , 'annie' : 42, 'jan': 100}  
lst = list(dict1.keys())  
print(lst)  
lst.sort()  
for key in lst:  
    print(key, counts[key])
```

## Output:

```
['chuck', 'annie', 'jan']  
annie 42  
chuck 1  
jan 100
```

# ADVANCED TEXT PARSING TRANSLATE() AND MAKETRANS()

While solving problems on semantic analysis, machine learning, data analysis etc., the punctuation marks like comma, full point, question mark etc. are also considered as a part of word and stored in the dictionary.

This type of analysis of words may lead to unexpected results. So, we need to be careful in parsing the text and we should try to eliminate punctuation marks, ignoring the case etc.

We will even let Python tell us the list of characters that it considers “punctuation”:

```
>>> import string
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

The **maketrans()** method returns a mapping table for translation usable for **translate()** method. This method creates a Unicode representation of each character for translation.

**line.translate(str.maketrans(fromstr, tostr, deletestr))**

The **translate()** method returns a string where each character is mapped to its corresponding character in the translation table.

**#code to demonstrate translations using maketrans() and translate**

```
# specify to translate chars
str1 = "wy"
```

```
# specify to replace with
str2 = "gf"
```

```
# delete chars
str3 = "u"
```

```
# target string
trg = "weeksyourweeks"
```

```
# using maketrans() to construct translate table
table = trg.maketrans(str1, str2, str3)
```

```
# Printing original string
print("The string before translating is : ", end="")
print(trg)
```

```
# using translate() to make translations.
print("The string after translating is : ", end="")
print(trg.translate(table))
```

**#output:**

The string before translating is : weeksyourweeks  
The string after translating is : geeksforgeeks

**# Code to delete all of the punctuation and to avoid treating “who” and “Who” as different words with different counts.**

```
import string
fname =input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()
counts = dict()
for line in fhand:
    line = line.translate(line.maketrans(' ', ' ', string.punctuation))
    # above line is an alternative that creates a dictionary mapping of every character from string.punctuation to None
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1
print(counts)
```

**#Output:**

Enter the file name: sample1.txt

{'when': 2, 'we': 1, 'run': 1, 'the': 2, 'program': 1, 'see': 1, 'a': 1, 'raw': 1, 'dump': 1, 'of': 2, 'all': 1, 'counts': 1, 'in': 1, 'unsorted': 1, 'hash': 1, 'order': 1}

**#sample1.txt**

When we run the program, when see a raw dump Of: all\$ of the counts in unsorted hash order

# DICTIONARIES & LISTS

Dictionary	Lists
A dictionary is a key:value pair	List is a collection of index value pair as that of array
It is created by placing elements in { } as “key”:”value”, which is separated by commas “ , ”	It is created by placing elements in [ ] separated by commas “ , ”
The elements are accessed via keys	The elements are accessed via indices.
Insertion order is not preserved	Insertion order is preserved
For dictionaries, “in” operator uses a hash table that has a remarkable property—the in operator takes about the same amount of time no matter how many items there are in a dictionary	For lists, “in” operator uses a linear search algorithm. As the list gets longer, the search time gets longer in direct proportion to the length of the list.

## Lists from Dictionaries

<pre>&gt;&gt;&gt; counts = { 'ECE' : 29 , 'ISE' : 30, 'EEE': 18} &gt;&gt;&gt; print(counts.items()) dict_items([('ECE', 29), ('ISE', 30), ('EEE', 18)]) &gt;&gt;&gt; print(counts.keys()) dict_keys(['ECE', 'ISE', 'EEE'])</pre>	<pre>&gt;&gt;&gt; print(counts.values()) dict_values([29, 30, 18])</pre>
--	--

# PREVIOUS YEAR QP QUESTIONS

- 1) List merits of dictionary over list. Write a python program to accept USN and Marks obtained. Find maximum, minimum and student USN who have scored in the range 100-85, 85-75, 75-60 and below 60 marks separately.
- 2) Write a program to input dictionary from the keyboard and print the sum of values?
- 3) Write a program to find number of occurrences of each letter present in the given string and display as a dictionary ?
- 4) Write a program to find number of occurrences of each vowel present in the given string and display as a dictionary ?
- 5) Write a python program that accepts a sentences and build dictionary with LETTERS, DIGITS, UPPERCASE, LOWERCASE as key values and their count in the sentences as values:

Ex: sentence=[VTU@123.e-learning](#)

D={"LETTERS":12,"DIGITS":3,"UPPERCASE":4,"LOWERCASE": 8}

**THANK  
YOU**



# **MODULE 3 – PART 1**

## **LISTS**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# BASICS

- ✓ **List:** It is a **sequence of values** and values can be of any type.  
The values in list are called elements or sometimes items.

There are several ways to create a new list; the simplest is to enclose the elements in square brackets ([ and ]):

Example:

**[10, 20, 30, 40]** List of integers

**['crunchy frog', 'ram bladder', 'lark vomit']** List of strings

- ✓ The elements of a list don't have to be the same type.
- ✓ **Nested List:** A list within another list is nested.  
The following list contains a string, a float, an integer, and another list:  
**['spam', 2.0, 5, [10, 20]]**
- ✓ **Empty List:** A list that contains no elements is called an empty list; you can create one with empty brackets, **[]**.

✓ **Assigning list values to variables:**

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
```

```
>>> numbers = [17, 123]
```

```
>>> empty = []
```

```
>>> print cheeses, numbers, empty
```

Output: ['Cheddar', 'Edam', 'Gouda'] [17, 123] []

✓ **The syntax for accessing the elements of a list** is the same as for accessing the characters of a string—the bracket operator.

The expression inside the brackets specifies the index. Indices always starts at 0

```
>>> print cheeses[0]
```

Output: Cheddar

✓ **List indices work the same way as string indices:**

- Any integer expression can be used as an index.
- If you try to read or write an element that does not exist, you get an Index Error.
- If an index has a negative value, it counts backward from the end of the list.

## Samples



```
>>>a = "bee"  
>>>print(list(a))  
Output: ['b', 'e', 'e']
```

```
>>>a = ("I", "am", "a", "tuple")  
>>>print(list(a))  
['I', 'am', 'a', 'tuple']
```

```
>>>my_list = ['p','r','o','b','e']
```

```
>>>print(my_list[0])
```

Output: p

```
>>>print(my_list[2])
```

Output: o

```
>>>print(my_list[4])
```

Output: e

```
>>>my_list[4.0]
```

Output: Error! Only integer can be used for indexing

### # Nested List

```
>>>n_list = ["Happy", [2,0,1,9]]
```

### # Nested indexing

```
>>>print(n_list[0][1])
```

Output: a

```
>>>print(n_list[1][3])
```

Output: 5

# LISTS ARE MUTABLE

- ✓ Unlike strings, lists are mutable because we can change the order of items in a list or reassign an item in a list.

When the bracket operator appears on the left side of an assignment, it identifies the element of the list that will be assigned.

```
>>> numbers = [17, 123]
```

```
>>> print(numbers)
```

Output: [17, 123]

```
>>> numbers[1] = 5
```

```
>>> print(numbers)
```

Output: [17, 5]

The one-eth element of numbers, which used to be 123, is now 5.

## We can use in operator on lists:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
```

```
>>> 'Edam' in cheeses
```

Output: True

```
>>> 'Brie' in cheeses
```

Output: False

## Traversing a list:

The most common way to traverse the elements of a list is with a for loop. The syntax is the same as for strings:

This works well if we only need to read the elements of the list

### Listra.py

```
# iterate over a list  
list = [1, 3, 5, 7, 9]
```

```
# Using for loop  
for i in list:  
    print(i)
```

### Output:

```
1  
3  
5  
7  
9
```

## ✓ If you want to write or update the elements, you need the indices.

A common way to do that is to combine the functions **range** and **len**:

### Listrav1.py

```
# code to iterate over a list
list = [1, 3, 5, 7, 9]

# getting length of list
length = len(list)

# Iterating the index
# same as 'for i in range(len(list))'

for i in range(length):
    list[i] = list[i]+5

print(list)
```

### Output:

```
[6, 8, 10, 12, 14]
```

This loop traverses the list and updates each element. len returns the number of elements in the list. range returns a list of indices from 0 to n-1, where n is the length of the list.

### Listlen.py

```
# code to find the length of nested list
list1=['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
print(len(list1))
```

### Output:

```
4
```

Although a list can contain another list, the nested list still counts as a single element. The length of this list is four:

### Emptylistra.py

```
empty=[]
for x in empty:
    print('It never executes')
```

### Output:

```
No output
```

A for loop over an empty list never executes the body:

# LIST OPERATIONS

**List Concatenation:** The + operator concatenates lists

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

**List Repetition:** Similarly, the \* operator repeats a list a given number of times

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The first example repeats [0] four times.

The second example repeats the list [1, 2, 3] three times.



# LIST SLICES

The slice operator also works on lists:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']  
>>> t[1:3]  
['b', 'c']
```

If you omit the first index, the slice starts at the beginning.

```
>>> t[:4]  
['a', 'b', 'c', 'd']
```

If you omit the second, the slice goes to the end.

```
>>> t[3:]  
['d', 'e', 'f']
```

So if you omit both, the slice is a copy of the whole list

```
>>> t[:]  
['a', 'b', 'c', 'd', 'e', 'f']
```

A slice operator on the left side of an assignment can update multiple elements:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']  
>>> t[1:3] = ['x', 'y']  
>>> print t  
['a', 'x', 'y', 'd', 'e', 'f']
```

# LIST METHODS

**append:** It adds a new element to the end of a list:

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
['a', 'b', 'c', 'd']
```

```
>>>t.append('d','e')
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    t.append('d','e')
TypeError: append() takes exactly one argument (2 given)
```

**extend:** It takes a list as an argument and appends all of the elements:

```
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print t1
['a', 'b', 'c', 'd', 'e']
```

This example leaves t2 unmodified.

**Sort:** It arranges the elements of the list from low to high:

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print t
['a', 'b', 'c', 'd', 'e']
```

Most list methods are void; they modify the list and return None.

# DELETING ELEMENTS

**pop()**- This method will be used If we know the index of the element to be deleted.

It modifies the list and returns the element that was removed.

If we don't provide an index, it deletes and returns the last element.

```
>>> t = ['a', 'b', 'c']
```

```
>>> x = t.pop(1)
```

```
>>> print t
```

```
['a', 'c']
```

```
>>> print x
```

```
b
```

**del**- If we don't need the removed value, you can use the del operator:

```
>>> t = ['a', 'b', 'c']
```

```
>>> del t[1]
```

```
>>> print t
```

```
['a', 'c']
```

To remove more than one element, you can use del with a slice index:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
```

```
>>> del t[1:5]
```

```
>>> print t
```

```
['a', 'f']
```

**remove()**- If we know the element we want to remove (but not the index), you can use remove:

```
>>> t = ['a', 'b', 'c']  
>>> t.remove('b')  
>>> print t  
['a', 'c']
```

It takes exactly one argument.

The return value from remove is None.

**insert()**- It inserts an item at a specified position.

```
>>> t = ['a', 'c', 'd']  
>>> t.insert(1, 'b')  
['a', 'b', 'c', 'd']
```

It is important to distinguish between operations that modify lists and operations that create new lists.

For example, the append method modifies a list, but

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print t1
[1, 2, 3]
>>> print t2
None
```

the + operator creates a new list:

```
>>> t3 = t1 + [3]
>>> print t3
[1, 2, 3]
>>> t1 is t3
False
```

# LISTS AND FUNCTIONS

built-in functions that can be used on lists:

```
>>> nums = [3, 41, 12, 9, 74, 15]
```

```
>>> print len(nums)
```

```
6
```

```
>>> print max(nums)
```

```
74
```

```
>>> print min(nums)
```

```
3
```

```
>>> print sum(nums)
```

```
154
```

```
>>> print sum(nums)/len(nums)
```

```
25
```

The `sum()` function only works when the list elements are numbers.

The other functions (`max()`, `len()`, etc.) work with lists of strings and other types that can be comparable.

# EXAMPLES

**# code to find the average numbers using list functions**

```
numlist = list()
while ( True ) :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)
average = sum(numlist) / len(numlist)
print('Average:', average)
```

**Output:**

```
Enter a number: 1
Enter a number: 2
Enter a number: 3
Enter a number: 4
Enter a number: done
Average: 2.5
```

**# Display the integers entered by the user in ascending order.**

```
data=[]
# read values, adding them to the list, until the user enters 0
num = int(input("enter a number - 0 to quit:"))
while num != 0:
    data.append(num)
    num = int(input("enter a number - 0 to quit:"))
```

```
#sort numbers
data.sort()
```

```
#Display the values
print("sorted numbers are")
for num in data:
    print(num,end=" ")
```

**Output:**

```
enter number of elements to be sorted:5
enter a number50
enter a number40
enter a number20
enter a number10
enter a number30
sorted numbers are
10 20 30 40 50
```

# LISTS AND STRINGS

**list():** A string is a sequence of characters and a list is a sequence of values, but a list of characters is not the same as a string. To convert from a string to a list of characters, we can use list() method:

```
>>> s = 'spam'
>>> t = list(s)
>>> print t
['s', 'p', 'a', 'm']
```

The list function breaks a string into individual letters.

**split():** If we want to break a string into words, we can use the split method:

```
>>> txt = "welcome to the jungle"
>>> x = txt.split()
>>> print(x)
['welcome', 'to', 'the', 'jungle']
```

you can use the index operator (square bracket) to look at a particular word in the list

```
>>> print(x[3])
jungle
```



**split(delimiter):** You can call split with an optional argument called a delimiter that specifies which characters to use as word boundaries.

The following example uses a hyphen as a delimiter:

```
>>> s = '10-05-2020'
>>> s.split('-')
['10', '05', '2020']
```

**join():** is the inverse of split. It takes a list of strings and concatenates the elements.

join is a string method, so you have to invoke it on the delimiter and pass the list as a parameter:

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> delimiter.join(t)
'pining for the fjords'
```

**#to print out the day of the week from those lines that start with "From "**

```
fhand = open('mbox.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:'): continue
    words = line.split()
    print(words[2])
```

**Output:**

Sat  
Mon  
Fri

**mbox.txt**

```
From: stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
From: louis@media.berkeley.edu Mon Jan  4 16:10:39 2008
Subject: [sakai] svn commit:
From: zqian@umich.edu Fri Jan  4 16:10:39 2008
Return-Path: <postmaster@collab.sakaiproject.org>
```

# OBJECTS AND VALUES

If we execute these assignment statements:

```
a = 'book'
```

```
b = 'book'
```

In this example, Python only created one string object, and both a and b refer to it.

To check whether **two variables refer to the same object**, you can use the “**is**” operator.

```
>>> a = 'book'
```

```
>>> b = 'book'
```

```
>>> a is b
```

**True**

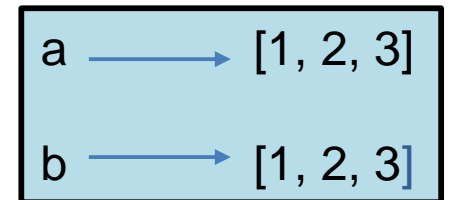
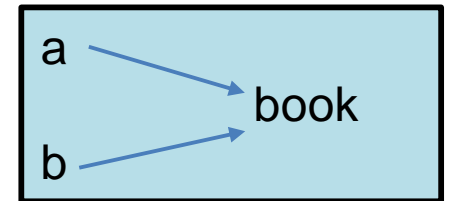
But when you create **two lists**, you get two objects:

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

```
>>> b = [1, 2, 3]
```

```
>>> a is b
```

**False**



# ALIASING

An **object with more than one reference** has more than one name, so we say that the object is aliased.

If `a` refers to an object and we assign `b = a`, then both variables refer to the same object:

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

The association of a variable with an object is called a reference. In this example, there are two references to the same object

If the aliased object is mutable, changes made with one alias affect the other:

```
>>> b[0] = 17
>>> print a
[17, 2, 3]
```

It is important to distinguish between operations that modify lists and operations that create new lists.

For example, the append method modifies a list, but

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print t1
[1, 2, 3]
>>> print t2
None
```

the + operator creates a new list:

```
>>> t3 = t1 + [3]
>>> print t3
[1, 2, 3]
>>> t2 is t3
False
```

```
"""
```

```
Exercise 8.1: Write a function called chop that takes a list and modifies it,  
removing the first and last elements, and returns None.  
Then write a function called middle that takes a list and returns a new list  
that contains all but the first and last elements."""
```

```
def chop(lst):  
    del lst[0] # Removes the first element  
    del lst[-1] # Removes the last element  
  
def middle(lst):  
    new = lst[1:] # Stores all but the first element  
    del new[-1] # Deletes the last element  
    return new  
  
my_list = [1, 2, 3, 4]  
my_list2 = [1, 2, 3, 4]  
  
chop_list = chop(my_list)  
print(my_list) # Should be [2,3]  
print(chop_list) # Should be None  
  
middle_list = middle(my_list2)  
print(my_list2) # Should be unchanged  
print(middle_list) # Should be [2,3]
```

**Output:**

```
[2, 3]  
None  
[1, 2, 3, 4]  
[2, 3]
```

**THANK  
YOU**

# **MODULE 3 – PART 4**

## **REGULAR EXPRESSIONS**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**



# INTRODUCTION

- **Regular expression** is a sequence of characters that define a search pattern.
- patterns are used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation.
- The regular expression **library “re” must be imported** into our program before we can use it.

# SEARCH() FUNCTION:

- **search() function:** used to search for a particular string. will only return the first occurrence that matches the specified pattern.

This function is available in “re” library.

- **the caret character (^)** : is used in regular expressions to match the **beginning of a line**.
- **The dollar character (\$)** : is used in regular expressions to match the **end of a line**.

Example: program to match only lines where “From:” is at the beginning of the line

```
import re
hand = open('mbox1.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print(line)
```

## #Output

```
From:stephen Sat Jan 5 09:14:16 2008
From: louis@media.berkeley.edu Mon Jan 4 16:10:39 2008
From:zqian@umich.edu Fri Jan 4 16:10:39 2008
```

## mbox1.txt

```
From:stephen Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
From: louis@media.berkeley.edu Mon Jan 4 16:10:39 2008
Subject: [sakai] svn commit:
From:zqian@umich.edu Fri Jan 4 16:10:39 2008
Return-Path: <postmaster@collab.sakaiproject.org>
```

- ✓ The instruction `re.search('^From:', line)` equivalent with the `startswith()` method from the string library.

# CHARACTER MATCHING IN REGULAR EXPRESSIONS

- **The dot character (.)** : The most commonly used special character is the period ("dot") or full stop, which matches any character.

The regular expression **"F..m:"** would match any of the following strings since the period characters in the regular expression match any character.

**"From:", "Fxxm:", "F12m:", or "F!@m:"**

- The program in the previous slide is rewritten using dot character which gives the same output

```
import re
hand = open('mbox1.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^F..m:', line):
        print(line)
```

## **#Output**

```
From:stephen Sat Jan 5 09:14:16 2008
From: louis@media.berkeley.edu Mon Jan 4 16:10:39 2008
From:zqian@umich.edu Fri Jan 4 16:10:39 2008
```

Character can be repeated any number of times using the “\*” or “+” characters in a regular expression.

- The Asterisk character (\*) : matches zero-or-more characters
- The Plus character (+) : matches one-or-more characters

Example: Program to match lines that start with “From:”, followed by mail-id

```
import re
hand = open('mbox1.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:.*@', line) :
        print(line)
```

**#Output**

```
From: louis@media.berkeley.edu Mon Jan 4 16:10:39 2008
From:zqian@umich.edu Fri Jan 4 16:10:39 2008
```

- ✓ The search string “^From:.\*@” will successfully match lines that start with “From:”, followed by one or more characters (“.\*”), followed by an at-sign. The “.\*” wildcard matches all the characters between the colon character and the at-sign.

➤ **non-whitespace character (\S)** - matches one non-whitespace character

➤ **findall() function:** It is used to search for “all” occurrences that match a given pattern.

In contrast, search() function will only return the first occurrence that matches the specified pattern.

Example1: Program returns a list of all of the strings that look like email addresses from a given line.

```
import re
s = 'Hello from csev@umich.edu to cwen@iupui.edu about the meeting @2PM'
lst = re.findall('\S+@\S+', s)
print(lst)
```

**#output**

```
['csev@umich.edu', 'cwen@iupui.edu']
```

**'\S+@\S+'** this regular expression matches substrings that have at least one non-whitespace character, followed by an at-sign, followed by at least one more non-whitespace character

**# same program using search() it will display only first mail id or first matching string**

```
import re
s = 'Hello from csev@umich.edu to cwen@iupui.edu about the meeting @2PM'
lst = re.search('\S+@\S+', s)
print(lst)
```

**#output**

```
<re.Match object; span=(11, 25), match='csev@umich.edu'>
```

Example2: Program returns a list of all of the strings that look like email addresses from a given file.

```
import re
hand = open('mbox1.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('\S+@\S+', line)
    if len(x) > 0:
        print(x)
```

**#Output**

```
['<postmaster@collab.sakaiproject.org>']
['louis@media.berkeley.edu']
['zqian@umich.edu']
['<postmaster@collab.sakaiproject.org>']
```

Some of our email addresses have incorrect characters like “<” or “;” at the beginning or end. we are only interested in the portion of the string that starts and ends with a letter or a number. To get the proper output we have to use following character.

- **Square brackets “[]”** : square brackets are used to indicate a set of multiple acceptable characters we are willing to consider matching.

Example: [a-z] matches single lowercase letter

[A-Z] matches single uppercase letter

[a-zA-Z] matches single lowercase letter or uppercase letter

[a-zA-Z0-9] matches single lowercase letter or uppercase letter or number

[amk] matches 'a', 'm', or 'k'

[(+\*)] matches any of the literal characters '(', '+', '\*', or ')'

[0-5][0-9] matches all the two-digits numbers from 00 to 59

## ➤ Characters that are not within a range can be matched by complementing the set

If the first character of the set is '^', all the characters that are not in the set will be matched.

For example,

[^5] will match any character except '5'

Ex: Program returns list of all email addresses in proper format.

```
import re
hand = open('mbox.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S*@ \S*[a-zA-Z]', line)
    if len(x) > 0:
        print(x)
#output
['postmaster@collab.sakaiproject.org']
['louis@media.berkeley.edu']
['zqian@umich.edu']
['postmaster@collab.sakaiproject.org']
```

**[a-zA-Z0-9]\S\*@ \S\*[a-zA-Z]** : substrings that start with a single lowercase letter, uppercase letter, or number "[a-zA-Z0-9]", followed by zero or more non-blank characters "\S\*", followed by an at-sign, followed by zero or more non-blank characters "\S\*", followed by an uppercase or lowercase letter "[a-zA-Z]".

# SEARCH AND EXTRACT

- **parentheses “()” in regular expression** : used to **extract a portion of the substring** that matches the regular expression.

Example1: Find numbers on lines that start with the string “X-”  
lines such as: X-DSPAM-Confidence: 0.8475

```
import re
hand = open('mbox2.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^X\S*: [0-9.]+' , line):
        print(line)
```

## #Output

X-DSPAM-Confidence: 0.8475

X-DSPAM-Probability: 0.9245

Above output has entire line we only want to extract  
numbers from lines that have the above syntax

## mbox2.txt

```
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/sakai_2-5-x/conter
impl/impl/src/java/org
X-Content-Type-Outer-Envelope: text/plain; charset=UTF-8
X-Content-Type-Message-Body: text/plain; charset=UTF-8
Content-Type: text/plain; charset=UTF-8
X-DSPAM-Result: Innocent
X-DSPAM-Processed: Sat Jan 5 09:14:16 2008
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.9245
```

```
import re
hand = open('mbox2.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X\S*: ([0-9.]+' , line)
    if len(x) > 0 :
        print(x)
```

## #Output

['0.8475']

['0.9245']

Search

Extract



## Example2: Program to print the day of received mails

```
import re
hand = open('mbox1.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^From.* ([0-3][0-9]):', line)
    if len(x) > 0 :
        print(x)
```

### #Output

['09']

['16']

['16']

# RANDOM EXECUTION

```
>>> s=" 0.9 .90 1.0 1. 138 pqr"
```

```
>>> re.findall('[0-9.]+' ,s)
['0.9', '.90', '1.0', '1.', '138']
```

```
>>> re.findall('[0-9]+[.][0-9]',s)
['0.9', '1.0']
```

```
>>> re.findall('[0-9]+[.][0-9]+' ,s)
['0.9', '1.0']
```

```
>>> re.findall('[0-9]*[.][0-9]+' ,s)
['0.9', '.90', '1.0']
```

```
>>> usn="1bycs123, 1byec249, 1bycs009, 1byme209, 1byis112, 1byee190"
```

```
>>> re.findall('1bycs...',usn)
['1bycs123', '1bycs009']
```

```
>>> re.findall('[a-zA-Z0-9]+cs[0-9]+' ,usn)
['1bycs123', '1bycs009']
```

```
>>> usn="1bycs123, 1byec249, 1bycs009, 1byme209, 1vecs112, 1svcs190"
```

```
>>> re.findall('[a-zA-Z0-9]+cs[0-9]+' ,usn)
['1bycs123', '1bycs009', '1vecs112', '1svcs190']
```

```
>>> re.findall('[0-9]+cs[0-9]+' ,usn)
[]
```

```
>>> re.findall('[a-zA-Z0-9]+cs([0-9]+)' ,usn)
['123', '009', '112', '190']
```

# ESCAPE CHARACTER

- **Escape character (backslash "\")** is a metacharacter in **regular** expressions. It allow **special characters** to be used without invoking their **special** meaning.

If you want to match `1+1=2`, the correct **regex** is `1\+1=2`. Otherwise, the plus sign has a special meaning.

For example, we can find money amounts with the following regular expression.

```
>>>import re
>>>x = 'We just received $10.00 for cookies.'
>>>y = re.findall('\$[0-9.]+',x)
>>>y
['$10.00']
```

# SUMMARY

Character	Meaning
<code>^</code>	Matches the beginning of the line
<code>\$</code>	Matches the end of the line
<code>.</code>	Matches any character (a wildcard)
<code>\s</code>	Matches a whitespace character
<code>\S</code>	Matches a non-whitespace character (opposite of <code>\s</code> )
<code>*</code>	Applies to the immediately preceding character and indicates to match zero or more of the preceding character(s)
<code>*?</code>	Applies to the immediately preceding character and indicates to match zero or more of the preceding character(s) in “non-greedy mode”
<code>+</code>	Applies to the immediately preceding character and indicates to match one or more of the preceding character(s)
<code>+?</code>	Applies to the immediately preceding character and indicates to match one or more of the preceding character(s) in “non-greedy mode”.
<code>[aeiou]</code>	Matches a single character as long as that character is in the specified set. In this example, it would match “a”, “e”, “i”, “o”, or “u”, but no other characters.
<code>[a-z0-9]</code>	You can specify ranges of characters using the minus sign. This example is a single character that must be a lowercase letter or a digit.

Character	Meaning
[^A-Za-z]	When the first character in the set notation is a caret, it inverts the logic. This example matches a single character that is anything other than an uppercase or lowercase letter.
( )	When parentheses are added to a regular expression, they are ignored for the purpose of matching, but allow you to extract a particular subset of the matched string rather than the whole string when using findall()
\b	Matches the empty string, but only at the start or end of a word.
\B	Matches the empty string, but not at the start or end of a word
\d	Matches any decimal digit; equivalent to the set [0-9].
\D	Matches any non-digit character; equivalent to the set [^0-9]

# ASSIGNMENT

1) Write a python program to check the validity of a Password In this program, we will be taking a password as a combination of alphanumeric characters along with special characters, and check whether the password is valid or not with the help of few conditions.

## **Primary conditions for password validation :**

1. Minimum 8 characters.
2. The alphabets must be between [a-z]
3. At least one alphabet should be of Upper Case [A-Z]
4. At least 1 number or digit between [0-9].
5. At least 1 character from [ \_ or @ or \$ ].

2) Write a pattern for the following:

Pattern to extract lines starting with the word From (or from) and ending with edu.

Pattern to extract lines ending with any digit.

Start with upper case letters and end with digits.

Search for the first white-space character in the string and display its position.

Replace every white-space character with the number 9: consider a sample text `txt = "The rain in Spain"`

**THANK  
YOU**

# **MODULE 3 – PART 3**

## **TUPLES**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**



# WHAT IS A TUPLE?

A tuple is a sequence of values much like a list which can be represented using parenthesis ( ) with comma separator.

The values stored in a tuple can be any type and they are indexed by integers.

The important difference is that tuples are immutable.

**Example :**

```
#creating a tuple  
a = ('ruby', 'java')  
  
#another approach  
b = 'EEE' , 'ME'  
print(a)  
print(b)
```

**Output:** ('ruby' , 'java')  
('EEE' , 'ME')

# CONTD..

- Tuples are also comparable and hashable so we can sort lists of them and use tuples as key values in Python dictionaries.
- If our data is fixed and never changes then we should go for Tuple.
- Insertion Order is preserved
- Duplicates are allowed
- Heterogeneous objects are allowed.
- Tuple support both +ve and -ve index.
  - +ve index forward direction(from left to right)
  - ve index means backward direction(from right to left)

```
t=10,20,30,40  
print(t)  
print(type(t))
```

**#Output :**

```
(10, 20, 30, 40)  
<class 'tuple'>
```

**#Empty tuple**

```
t=()  
print(type(t))
```

**#Output :**

```
<class 'tuple'>
```

# TUPLES ARE IMMUTABLE

Once we create a tuple, we cannot change its content. Hence tuple objects are immutable.

Eg:

```
t=(10,20,30,40)
```

```
t[1]=70
```

```
TypeError: 'tuple' object does not support item assignment
```

To create a tuple with a single element, you have to include a final comma:

```
>>> t1 = 'a',
```

```
>>> type(t1)
```

```
<class 'tuple'>
```

A value in parentheses is not a tuple:

```
>>> t2 = ('a')
```

```
>>> type(t2)
```

```
<class 'str'>
```

```
>>> t2 = (10)
```

```
>>> type(t2)
```

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

## Valid Tuples

1. `t=()`
2. `t=10,20,30,40`
3. `t=10,`
4. `t=(10,)`
5. `t=(10,20,30,40)`

# CREATING TUPLES

There are 2 ways of creating tuples

## Using parenthesis

1. `t=()`  
creation of empty tuple
2. `t=(10,)`  
`t=10,`  
creation of single valued tuple ,parenthesis are optional ,should end with comma
3. `t=10,20,30`  
`t=(10,20,30)`  
creation of multi values tuples & parenthesis are optional

## Using tuple() -Function

```
list=[10,20,30]
t=tuple(list)
print(t)
#Output: (10,20,30)

t=tuple(range(10,20,2))
print(t)
#Output: (10,12,14,16,18)

t = tuple('lupins')
print( t)
#Output : ('l', 'u', 'p', 'i', 'n', 's')
```

# ACCESSING ELEMENTS OF TUPLE

## 1. Using indexing

```
t=(10,20,30,40,50,60)
```

```
print(t[0])    #10
```

```
print(t[-1])   #60
```

```
print(t[100])  #IndexError: tuple index out of range
```

```
t[0]= 25       #TypeError: object doesn't support item assignment
```

## 2. By using slice operator:

```
t=(10,20,30,40,50,60)
```

```
print(t[2:5])
```

```
print(t[2:100])
```

```
print(t[-5:-2])
```

Output : (30, 40, 50)

(30, 40, 50, 60)

(20,30,40)

-6	-5	-4	-3	-2	-1
10	20	30	40	50	60
0	1	2	3	4	5

# MATHEMATICAL OPERATORS

- 1. Concatenation (+)
- 2. Multiplication (\*)
- 3. Relational (< > <= >= == !=)
- 4. Membership (in and not in)

<p><b>Concatenation</b></p> <pre>t1=(10,20,30) t2=(40,50,60) t3=t1+ t2 print(t3) Output: (10,20,30,40,50,60)</pre>	<p><b>Multiplication</b></p> <pre>t1=(10,20,30) t2=t1*3 print(t2) Output: (10,20,30,10,20,30,10,20,30)</pre>
<p><b>Relational (Comparing):</b></p> <p>compares the first element from each sequence. It goes on to the next elements until it finds elements that differ. Subsequent elements are not considered (even if they are really big).</p> <pre>&gt;&gt;&gt; (0, 1, 2) &lt; (0, 3, 4) True &gt;&gt;&gt; (0, 1, 2000000) &lt; (0, 3, 4) True &gt;&gt;&gt; (10,13)==(10,19) False &gt;&gt;&gt; (10,13)==(10,13) True</pre>	<p><b>Membership (in and not in)</b></p> <pre>&gt;&gt;&gt; t1=(10, 12, 14, 16, 18) &gt;&gt;&gt; 10 in t1 True &gt;&gt;&gt; 17 in t1 False &gt;&gt;&gt; 17 not in t1 True &gt;&gt;&gt; 'a' in tuple('string') False &gt;&gt;&gt; 'n' in tuple('string') True</pre>

# TUPLE FUNCTIONS [FOR REFERNCE]

<b>len()</b>	Returns the length or the number of elements	<pre>&gt;&gt;&gt; tuple1 = (10,20,30,40,50) &gt;&gt;&gt; len(tuple1) 5</pre>
<b>tuple()</b>	Creates a tuple if a sequence is passed as argument	<pre>&gt;&gt;&gt; tuple2 = tuple([1,2,3]) #list &gt;&gt;&gt; tuple2 (1, 2, 3)</pre>
<b>count()</b>	Returns the number of times the given element appears in the tuple	<pre>&gt;&gt;&gt; tuple1=(10,20,30,10,40,10,50) &gt;&gt;&gt; tuple1.count(10) 3</pre>
<b>index()</b>	Returns the index of the first occurrence of the element in the given tuple	<pre>&gt;&gt;&gt; tuple1 = (10,20,30,40,50) &gt;&gt;&gt; tuple1.index(30) 2</pre>
<b>sorted()</b>	Takes elements in the tuple and returns a new sorted list. It should be noted that, sorted() does not make any change to the original tuple	<pre>&gt;&gt;&gt; tuple1=("Rama","Heena","Raj") &gt;&gt;&gt; sorted(tuple1) ['Heena', 'Raj', 'Rama']</pre>
<b>min()</b> <b>max()</b> <b>sum()</b>	Returns minimum element of the tuple Returns maximum element of the tuple Returns sum of the elements of the tuple	<pre>&gt;&gt;&gt; tuple1 = (19,12,56,18,9,87,34) &gt;&gt;&gt; min(tuple1)      #9 &gt;&gt;&gt; max(tuple1)      # 87 &gt;&gt;&gt; sum(tuple1)      #235</pre>

# TUPLE ASSIGNMENT

One of the unique syntactic features of the Python language is the ability to have a tuple on the left side of an assignment statement. This allows you to assign more than one variable at a time when the left side is a sequence.

In this example we have a two-element list (which is a sequence) and assign the first and second elements of the sequence to the variables x and y in a single statement.

```
>>> m = ('have', 'fun' )
```

```
>>> x, y = m
>>> x
'have'
>>> y
'fun'
```

OR

```
>>> x = m[0]
>>> y = m[1]
>>> x
'have'
>>> y
'fun'
```

A particularly clever application of tuple assignment allows us to swap the values of two variables in a single statement:

```
>>> a, b = b, a
```

**The number of variables on the left and the number of values on the right must be the same:**

```
>>> a, b = 1, 2, 3
```

**ValueError: too many values to unpack**

```
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')
```

**The return value from split is a list with two elements; the first element is assigned to uname, the second to domain.**

```
>>> print(uname)    #monty
>>> print(domain)   #python.org
```



# SORTING IN TUPLES- LISTS OF TUPLES

The sort function works in this way. It sorts primarily by first element, but in the case of a tie, it sorts by second element, and so on.

This feature lends itself to a pattern called **DSU** for

**Decorate** a sequence by building a list of tuples with one or more sort keys preceding the elements from the sequence,

**Sort** the list of tuples using the Python built-in sort, and

**Undecorate** by extracting the sorted elements of the sequence.

# SAMPLE

## #sample list

```
>>> t=[1,2,3]
>>> t.append(4)
>>> t
```

```
[1, 2, 3, 4]
```

## #append a tuple into a list

```
>>> t.append((2,5))
>>> t
```

```
[1, 2, 3, 4, (2, 5)]
```

## #creating a list of tuples - exapmle1

```
>>> t1=[(21,'w'),(39,'x'),(19,'y')]
>>> t1
```

```
[(21, 'w'), (39, 'x'), (19, 'y')]
```

## #sorting list of tuples

```
>>> t1.sort()
>>> t1
```

```
[(19, 'y'), (21, 'w'), (39, 'x')]
```

## #sorting reverse

```
>>> t1.sort(reverse=True)
>>> t1
```

```
[(39, 'x'), (21, 'w'), (19, 'y')]
```

## #example2

```
>>> t2=[(19,'z'),(21,'w'),(39,'x'),(19,'y')]
>>> t2.sort()
>>> t2
```

```
[(19, 'y'), (19, 'z'), (21, 'w'), (39, 'x')]
```

```
>>> t2.sort(reverse=True)
>>> t2
```

```
[(39, 'x'), (21, 'w'), (19, 'z'), (19, 'y')]
```

## #example3

```
>>> m=[('akash','24.5'),('nikil','18.2'),('anand','4.2')]
>>> m
```

```
[('akash', '24.5'), ('nikil', '18.2'), ('anand', '4.2')]
```

```
>>> m.sort()
>>> m
```

```
[('akash', '24.5'), ('anand', '4.2'), ('nikil', '18.2')]
```

```
>>> m.sort(reverse=True)
>>> m
```

```
[('nikil', '18.2'), ('anand', '4.2'), ('akash', '24.5')]
```

## WRITE A PROGRAM TO DISPLAY A LIST OF WORDS FROM LONGEST TO SHORTEST

```
txt = 'but soft what light in yonder window breaks'
```

```
words = txt.split()
```

```
t = list()
```

```
for word in words:
```

```
    t.append((len(word), word))
```

```
print(t)
```

```
t.sort(reverse=True)
```

```
print(t)
```

```
res = list()
```

```
for length, word in t:
```

```
    res.append(word)
```

```
print("Sorted list from longest to shortest\n" res)
```

**#Output:**

# finding the length for each word and printing it

```
[(3, 'but'), (4, 'soft'), (4, 'what'), (5, 'light'), (2, 'in'), (6, 'yonder'), (6, 'window'), (6, 'breaks')]
```

#printing list in reverse order

```
[(6, 'yonder'), (6, 'window'), (6, 'breaks'), (5, 'light'), (4, 'what'), (4, 'soft'), (3, 'but'), (2, 'in')]
```

#printing only words according to problem statement

Sorted list from longest to shortest

```
['yonder', 'window', 'breaks', 'light', 'what', 'soft', 'but', 'in']
```

# WRITE A PROGRAM TO FIND MOST COMMONLY USED WORDS IN A TEXT FILE.

```
import string
fhand = open('sample.txt')
counts = dict()
for line in fhand:
    line = line.translate(str.maketrans("", "", string.punctuation))
    line = line.lower()
    for word in line.split():
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

lst = list()
for key, val in list(counts.items()):
    lst.append((val, key))
lst.sort(reverse=True)
for key, val in lst[:10]:
    print(key, val)
```

## Output

4 tuples  
4 lists  
2 use  
2 tuple  
2 the  
2 is  
2 as  
2 are  
2 and  
2 a

## #sample.txt

A tuple is an immutable sequence of Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values.

# DIFFERENCES BETWEEN LIST AND TUPLE:

List	Tuple
List objects are mutable	Tuple objects are immutable.
insertion order is preserved, duplicate objects are allowed, heterogeneous objects are allowed, index and slicing are supported.	insertion order is preserved, duplicate objects are allowed, heterogeneous objects are allowed, index and slicing are supported.
List is a Group of Comma separated Values within Square Brackets and Square Brackets are mandatory. Eg: i = [10, 20, 30, 40]	Tuple is a Group of Comma separated Values within Parenthesis and Parenthesis are optional. Eg: t = (10, 20, 30, 40)      t = 10, 20, 30, 40
If the Content is not fixed and keep on changing then we should go for List.	If the content is fixed and never changes then we should go for Tuple.
List Objects can not used as Keys for Dictionaries because Keys should be Hashable and Immutable.	Tuple objects can be used as Keys for Dictionaries because Keys should be Hashable and Immutable.

# TUPLES AND DICTIONARIES

Dictionaries have a method called `items()` that returns a list of tuples, where each tuple is a key-value pair:

**# from dictionary to list of tuples**

```
>>> d = {'a':10, 'b':1, 'c':22}
```

```
>>> t = list(d.items())
```

```
>>> print(t)
```

```
[('b', 1), ('a', 10), ('c', 22)]
```

As you should expect from a dictionary, the items are in no particular order. However, since the list of tuples is a list, and tuples are comparable, we can now sort the list of tuples.

**Converting a dictionary to a list of tuples is a way for us to output the contents of a dictionary sorted by key:**

```
>>> t.sort()
```

```
>>> t
```

```
[('a', 10), ('b', 1), ('c', 22)]
```

**#The new list is sorted in ascending alphabetical order by the key value.**

# MULTIPLE ASSIGNMENT WITH DICTIONARIES

- We can combine the method `items()`, tuple assignment and a for-loop to get a pattern for traversing dictionary:

```
d={'Tom': 1292, 'Jerry': 3501, 'Donald': 8913}  
for key, val in list(d.items()):  
    print(val,key)
```

**Output:**  
1292 Tom  
3501 Jerry  
8913 Donald

- This loop has two iteration variables because `items()` returns a list of tuples.
- And `key, val` is a tuple assignment that successively iterates through each of the key-value pairs in the dictionary.
- For each iteration through the loop, both key and value are advanced to the next key-value pair in the dictionary in hash order.

# CONTD..

Once we get a key-value pair, we can create a list of tuples and sort them

```
d={'Tom': 9291, 'Jerry': 3501, 'Donald': 8913} ls=list()
for key, val in d.items():
    ls.append((val,key))      #observe inner parentheses
print("List of tuples:",ls)
ls.sort(reverse=True)
print("List of sorted tuples:",ls)
```

**The output would be –**

List of tuples:

[(9291, 'Tom'), (3501, 'Jerry'), (8913, 'Donald')]

List of sorted tuples:

[(9291, 'Tom'), (8913, 'Donald'), (3501, 'Jerry')]

- In the above program, we are extracting key, val pair from the dictionary and appending it to the list
- While appending, we are putting inner parentheses to make sure that each pair is treated as a tuple.
- Then, we are sorting the list in the descending order.



# USING TUPLES AS KEYS IN DICTIONARIES

As tuples and dictionaries are hashable, when we want a dictionary containing composite keys, we will use tuples.

We wanted to create a telephone directory that maps from last-name, first-name pairs to telephone numbers.

Write a dictionary assignment statement as follows:

```
directory[last,first] = number
```

The expression in brackets is a tuple. We could use tuple assignment in a for loop to traverse this dictionary.

```
for last, first in directory:
```

```
    print(first, last, directory[last,first])
```

This loop traverses the keys in directory, which are tuples. It assigns the elements of each tuple to last and first, then prints the name and corresponding telephone number.

```
>>> di={'r','b':90,('c','s'):98}
```

```
>>> di
```

```
    {'r', 'b': 90, ('c', 's'): 98}
```

```
>>> for fname,lname in di:
```

```
    print(di[fname,lname])
```

```
    90
```

```
    98
```

```
>>> for fname,lname in di:
```

```
    print(fname,lname, di[fname,lname])
```

```
    r b 90
```

```
    c s 98
```

```
>>> for fname,lname in di:
```

```
    print(fname,lname,"=", di[fname,lname])
```

```
    r b = 90
```

```
    c s = 98
```

# SUMMARY ON SEQUENCES: STRINGS, LISTS AND TUPLES

1. Strings are more limited compared to other sequences like lists and Tuples. Because, the elements in strings must be characters only. Moreover, strings are immutable. Hence, if we need to modify the characters in a sequence, it is better to go for a list of characters than a string.
2. As lists are mutable, they are most common compared to tuples. But, in some situations as given below, tuples are preferable.
  - a. When we have a return statement from a function, it is better to use tuples rather than lists.
  - b. When a dictionary key must be a sequence of elements, then we must use immutable type like strings and tuples
  - c. When a sequence of elements is being passed to a function as arguments, usage of tuples reduces unexpected behaviour due to aliasing.
3. As tuples are immutable, the methods like `sort()` and `reverse()` cannot be applied on them. But, Python provides built-in functions `sorted()` and `reversed()` which will take a sequence as an argument and return a new sequence with modified results.

# TUPLES WONT SUPPORT SORT():

```
>>> t1=(10,29,12,45)
```

```
>>> t1.sort()
```

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

**File "<pyshell#52>", line 1, in <module>**

**t1.sort()**

**AttributeError: 'tuple' object has no attribute 'sort'**

```
>>> sorted(t1)
```

```
[10, 12, 29, 45]
```

```
>>> list(reversed(t1))
```

```
[45, 12, 29, 10]
```

# ASSIGNMENT:

1) WAP create a telephone directory where name of a person is Firstname- last name pair and value is the telephone number by using Tuples as Keys in Dictionary.

Ex: **names= (('Tom','Cat'),('Jerry','Mouse'), ('Donald', 'Duck'))**  
**Tel number=[3561, 4014, 9813]**

('Cleeese', 'John')	→	'08700 100 222'
('Chapman', 'Graham')	→	'08700 100 222'
('Idle', 'Eric')	→	'08700 100 222'
('Gilliam', 'Terry')	→	'08700 100 222'
('Jones', 'Terry')	→	'08700 100 222'
('Palin', 'Michael')	→	'08700 100 222'

2) Compare and contrast strings, lists, dictionaries and tuples.

**THANK  
YOU**

# **MODULE 4 – PART 2**

## **CLASSES & FUNCTIONS**

By,

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# PYTHON FUNCTIONS

Though Python is object oriented programming languages, it is possible to use it as functional programming.

There are two types of functions viz.

- ❑ Pure functions and

- ❑ Modifiers functions.

A **pure function** takes objects as arguments and does some work without modifying any of the original argument.

On the other hand, as the name suggests, **modifier function** modifies the original argument.



# TIME

Consider an example for Time class. The below class represents the time of the day.

A time object is created, which has instance variables hour, minute and second. The state diagram of Time object is shown above.

```
class Time:
```

```
    """Represents the time of day.  
    attributes: hour, minute, second"""
```

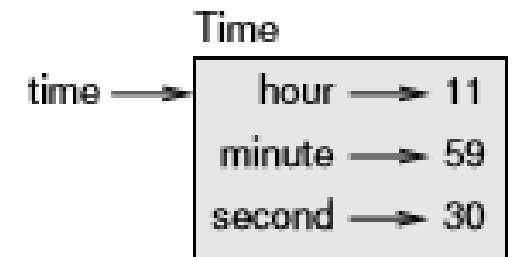
```
time = Time()  
time.hour = 11  
time.minute = 59  
time.second = 30
```

```
print_time(time)
```

```
"""print_time( ) is a function to print in the format  
hour:minute:second"""
```

```
def print_time(t):  
    print("%g : %g: %g" %(t.hour,t.minute,t.second))
```

```
11:59:30
```



# PURE FUNCTIONS

➤ **Definition:** A function that does not modify any of the objects it receives as arguments.

➤ A pure function is a function where the return value is only determined by its input values, without observable side effects.

Ex: `Math.cos(x)` will, for the same value of `x`, always return the same result.

➤ A pure function is a function that doesn't cause or rely on *side effects*. The output of a pure function should only depend on its inputs. It communicates with the calling program only through parameters (which it does not modify) and a return value.

## Example:

➤ Let us consider an example of creating a class called `Time`. An object of class `Time` contains hour, minutes and seconds as attributes.

➤ **Write a function to print time in HH:MM:SS format and another function to add two time objects.** Here, the function `add_time()` takes two arguments of type `Time`, and returns a `Time` object, whereas, it is not modifying contents of its arguments `t1` and `t2`. Such functions are called as pure functions.

➤ *Note that, adding two time objects should yield proper result and hence we need to check whether number of seconds exceeds 60, minutes exceeds 60 etc, and take appropriate action.*

```
class Time:
    """Represents the time of day.
    attributes: hour, minute, second"""
```

```
def print_time(t):
    print("%g : %g: %g" %(t.hour,t.minute,t.second))
```

```
def add_time(t1,t2):
    sum=Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second
    if sum.second >= 60:
        sum.second = sum.second - 60
        sum.minute = sum.minute + 1
    if sum.minute >= 60:
        sum.minute= sum.minute - 60
        sum.hour += 1
    return sum
```

```
t1=Time()
t1.hour=11
t1.minute=59
t1.second=30
print("Time1 is:")
printTime(t1)
```

```
t2=Time()
t2.hour=12
t2.minute=30
t2.second=10
print("Time2 is:")
printTime(t2)
```

```
t3=add_time(t1,t2)
print("After adding two time objects:")
printTime(t3)
```

### Output:

```
Time1 is 11 : 59: 30
```

```
Time 2 is
12 : 30: 10
```

```
After adding two time objects:
24 : 29: 40
```

# MODIFIERS

Sometimes, it is necessary to modify the underlying argument so as to reflect the caller. That is, arguments have to be modified inside a function and these modifications should be available to the caller. The functions that perform such modifications are known as **modifier function**.

**Definition:** A function that changes one or more of the objects it receives as arguments. Most modifiers are void; that is, they return None.

Increment( ) which adds a given number of seconds to a Time object, can be written naturally as a modifier.

```
def increment(time, seconds):  
    time.second += seconds  
    while time.second >= 60:  
        time.second -= 60  
        time.minute += 1  
    while time.minute >= 60:  
        time.minute -= 60  
        time.hour += 1
```

```
increment(time1,55)  
print_time(time1)
```

**Output:**

**12 : 30: 10**

**After calling increment()**

**12 : 31: 05**

Modifiers have side effects.

# PROTOTYPING V/S PLANNING

A development plan that involves writing a rough draft of a program, testing, and correcting errors as they are found.- *Prototype and Patch*

Whenever we do not know the complete problem statement, we may write the program initially, and then keep on modifying it as and when requirement (problem definition) changes. **This methodology is known as prototype and patch.**

That is, first design the prototype based on the information available and then perform patch-work as and when extra information is gathered.

But, this type of incremental development may end-up in unnecessary code, with many special cases and it may be unreliable too.

An alternative is designed development, in which high-level insight into the problem can make the programming much easier.

For example, if we consider the problem of adding two time objects, adding seconds to time object etc. as a problem involving numbers with base 60 (as every hour is 60 minutes and every minute is 60 seconds), then our code can be improved.

# ANOTHER EXAMPLE TO IMPROVE THE CODE

## Function that converts Times to integers:

```
def time_to_int(time):  
    minutes = time.hour * 60 + time.minute  
    seconds = minutes * 60 + time.second  
    return seconds
```

## Function that converts an integer to a Time

```
def int_to_time(seconds):  
    time = Time()  
    minutes, time.second = divmod(seconds, 60)  
    time.hour, time.minute = divmod(minutes, 60)  
    return time
```

*Note : divmod divides the first argument by the second and returns the quotient and remainder as a tuple*

**Ex: 12:20:35**

**Min= (12\*60)mins+20 mins  
= 720 +20=740 mins**

**Sec= (740 \* 60) +35**

## Rewrite add\_time( ) function with

```
def add_time(t1, t2):  
    seconds = time_to_int(t1) + time_to_int(t2)  
    return int_to_time(seconds)
```

# FUNCTION TO CHECK WHETHER THE TIME IS VALID OR NOT

A Time object is well-formed if the values of minute and second are between 0 and 60 (including 0 but not 60) and if hour is positive. hour and minute should be integral values, but we might allow second to have a fraction part. **Requirements like these are called invariants** because they should always be true.

Invariant: A condition that should always be true during the execution of a program.

```
def valid_time(time):  
    if time.hour < 0 or time.minute < 0 or time.second < 0:  
        return False  
    if time.hour >= 25 or time.minute >= 60 or time.second >= 60:  
        return False  
    return True
```

```
t1= Time()  
t1.hour=11  
t1.minute=59  
t1.second=30  
print(valid_time(t1))
```

**Assert statement:** A statement that check a condition and raises an exception if it fails.

The usage of assert is shown here –

```
def add_time(t1, t2):  
    assert valid_time(t1) and valid_time(t2)  
    seconds = time_to_int(t1) + time_to_int(t2)  
    return int_to_time(seconds)
```

In python assert keyword helps in achieving this task. This statement simply takes input a boolean condition, which when returns true doesn't return anything, but if it is computed to be false, then it raises an AssertionError along with the optional message provided.



# WAP TO GET CURRENT DATE AND TIME AND PRINT THE DAY OF THE WEEK

```
import datetime
import datetime
print("Current date and time: ", datetime.datetime.now())
print("Current year: ", datetime.date.today().strftime("%Y"))
print("Month of year: ", datetime.date.today().strftime("%B"))
print("Day of year: ", datetime.date.today().strftime("%j"))
print("Day of the month : ", datetime.date.today().strftime("%d"))
print("Day of week: ", datetime.date.today().strftime("%A"))
```

## Output:

**Current date and time: 2020-04-26 06:18:04.752444**

**Current year: 2020**

**Month of year: April**

**Day of year: 117**

**Day of the month : 26**

**Day of week: Sunday**

**strftime()** converts a tuple or struct\_time representing a time as returned by gmtime() or localtime() to a string as specified by the format argument.

# WAP THAT USES DATETIME MODULE WITHIN A CLASS, INPUT BIRTHDAY AND CALCULATE AGE, AND NUMBER OF DAYS, HOURS, MINUTES AND SECONDS

```
import datetime
currentDate = datetime.datetime.now()

birthday= input ('Plz enter your date of birth (mm/dd/yyyy) ')
birthday_date=
datetime.datetime.strptime(birthday,'%m/%d/%Y')
print (birthday_date)

daysLeft = birthday_date – currentDate
print(daysLeft)

years = ((daysLeft.total_seconds())/(365.242*24*3600))
yearsInt=int(years)

months=(years-yearsInt)*12
monthsInt=int(months)

days=(months-monthsInt)*(365.242/12)
daysInt=int(days)
```

```
hours = (days-daysInt)*24
hoursInt=int(hours)

minutes = (hours-hoursInt)*60
minutesInt=int(minutes)seconds = (minutes-
minutesInt)*60

secondsInt =int(seconds)

print('You are {0:d} years, {1:d} months, {2:d} days,
{3:d} hours, {4:d} \ minutes, {5:d} seconds
old.'.format(yearsInt,monthsInt,daysInt,hoursInt,minut
esInt,secondsInt))
```

**NOTE: USE THE SAME PROGRAM BY USING CLASS**

**THANK  
YOU**

# **MODULE 4 – PART 3**

## **CLASSES & METHODS**

By,

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# OBJECT-ORIENTED FEATURES

As an object oriented programming language, Python possess following characteristics:

- Programs include class and method definitions.
- Most of the computation is expressed in terms of operations on objects.
- Objects often represent things in the real world, and methods often correspond to the ways objects in the real world interact.

**Method:** a method is a function that is associated with a particular class.

Functions	Methods
Functions are defined outside the class, & there is no association between class & function	Methods are defined inside a class definition in order to make the relationship between the class and the method explicit.
The syntax of invoking a function is <b>Syntax: var = function ( )</b>	The syntax for invoking a method is <b>Syntax: var = object_reference.method( )</b>
Function is independent of object but cannot access class variables directly.	The method is implicitly used for an object for which it is called and is accessible to data that is contained within the class.

# CLASS METHODS

## General Method Syntax

```
class ClassName:
    #Define attributes of the class
    def method_name():
        .....
        # Method Body
        .....
```

## Accessing the method

Object.method\_name( )

OR

Classname.method\_name(objectname)

## class Time:

```
    """Represents the time of day
    attributes: hour, minute, second"""
```

## def print\_time(t):

```
    print('%0.2d:%0.2d:%0.2d' % (t.hour, t.minute, t.second))
```

**#To call this function, you have to pass a Time object as an argument:**

```
start= Time()
```

```
start.hour = 11
```

```
start.minute = 59
```

```
start.second = 30
```

```
start.print_time()
```

or

```
Time.print_time(start) #Output 11:59:30
```

# REWRITING THE CODE USING - SELF KEYWORD

Here, the magic keyword "self" represents the instance of the class. It binds the attributes with the given arguments. Usage of "self" in class to access the methods and attributes

```
class Time:  
    """Represents the time of day  
    attributes: hour, minute, second"""  
    def print_time(self):  
        print('%0.2d:%0.2d:%0.2d' % (self.hour, self.minute, self.second))
```

Invoking would be the same (start is an object of class Time)

Time.print\_time(start)

OR

start.print\_time()

Where the object **start** gets assigned to the parameter self.

# THE INIT METHOD

- The init method (short for “initialization”) is a special method that gets invoked when an object is instantiated. Its full name is `__init__` (two underscore characters, followed by init, and then two more underscores).
- The init method is similar to constructors in C++/Java.
- Constructors are used to initialize the object’s state. The name of the constructor should be `__init__(self)`. Constructor will be executed automatically at the time of object creation.

<pre>class Time:     def __init__(self, hour=0, minute=0, second=0):         self.hour = hour         self.minute = minute         self.second = second</pre>	<p>If you provide one argument, it overrides hour:</p> <pre>time = Time(9) time.print_time() #Output is 09:00:00</pre>
<p>Passing parameters are optional. So, if</p> <pre>time = Time() time.print_time() #Output is 00:00:00</pre>	<p>If you provide two arguments, they override hour and minute.</p> <pre>time = Time(9, 45, 36) time.print_time() #Output is 09:45:36</pre>



# CONTD..

- Most importantly, notice that we do not explicitly call the `__init__` method but pass the arguments in the parentheses
- Every method of any class must have the first argument as **self**. The argument `self` is a reference to the current object. That is, it is reference to the object which invoked the method. (Those who know C++, can relate `self` with this pointer).

Class method	Constructor
Name of method can be any name	Constructor name should be always <code>__init__</code>
Method will be executed only when it is called	Constructor will be executed automatically at the time of object creation.
Inside method we can write business logic	Inside Constructor we have to declare and initialize instance variables

# RECTANGLE PROGRAM

```
class Rectangle:
    """Represents the rectangle"""
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def get_perimeter(self):
        return 2 * (self.length + self.breadth)

    def get_area(self):
        return self.length * self.breadth

rect = Rectangle(160, 120)
print("Area of Rectangle: %d cm^2" % (rect.get_area()))
print("Perimeter of rectangle is : %d " %(rect.get_perimeter()))
```

## Output:

**Area of Rectangle: 19200 cm^2**  
**Perimeter of rectangle is : 560**

# THE \_\_STR\_\_ METHOD

- `__str__` is a special method, that is supposed to return a string representation of an object. This method is especially used for debugging.
- This method is called when `print()` or `str()` function is invoked on an object.
- In fact, `str ()` method will return the string format what we have given inside it, and that string will be printed by `print()` method. If we don't implement `__str__()` function for a class, then built-in object implementation is used.

**Output:**

**Person(name=Jack, age=25)**

**Person(name=Jack, age=25)**

**when we write just `print()` in the main part of the program, the `__str __()` method will be invoked automatically.**

```
class Person:
    name = ""
    age = 0
    def __init__(self, personName, personAge):
        self.name = personName
        self.age = personAge
    def __str__(self):
        return 'Person(name='+self.name+',age='+str(self.age)+')'

p = Person('Jack',25)
print(p)
print(p.__str__())
```

# CONTD...

```
class Time
    """Represents the time of day."""
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour, self.minute,
self.second)

time = Time(13,9, 45)
print(time)
```

**Output:**  
12:24:56

# OPERATOR OVERLOADING

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists.

It is achievable because '+' operator is overloaded by int class and str class.

Ex:

```
#addition of 2 numbers
print(1 + 2)
# concatenate two strings
print("BMSIT"+"& M")

# Product two numbers
print(3 * 4)
# Repeat the String
print("BMS"*4)
```

[Python operators](#) work for built-in classes. But same operator behaves differently with different types. Ability of an existing operator to work on user-defined data type (class) is known as operator overloading.

To overload an operator, one needs to write a method within user-defined class. Python provides a special set of methods which have to be used for overloading required operator.

<code>__add__(self, other)</code>	<code>a+b</code>
<code>__sub__(self, other)</code>	<code>a-b</code>
<code>__mul__(self, other)</code>	<code>a*b</code>
<code>__floordiv__(self, other)</code>	<code>a // b</code>
<code>__div__(self, other)</code>	<code>a / b</code>
<code>__truediv__(self, other)</code>	<code>a / b (from __future__ import division)</code>
<code>__mod__(self, other)</code>	<code>a % b</code>
<code>__divmod__(self, other)</code>	<code>divmod(a, b)</code>
<code>__pow__</code>	<code>a ** b</code>
<code>__lshift__(self, other)</code>	<code>a &lt;&lt; b</code>
<code>__rshift__(self, other)</code>	<code>a &gt;&gt; b</code>
<code>__and__(self, other)</code>	<code>a &amp; b</code>
<code>__or__(self, other)</code>	<code>a   b</code>
<code>__xor__(self, other)</code>	<code>a ^ b</code>

```

class Time
    """Represents the time of day."""
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
    def __str__(self):
        return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)
    def __add__(self, other):
        seconds = self.time_to_int() + other.time_to_int()
        return int_to_time(seconds)
start= Time(13,9, 45)
duration = Time(1, 35,10)
print(start + duration)

```

## Adding these 2 functions inside the class

### Function that converts Times to integers:

```

def time_to_int(time):
    minutes = time.hour * 60 + time.minute
    seconds = minutes * 60 + time.second
    return seconds

```

### Function that converts an integer to a Time

```

def int_to_time(seconds):
    time = Time()
    minutes, time.second = divmod(seconds, 60)
    time.hour, time.minute = divmod(minutes, 60)
    return time

```

*Note : divmod divides the first argument by the second and returns the quotient and remainder as a tuple*

# EXERCISE PROBLEMS:

- 1) Write an init method for the Point class that takes x and y as optional parameters and assigns them to the corresponding attributes.



**THANK  
YOU**

# **MODULE 4 – PART 1**

## **CLASSES & OBJECTS**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# INTRODUCTION

- ❖ Python is a multi-paradigm programming language. One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).
- ❖ Class is a user-defined data type which binds data and functions together into single entity
- ❖ An object has two characteristics:
  - ❖ Attributes(member variables)
  - ❖ Behavior (member functions ,Methods)
- ❖ Class is just a prototype (or a logical entity/blue print) which will not consume any memory.
- ❖ An object is an instance of a class and it has physical existence.
- ❖ One can create any number of objects for a class.

## *Syntax :*

**class** ClassName:

**""" documentation string """**

    variables: instance variables, static and local variables

    methods: instance methods,static methods,class methods

# PROGRAMMER-DEFINED TYPES-CLASS

- ✓ We will create a type called Point that represents a point in two-dimensional space.
- ✓ A class in Python can be created using a keyword class.
- ✓ Here, we are creating an empty class without any members by just using the keyword pass within it.

```
class Point:  
    pass  
print(Point)
```

**Output:**

```
<class '__main__.Point'>
```

- ✓ This class is at the top level while executing the program.
- ✓ In mathematical notation, points are often written in parentheses with a comma separating the coordinates.(x,y)
- ✓ There are several ways we might represent points in Python:
  - We could store the coordinates separately in two variables x and y.
  - We could store the coordinates as elements in a list or tuple.
  - We could create a new type to represent points as objects.

# CONTD...

## Class:

- ✓ A programmer-defined type is also called a **class**.
- ✓ A class definition looks like this: →

## Object :

- ✓ Physical existence of a class is nothing but object.
- ✓ We can create any number of objects for a class.
- ✓ Creating a new object is called **instantiation** and the object is an instance of the class. When you print an instance, Python tells you what class it belongs to and where it is stored in memory.

**Syntax:** referencevariable = classname()

The return value is a reference to a Point object, which we assign to blank.

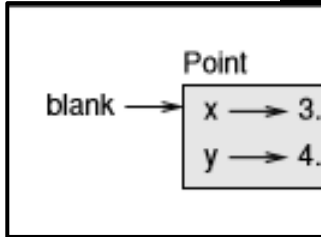
```
class Point:  
    """Represents a point in 2-D space."""
```

```
blank=Point()  
print(blank)
```

```
-----  
Output:  
<__main__.Point object at 0x7f17c7406b38>
```

# ATTRIBUTES (CLASS & INSTANCE ATTRIBUTES)

- ✓ An object can contain named elements known as **attributes**.
- ✓ One can access and assign values to these attributes using **dot** operator.
- ✓ A state diagram that shows an object and its attributes is called an **object diagram**.



The variable **blank** refers to a Point object, which contains two attributes. Each attribute refers to a floating-point number. You can read the value of an attribute using the same syntax:

**x, y represents class attributes**

<pre>class point:     x=0     y=0  blank = point() blank.x=3.0 blank.y=4.0 print(blank.x) print(blank.y)</pre>	<b>Output:</b> 3.0 4.0
--	------------------------------

**x, y represents instance attributes**

<pre>class point:     """ This is a class Point     representing a coordinate point"""  blank = point() blank.x=3.0 blank.y=4.0 print(blank.x) print(blank.y)</pre>	<b>Output:</b> 3.0 4.0
---	------------------------------

Class Variables	Instance Variables
Also called static Variables	Also called Object Level Variables.
Are defined inside the class	Are defined for individual objects. And available only for that instance /object.
The value of a variable is not varied from object to object, such type of variables we have to declare with in the class directly	The value of a variable is varied from object to object
Only one copy of static variable will be created and shared by all objects	For every object a separate copy of instance variables will be created.
<pre> class Point:     x=2     y=4  p1= Point()           #first object of the class print(p1.x, p1.y)     #Outputs  2 4 </pre>	<pre> class Point:     """ This is a class Point"""  p1= Point()           #first object of the class p1.x=10.0             #attributes for p1 p1.y=20.0 print(p1.x, p1.y)     #Outputs  10.0 20.0 </pre>

# DIFFERENT WAYS OF USING OBJECTS AND ACCESSING ITS ATTRIBUTES

There is no conflict between the variable x and the attribute x.

```
class Point:
```

```
    x=3.0
```

```
    y=4.9
```

```
blank = Point()
```

```
x = blank.x
```

```
print(x)
```

**Output: 3.0**

You can use dot notation as part of any expression

```
print('%f, %f' % (blank.x, blank.y))
```

**Output:**

(3.000000, 4.900000)

You can pass an instance as an argument to a function

```
def print_point(p):
```

```
    print('%f, %f' % (p.x, p.y))
```

```
class Point:
```

```
    x=y=0
```

```
blank = Point()
```

```
blank.x= 3.0
```

```
blank.y= 4.0
```

```
print_point(blank) #Output: (3.000000, 4.000000)
```

```
import math
```

```
distance = math.sqrt(blank.x**2 + blank.y**2)
```

```
print(distance)
```

**Output: 5.0**



# RECTANGLES

- ✓ It is possible to make an object of one class as an attribute to other class.
- ✓ To illustrate this, consider an example of creating a class called as Rectangle.

A rectangle can be created using any of the following data –

- By knowing width and height of a rectangle and one corner point (ideally, a bottom-left corner) in a coordinate system
- By knowing two opposite corner points

## Class Rectangle

### Attributes:

Height

Width

Corner( Which represents object)

**Write a class Rectangle containing numeric attributes width and height.**

**This class should contain another attribute corner which is an instance of another class Point**

**Implement following functions –**

- ✓ A function to print corner point as an ordered-pair
- ✓ A function find\_center() to compute center point of the rectangle
- ✓ A function resize() to modify the size of rectangle

# CONTD..

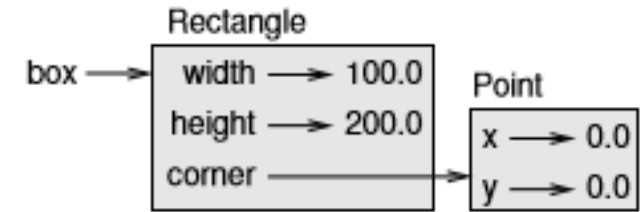


Figure 15.2: Object diagram.

**class Point:**

""" This is a class Point representing coordinate point """

**class Rectangle:**

""" This is a class Rectangle. Attributes: width, height and Corner Point """

```
box=Rectangle()           #create Rectangle object
box.corner=Point()         #define an attribute corner for box
box.width=100              #set attribute width to box
box.height=200             #set attribute height to box
box.corner.x=10            #corner itself has two attributes x & y
box.corner.y=20            #initialize x and y
```

The below function takes pointer object as a parameter and prints the values of x and y that belongs to instance Point class

```
def print_point(p):
    print('(%f, %f)' % (p.x, p.y))
```

# INSTANCES AS RETURN VALUES

- ✓ Functions can return instances(Objects).

```
def find_center(rect):  
    p=Point()  
    p.x = rect.corner.x + rect.width/2  
    p.y = rect.corner.y + rect.height/2  
    return p
```

10X20 (W,H)  
(x,y)= (1,1)

because x is half the width,  
and y is half the height plus  
x & y.

(x,y) = (6, 11)

- ✓ When a function is called, it takes box as a parameter to the function.
- ✓ `find_center()` function returns a point object and is assigned to center.

```
box= Rectangle( )  
center = find_center(box) #returns p.x & p.y to variable center  
print_point(center)
```

Output: (60,120)

# OBJECTS ARE MUTABLE

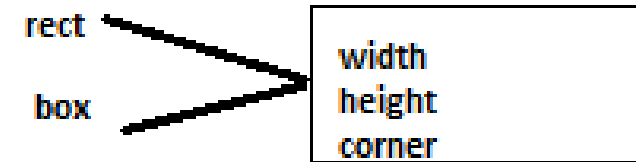
- ✓ You can change the state of an object by making an assignment to one of its attributes.
- ✓ For example, to change the size of a rectangle without changing its position, you can modify the values of width and height:

```
box.width = box.width + 50
```

```
box.height = box.height + 100
```

- ✓ You can also write functions that modify objects.

```
def grow_rectangle(rect, w, h):  
    rect.width += w  
    rect.height += h
```



- ✓ To call the above function:

```
grow_rectangle(box, 50, 100)
```

```
print(box.width, box.height)
```

**Inside the function, rect object is an alias for box, so when the function modifies rect, box object also changes.**

**Example:** Write a function named `move_rectangle` that takes a rectangle and two numbers named `dx` and `dy`. It should change the location of the rectangle by adding `dx` to the x coordinate of corner and adding `dy` to the y coordinate of corner.

```
def move_rectangle(rect, dx, dy):  
    rect.corner.x = rect.corner.x +dx  
    rect.corner.y = rect.corner.y +dy
```

To call a function

```
move_rectangle(box, 10, 20)
```

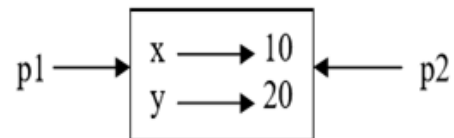
# COPYING

- ✓ An object will be aliased whenever an object is assigned to another object of same class. This may happen in following situations –
  - ❖ Direct object assignment (like p2=p1)
  - ❖ When an object is passed as an argument to a function
  - ❖ When an object is returned from a function

```
>>>p1==p2  
True
```

```
p1= Point()  
p2=p1  
print(p1)  
print(p2)  
Output:  
<__main__.Point object at 0x7f682f5060b8>  
<__main__.Point object at 0x7f682f5060b8>
```

- ✓ Observe that both **p1 and p2 objects have same physical memory.**
- ✓ It is clear now that the object p2 is an alias for p1. So, we can draw the object diagram as below



**It is difficult to track all the objects that have been aliased. Hence we will go for copying objects. Copying an object is often an alternative to aliasing.**

# CONTD..

The **copy** module contains a function called `copy( )` that can duplicate any object:

```
class Point:
    pass
p1=Point()
p1.x=10
p1.y=20

import copy          #import copy library
p2=copy.copy(p1)     #use the method copy()
print(p1)
print(p2)
print(p2.x,p2.y)
print(p1==p2)        #Outputs False
print(p1 is p2)      #Outputs False
```

**Output:**

<\_\_main\_\_.Point object at 0x7f93a2b49160>

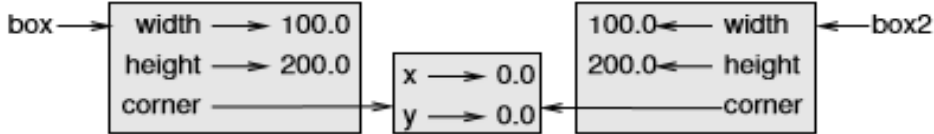
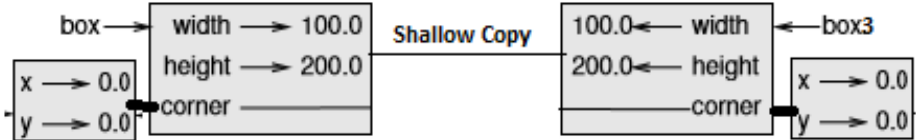
<\_\_main\_\_.Point object at 0x7f93a2b496d8>

10 20

False

False

# SHALLOW COPY & DEEP COPY

Shallow Copy	Deep Copy
Copies one object to another, but doesn't create a copy of nested objects.	A deep copy copies not only the object but also the objects it refers to, and the objects they refer to, &so on
	
<pre>import copy copy.copy(x)</pre>	<pre>import copy copy.deepcopy(x)</pre>
<pre>&gt;&gt;&gt;box2 = copy.copy(box) &gt;&gt;&gt;box2 is box False &gt;&gt;&gt;box2.corner is box.corner True</pre>	<pre>&gt;&gt;&gt; box3 = copy.deepcopy(box) &gt;&gt;&gt; box3 is box False &gt;&gt;&gt; box3.corner is box.corner False</pre>
The above code aliases for a nested objects	The above code will create a separate copy of nested objects



# ASSIGNMENT

- 1) Write a function that calculates the distance between points that takes two Points as arguments and returns the distance and also find the midpoint between two points.  
( Hint: Apply Euclidean Distance).
- 2) Create a student class with name ,USN and percentage as attributes. Create 2-4 student objects and define a function that finds highest percentage marks and display student details.
- 3) Write a definition for a class named Circle with attributes center and radius, where center is a Point object and radius is a number.
  - Instantiate a Circle object that represents a circle with its center at (150,100) and radius 75.
  - Write a function named point\_in\_circle that takes a Circle and a Point and returns True if the Point lies in or on the boundary of the circle.

**THANK  
YOU**

# **MODULE 5 – PART 1**

## **NETWORKED PROGRAMS**

**By,**

**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# INTRODUCTION

Here we will pretend to be a web browser and retrieve web pages using the Hyper Text Transport Protocol (HTTP). Then we will read through the web page data and parse it.

## (HTTP) Hypertext Transfer Protocol :

is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers

## Sockets:

A socket is a link between two applications that can communicate with one another (either locally on a single machine or remotely between two machines in separate locations).

Basically, sockets act as a communication link between two entities, i.e. a server and a client. A server will give out information being requested by a client.

# THE SOCKET MODULE

- In order to create a socket, we use the `socket.socket()` function, and the syntax is as simple as:

```
import socket  
s= socket.socket (socket_family, socket_type, protocol=0)
```

- Here is the description of the arguments:
  - socket\_family**: Represents the address (and protocol) family. It can be either `AF_UNIX` or `AF_INET`.
  - socket\_type**: Represents the socket type, and can be either `SOCK_STREAM` or `SOCK_DGRAM`.
  - protocol**: This is an optional argument, and it usually defaults to 0.
- After obtaining your socket object, we can then create a server or client as desired using the methods available in the socket module.

# THE WORLD'S SIMPLEST WEB BROWSER

- Python program that makes a connection to a web server and follows the rules of the HTTP protocol to requests a document and display what the server sends back.

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET https://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if len(data) < 1:
        break
    print(data.decode(),end="")

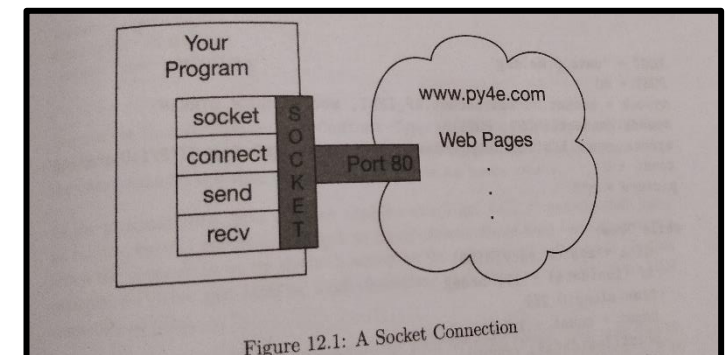
mysock.close()
```

## Output:

```
HTTP/1.0 200 OK
Date: Mon, 18 May 2020 08:47:41 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Sat, 13 May 2017 11:22:22 GMT
ETag: "a7-54f6609245537"
Accept-Ranges: bytes
Content-Length: 167
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Content-Type: text/plain
```

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

- The `AF_INET` argument indicates that you're requesting an Internet Protocol (IP) socket, specifically IPv4. The second argument is the transport protocol type `SOCK_STREAM` for TCP sockets.
- First the program makes a connection to port 80 on the server `www.py4e.com`. Since our program is playing the role of the “web browser”, the HTTP protocol says we must send the GET command followed by a blank line.
- Once we send that blank line, we write a loop that receives data in 512-character chunks from the socket and prints the data out until there is no more data to read (i.e., the `recv()` returns an empty string).
- The output starts with headers which the web server sends to describe the document. For example, the Content-Type header indicates that the document is a plain text document (`text/plain`). After the server sends us the headers, it adds a blank line to indicate the end of the headers, and then sends the actual data of the file `romeo.txt`.



# RETRIEVING AN IMAGE OVER HTTP

```
import socket
import time

HOST = 'data.pr4e.org'
PORT = 80
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect((HOST, PORT))
mysock.sendall(b'GET https://data.pr4e.org/cover3.jpg
HTTP/1.0\r\n\r\n')
count = 0
picture = b""

while True:
    data = mysock.recv(5120)
    if len(data) < 1: break
    #time.sleep(0.25)
    count = count + len(data)
    print(len(data), count)
    picture = picture + data

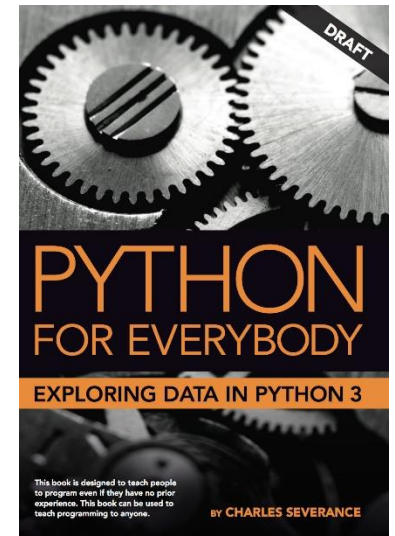
mysock.close()

# Look for the end of the header (2 CRLF)
pos = picture.find(b"\r\n\r\n")
print('Header length', pos)
print(picture[pos:].decode())

# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg", "wb+")
fhand.write(picture)
fhand.close()
```

Retrieve an image across using HTTP. Instead of copying the data to the screen as the program runs, we accumulate the data in a string, trim off the headers, and then save the image data to a file as follows:

```
2920 2920
1460 4380
1460 5840
1460 7300
...
1460 62780
1460 64240
2920 67160
1460 68620
1681 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:15:07 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```





- The **recv()** function receives data on a socket with descriptor socket and stores it in a buffer. The **recv()** call applies only to connected sockets. The socket descriptor- the pointer to the buffer that receives the data.
- `time.sleep()` it slow down our successive `recv()` calls. This way, we wait a quarter of a second after each call so that the server can “get ahead” of us and send more data to us before we call `recv()` again
- By uncommenting the call to `time.sleep()` in above program we will get the following output

```
1460 1460
5120 6580
5120 11700
...
5120 62900
5120 68020
2281 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:22:04 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```

Now other than the first and last calls to `recv()`, we now get 5120 characters each time we ask for new data.

# RETRIEVING WEB PAGES WITH URLLIB- URL HANDLING MODULE FOR PYTHON.

- It is used to fetch URLs (Uniform Resource Locators). It uses the urlopen function and is able to fetch URLs using a variety of different protocols
- urllib library simplifies the task of sending and receiving data over HTTP using the socket library.
- urllib treats a web page like a file- We simply indicate which web page we would like to retrieve. It also handles all of the HTTP protocol and header details.
- The equivalent code to read the romeo.txt file from the web using urllib is as follows:

```
import urllib.request  
  
fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')  
for line in fhand:  
    print(line.decode().strip())
```

## Output

```
But soft what light through yonder window breaks  
It is the east and Juliet is the sun  
Arise fair sun and kill the envious moon  
Who is already sick and pale with grief
```

- Once the web page has been opened with urllib.urlopen, we can treat it like a file and read through it using a for loop.

Example: write a program to retrieve the data for romeo.txt and compute the frequency of each word in the file as follows:

```
import urllib.request
counts = dict()
fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1
print(counts)
```

#### Output

```
{b'But': 1, b'soft': 1, b'what': 1, b'light': 1, b'through':
1, b'yonder': 1, b'window': 1, b'breaks': 1, b'It': 1,
b'is': 3, b'the': 3, b'east': 1, b'and': 3, b'Juliet': 1,
b'sun': 2, b'Arise': 1, b'fair': 1, b'kill': 1, b'envious': 1,
b'moon': 1, b'Who': 1, b'already': 1, b'sick': 1, b'pale':
1, b'with': 1, b'grief': 1}
```

# PARSING HTML USING REGULAR EXPRESSIONS

- Web scraping or web data extraction is data scraping used for extracting data from websites.
  - specific data is gathered and copied from the web, typically into a central local database or spreadsheet, for later retrieval or analysis.
- One simple way to parse HTML is to use regular expressions to repeatedly search for and extract substrings that match a particular pattern.
- Consider a simple web page:

```
<h1>The First Page</h1>  
<p> If you like, you can switch to the  
<a href="http://www.dr-chuck.com/page2.htm">  
Second Page</a>.  
</p>
```

- We can construct a well-formed regular expression to match and extract the link values from the above text as follows:

**href=**<http://.+.?>

The question mark in .+? indicate that the match should find smallest possible matching string. Also indicates that the match is to be done in a “non-greedy” fashion instead of a “greedy” fashion.

## CONTD...

- ❑ A **Non-Greedy** match tries to find the smallest possible matching string.
- ❑ A **Greedy** match tries to find the largest possible matching string.
- ❑ We add parentheses to our regular expression to indicate which part of our matched string we would like to extract

```
import urllib.request
import re
url = input('Enter - ')
html = urllib.request.urlopen(url).read()
links = re.findall(b'href="(http[s]?://.*?)"', html)
for link in links:
    print(link.decode())
```

### Output:

Enter - <http://www.py4inf.com/book.htm>

<http://www.greenteapress.com/thinkpython/thinkpython>  
<http://allendowney.com/> <http://www.py4inf.com/code>  
<http://www.lib.umich.edu/espresso-book-machine>  
<http://www.py4inf.com/py4inf-slides.zip>

- ❑ The `findall()` regular expression method will give us a list of all of the strings that match our regular expression, returning only the link text between the double quotes.

# PARSING HTML USING BEAUTIFULSOUP

There are a number of Python libraries which can help you parse HTML and extract data from the pages. Few of them are:

- Beautiful Soup
- Selenium
- Lxml
- Scrapy
- Requests

**BeautifulSoup:** Python library for parsing HTML documents and extracting data from HTML documents that compensates for most of the imperfections in the HTML that browsers generally ignore

BeautifulSoup tolerates highly flawed HTML and still lets you easily extract the data you need.

We will use urllib to read the page and then use BeautifulSoup to extract the href attributes from the anchor (a) tags.

```
import urllib.request
from bs4 import BeautifulSoup
url = input('Enter - ')
html = urllib.request.urlopen(url).read()

soup = BeautifulSoup(html)
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    print(tag.get('href', None))
```

The program prompts for a web address, then opens the web page, reads the data and passes the data to the BeautifulSoup parser, and then retrieves all of the anchor tags and prints out the href attribute for each tag.

#### Output1:

Enter - <http://www.dr-chuck.com/page1.htm> <http://www.dr-chuck.com/page2.htm>

#### Output2:

Enter - <http://www.py4inf.com/book.htm>  
<http://www.greenteapress.com/thinkpython/thinkpython.html>  
<http://allendowney.com/> <http://www.si502.com/>  
<http://www.lib.umich.edu/espresso-book-machine>  
<http://www.py4inf.com/code> <http://www.pythonlearn.com/>

#### Output3:

Enter the link- <https://www.facebook.com/careers/?ref=pf>  
[/privacy/explanation](https://www.facebook.com/privacy/explanation)  
[/policies/cookies/](https://www.facebook.com/policies/cookies/)  
<https://www.facebook.com/help/568137493302217>  
[/policies?ref=pf](https://www.facebook.com/policies?ref=pf)  
[/help/?ref=pf](https://www.facebook.com/help/?ref=pf)  
[/settings](https://www.facebook.com/settings)  
[/allactivity?privacy\\_source=activity\\_log\\_top\\_menu](https://www.facebook.com/allactivity?privacy_source=activity_log_top_menu)

You can use BeautifulSoup to pull out various parts of each tag as follows:

```
import urllib.request
from bs4 import BeautifulSoup
url = input('Enter - ')
html = urllib.request.urlopen(url).read()
soup = BeautifulSoup(html)
# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    # Look at the parts of a tag
    print 'TAG:',tag
    print 'URL:',tag.get('href', None)
    print 'Content:',tag.contents[0]
    print 'Attrs:',tag.attrs
```

**Output:**

Enter - <http://www.dr-chuck.com/page1.htm>

TAG: <a href="http://www.dr-chuck.com/page2.htm"> Second Page</a>

URL: <http://www.dr-chuck.com/page2.htm>

Content: [u'\nSecond Page']

Attrs: [(u'href', u'<http://www.dr-chuck.com/page2.htm>')]



# IMPORTANT QUESTIONS

- Demonstrate with python program how o retrieve image over http
- Demonstrate with python program how o retrieve webpages using urllib
- Explain socket functions and write a python code to parse html using regular expression.

**THANK  
YOU**

# **MODULE 5 – PART 2**

## **WEB SERVICES**

**By,**

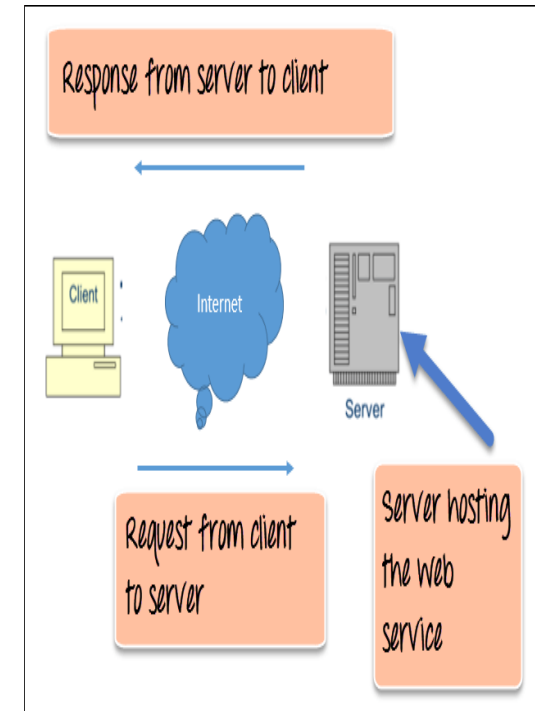
**Ravi Kumar B N**

**Assistant professor, Dept. of CSE**

**BMSIT & M**

# INTRODUCTION TO WEB SERVICES

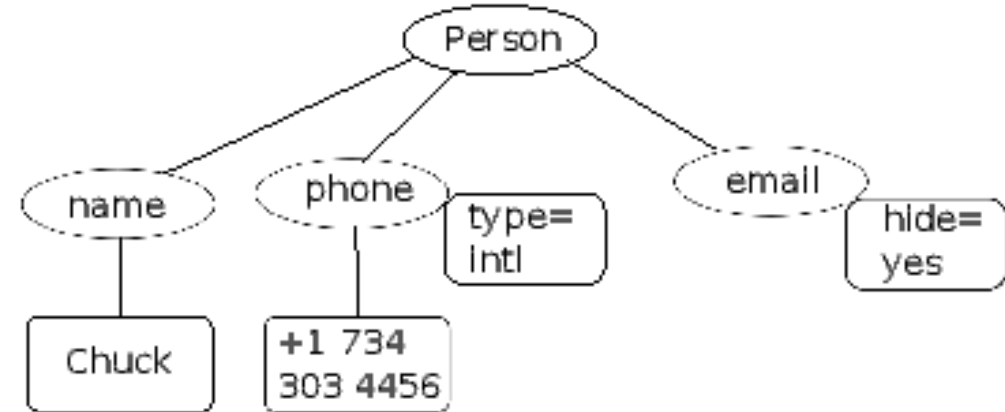
- ✓ Web services are web application components, which communicate using open protocols
- ✓ Web services can be published, found and used on the Web.
- ✓ It is a collection of protocols and standard formats to exchange data.
- ✓ HTTP and XML is the basis for Web services.
- ✓ Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
- ✓ Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction.
- ✓ There are mainly two types of web services.
  - ✓ **SOAP** (Simple Object Access Protocol) web services.
  - ✓ **RESTful** (Representational State Transfer) web services.
- ✓ SOAP is based on transferring XML data as SOAP Messages.
- ✓ REST is a way to access resources which lie in a particular environment. (Images, Audio, Video)
- ✓ There are two common formats that we use when exchanging data across the web.
  - XML & JSON(text-based data exchange format)



# XML - EXTENSIBLE MARKUP LANGUAGE

XML looks very similar to HTML, but XML is more structured than HTML. Here is a sample of an XML document:

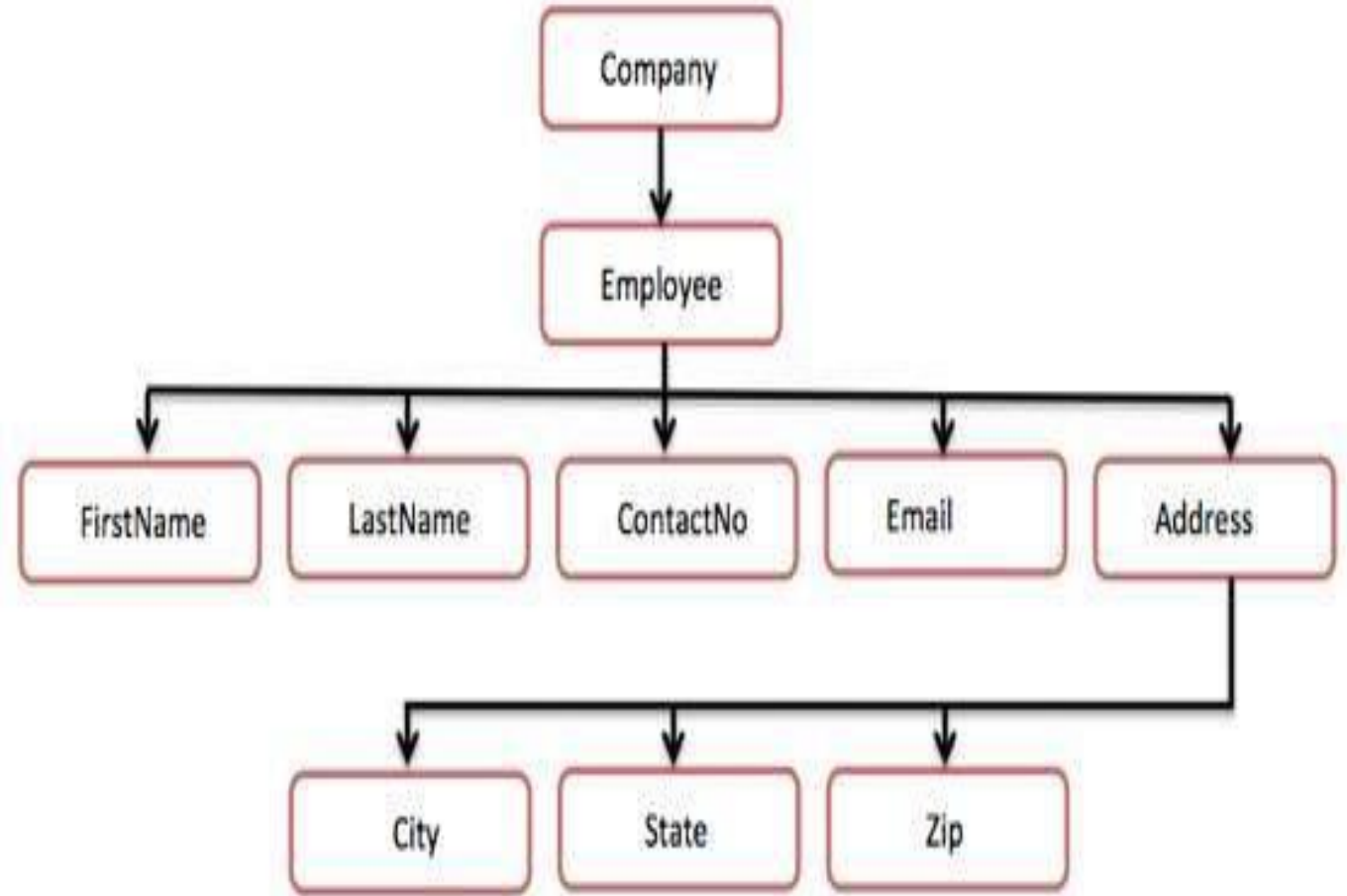
```
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes"/>
</person>
```



XML document as a tree structure where there is a top tag person and other tags such as phone are drawn as children of their parent nodes.

# EXAMPLE

```
<Company>
  <Employee>
    <FirstName>Tanmay</FirstName>
    <LastName>Patil</LastName>
    <ContactNo>1234567890</ContactNo>
    <Email>tanmaypatil@xyz.com</Email>
    <Address>
      <City>Bangalore</City>
      <State>Karnataka</State>
      <Zip>560212</Zip>
    </Address>
  </Employee>
</Company>
```



# PARSING XML

- XML parser is a software library or a package that provides interface for client applications to work with XML documents.
- It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers. The goal of a parser is to transform XML into a readable code.



- *fromstring* converts the string representation of the XML into a 'tree' of XML nodes. When the XML is in a tree, we have a series of methods which we can call to extract portions of data from the XML.
- The *find* function searches through the XML tree and retrieves a node that matches the specified tag. Each node can have some text, some attributes (like `hide`), and some “child” nodes. Each node can be the top of a tree of nodes.

# ELEMENTTREE

- **ElementTree** allows us to extract data from XML without worrying about the rules of XML syntax.
- **ElementTree** acts as a parser and provides a set of relevant methods to extract the data.

Hence, the programmer need not know the rules and the format of XML document syntax.

- Python has a built in library, **ElementTree**, that has functions to read and manipulate XMLs.
- The tree is a hierarchical structure of elements starting with root followed by other elements. Each element is created by using **Element()** function of this module. Each element is characterized by a tag and attrib attribute which is a dict object.



# SIMPLE APPLICATION THAT PARSES SOME XML AND EXTRACTS SOME DATA ELEMENTS FROM XML:

```
import xml.etree.ElementTree as ET

data = '''
<person>
  <name>Chuck</name>
  <phone type="intl">
    +1 734 303 4456
  </phone>
  <email hide="yes"/>
</person>'''
tree = ET.fromstring(data)
print ('Name:',tree.find('name').text)
print ( 'Attr:',tree.find('email').get('hide'))
```

**Output:**

**Name: Chuck**

**Attr: yes**

# LOOPING THROUGH NODES

- XML has multiple nodes and we need to write a loop to process all of the nodes. The `findall()` method retrieves a Python list of subtrees that represent the user structures in the XML tree.

```
import xml.etree.ElementTree as ET
input = '''
<stuff>

    <users>
        <user x="2">
            <id>001</id>

            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>

            <name>Brent</name>
        </user>
    </users>
</stuff>'''
```

```
stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```

## Output:

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

# JAVASCRIPT OBJECT NOTATION - JSON

The JSON format was inspired by the object and array format used in the JavaScript language.

The format of JSON is nearly identical to a combination of Python lists and dictionaries.

```
import json
data = '''
{
  "name" : "Chuck",
  "phone" : {
    "type" : "intl",
    "number" : "+1 734 303 4456"
  },
  "email" : {
    "hide" : "yes"
  }
}'''
```

```
info = json.loads(data)
print('Name:',info["name"])
print('Hide:',info["email"]["hide"])
```

**Output:**

**Name: Chuck**  
**Hide: yes**

**JSON Objects**

## CONTD..

- **In general, JSON structures are simpler than XML because JSON has fewer capabilities than XML.**
- **But JSON has the advantage that it maps directly to some combination of dictionaries and lists.**
- **And since nearly all programming languages have something equivalent to Python's dictionaries and lists, JSON is a very natural format to have two cooperating programs exchange data.**

# PARSING JSON

- We construct our JSON by nesting dictionaries (objects) and lists as needed.
- Consider an example for a list of users where each user is a set of key-value pairs.
- A built-in JSON library to parse the JSON and read through the data. `import json`
- `json.loads()`- Convert from JSON to Python and parse it by using the `json.loads()` method.
- In the example, `json.loads()` is a Python list which we traverse with a for loop, and each item within that list is a Python dictionary.
- After parsing is done, we can use index operator `[]` to extract the data

```
import json
input = '''
[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  },
  { "id" : "009",
    "x" : "7",
    "name" : "Chuck"
  }
]'''
info = json.loads(input)
print ('User count:', len(info))
for item in info:
    for item in info:
        print('Name', item['name'])
        print('Id', item['id'])
        print('Attribute', item['x'])
```

## **Output:**

**User count: 2**  
**Name Chuck**  
**Id 001**  
**Attribute 2**

**Name Chuck**  
**Id 009**  
**Attribute 7**

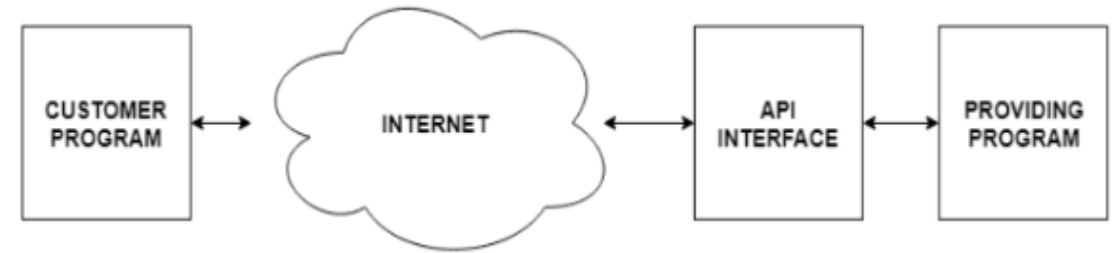
# SIMPLE PYTHON OBJECTS ARE TRANSLATED TO JSON

<b>Python</b>	<b>JSON</b>
dict	object
list, tuple	array
str	string
int, long, float	number
True	true
False	false
None	null

JSON	XML
JSON object has a type	XML data is typeless
JSON types: string, number, array, Boolean	All XML data should be string and best suited for exchanging document-style data.
Data is readily accessible as JSON objects	XML data needs to be parsed.
JSON has no display capabilities.	XML offers the capability to display data because it is a markup language.
JSON supports only text and number data type, dictionaries, lists, or other internal information	XML support various data types such as number, text, images, charts, graphs, etc. It also provides options for transferring the structure or format of the data with actual data.
Retrieving value is easy	Retrieving value is difficult
A fully automated way of deserializing/serializing JavaScript.	Developers have to write JavaScript code to serialize/de-serialize from XML
Native support for object.	The object has to be express by conventions - mostly missed use of attributes and elements.
It supports only UTF-8 encoding.	It supports various encoding.
It doesn't support comments.	It supports comments.
JSON files are easy to read as compared to XML.	XML documents are relatively more difficult to read and interpret.
It is less secured.	It is more secure than JSON.



# APPLICATION PROGRAMMING INTERFACES



## ❖ Application-to-Application contracts

❖ In general API is a software intermediary that is intended to be used as an interface by software components to allow two applications to communicate to each other.

❖ Modern APIs adhere to standards (typically HTTP and REST), that are developer-friendly, easily accessible and understood broadly.

❖ When we use an API, generally one program makes a set of services available for use by other applications and publishes the APIs (i.e., the “rules”) that must be followed to access the services provided by the program.

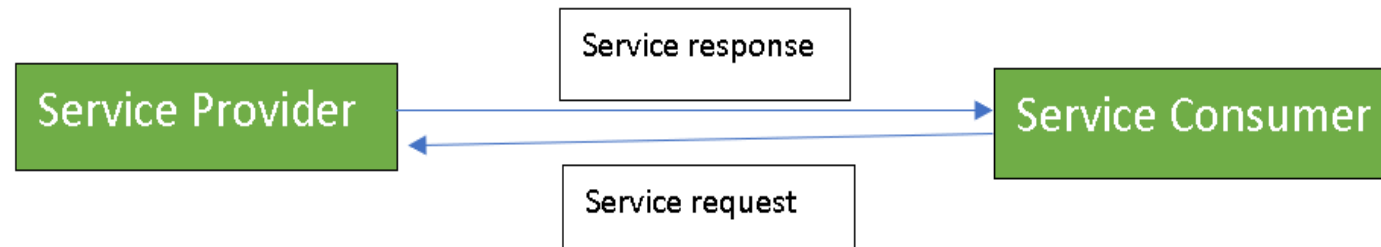
❖ Examples of APIs are [Google Maps API](#), Youtube API, Twitter, facebook, Amazon, E-commerce API etc.

❖ Server side web APIs have an interface that contains endpoints which lead to request-response message systems that are written in JSON or XML. Most of this is achieved using a HTTP web server. Client side web API's are used to extend the functionality of a web browser.

# SERVICE-ORIENTED ARCHITECTURE(SOA)

**(SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.**

**It is an architectural approach in which applications make use of services available in the network.**

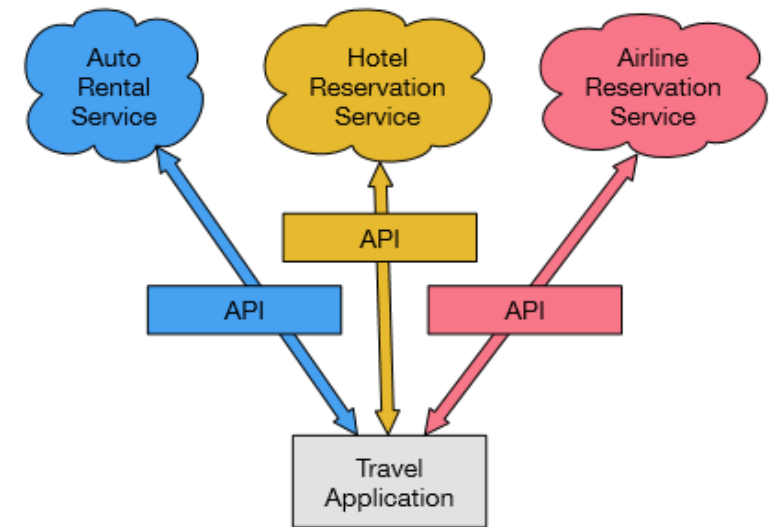


**When we begin to build our programs where the functionality of our program includes access to services provided by other programs, we call the approach a Service-Oriented Architecture(SOA).**

**A non-SOA approach is where the application is a single stand-alone application which contains all of the code necessary to implement the application**

# EXAMPLE OF SOA

We can go to a single web site and book air travel, hotels, and automobiles all from a single site. The data for hotels is not stored on the airline computers. Instead, the airline computers contact the services on the hotel computers and retrieve the hotel data and present it to the user. When the user agrees to make a hotel reservation using the airline site, the airline site uses another web service on the hotel systems to actually make the reservation. And when it comes time to charge your credit card for the whole transaction, still other computers become involved in the process.



# ADVANTAGES

- We always maintain only one copy of data (this is particularly important for things like hotel reservations where we do not want to over-commit)
- the owners of the data can set the rules about the use of their data.
- Creates reusable code
- Reduced costs
- Platform Independent
- Easy maintenance, scalable and reliable

**When an application makes a set of services in its API available over the web, then it is called as web services. data can set the rules about the use of their data.**

# GOOGLE GEOCODING WEB SERVICE

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.

Google has an excellent web service that allows us to make use of their large database of geographic information. Also allows developers to translate between human-readable location addresses to location coordinates.

For example “latitude and longitude of “1600 Amphitheatre Parkway, Mountain View, CA”, to their geocoding API and have Google return its best guess

The response sample:

```
"location" : {  
    "lat" : 37.4267861,  
    "lng" : -122.0806032  
},
```

## THE FOLLOWING IS A SIMPLE APPLICATION TO PROMPT THE USER FOR A SEARCH STRING, CALL THE GOOGLE GEOCODING API, AND EXTRACT INFORMATION FROM THE RETURNED JSON.

```
import urllib.request, urllib.parse, urllib.error
import json
serviceurl =
'http://maps.googleapis.com/maps/api/geocode/json?'
```

```
    address = input('Enter location: ')
    if len(address) < 1:
        break
    url = serviceurl + urllib.parse.urlencode( {'address':
address})
    print('Retrieving', url)
    uh = urllib.request.urlopen(url)
    data = uh.read().decode()
    print('Retrieved', len(data), 'characters')
    try:
        js = json.loads(data)
    except:
        js = None
```

```
if not js or 'status' not in js or js['status'] != 'OK':
    print('==== Failure To Retrieve ====')
    print(data)
```

```
print(json.dumps(js, indent=4))
lat = js["results"][0]["geometry"]["location"]["lat"] lng
= js["results"][0]["geometry"]["location"]["lng"]

print('lat', lat, 'lng', lng)

location = js['results'][0]['formatted_address']

print(location)
```

# CONTD..

The above program retrieves the search string and then encodes it. This encoded string along with Google API link is treated as a URL to fetch the data from the internet. The data retrieved from the internet will be now passed to JSON to put it in JSON object format.

If the input string (which must be an existing geographical location like (Avalahalli, BLSIT & M) cannot be located by Google API either due to bad internet or due to unknown location, it just display the message as „Failure to Retrieve“.

If Google successfully identifies the location, then we will dump that data in JSON object.

Then, using indexing on JSON (as JSON will be in the form of dictionary), we can retrieve the location address, longitude, latitude etc.

# SECURITY AND API USAGE

- It is quite common that you need some kind of “API key” to make use of a vendor’s API.
- Public APIs can be used by anyone without any problem.
- But, if the API is set up by some private vendor, then one must have API key to use that API.
- If API key is available, then it can be included as a part of POST method or as a parameter on the URL while calling API.
- Sometimes, vendor wants more security and expects the user to provide cryptographically signed messages using shared keys and secrets.
- The most common protocol used in the internet for signing requests is OAuth.
- Example:

As the Twitter API became increasingly valuable, Twitter went from an open and public API to an API that required the use of OAuth signatures on each API request.

But, there are still a number of convenient and free OAuth libraries so you can avoid writing an OAuth implementation from scratch by reading the specification.

These libraries are of varying complexity and have varying degrees of richness.

The OAuth web site has information about various OAuth libraries.



# PYTHON PROGRAM TO RETRIEVE A USER'S TWITTER FRIENDS, PARSE THE RETURNED JSON, AND EXTRACT SOME OF THE INFORMATION ABOUT THE FRIENDS

```
import urllib.request, urllib.parse, urllib.error
import twurl
import json
TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'
while True:
    print('')
    acct = input('Enter Twitter Account:')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL, {'screen_name': acct, 'count': '5'})

    print('Retrieving', url)
    connection = urllib.request.urlopen(url)
    data = connection.read().decode()
    headers = dict(connection.getheaders())
    print('Remaining', headers['x-rate-limit-remaining'])
    js = json.loads(data)

    print(json.dumps(js,
        indent=4))
    for u in js['users']:
        print(u['screen_name'])
        s = u['status']['text']
        print(' ', s[:50])
```

## Output:

```
{
  "next_cursor": 1444171224491980205,
  "users":
    [ { "id": 662433,
        "followers_count": 28725,
        "status": {
          "text": "@jazzychad I just bought one ._.",
          "created_at": "Fri Sep 20 08:36:34 +0000 2013",
          "retweeted": false,
        },
        "location": "San Francisco, California",
        "screen_name": "leahculver",
        "name": "Leah Culver",
      }
    ]
}
```

# IMPORTANT QUESTIONS

- Demonstrate with python program how o retrieve image over http
- Demonstrate with python program how o retrieve webpages using urllib
- Explain socket functions and write a python code to parse html using regular expression.

**THANK  
YOU**