Python Programming

LIET Notes
All 5 Units

Python Basics

Python is a high-level PL, with applications in numerous areas, including Web Programming, scripting, Scientific Computing, and AI.

It is very popular and used by organizations such as Google, NASA, the CIA and Disney.

→ Python is Interpreted

→ Your first Program

```
print("Hello World")
```

→ Hello World

→ You can download and install python from www.python.org

→ Python also includes IDLE, the integrated development Environment, which includes tools for writing and debugging entire programs

```
>>> Print("Hi")
```

to print Hi

```
>> print('ninini')
```

to print ninini

Simple operations

Python has the capability of carrying out calculations. Enter a calculation directly into the python console, and it will output the answer

```
>> 2+2
4
>> 5+4-3
6
```

AI - Artificial intelligence

Python also carries out multiplication and division using asterick to indicate multiplication (*) and forward slash to division.

→ use perentheesis to determine which operations are performed first

```
>> 2*(3+4)        >> 10/2
14                5.0 (decimal always)
```

>> which option is output by this code.

```
>>> (4+8)/2
6.0
```

a) 8 b) 6 c) 6.0

```
>>> (-7+2) * (4)
-20
```

→ Division by zero in python produces an Error, as no answer can be calculated

```
>> 11/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

(Error type, out line)

→ Dividing any integers produces a float

```
>> 8/2          >> 6*7.0          >> 1+2+3+4.0+5
4.0             42.0              => 15.0
```

→ Expmential **

```
>> 2**5          >> 9**(1/2)
32              3
```

```
>>(1+3)**2
16
```

Language fundamentals

Introduction

What is Python

Python is a powerful multi-purpose PL created by Guido Van Rossum in 1989.

→ Python is a cross-platform PL, i.e it runs on multiple platforms like windows, Mac OS, Linux. It's free and open source.

First Python Program

Language fundamentals

1. Introduction
2. Application Areas of Python
3. Features of Python
4. Limitations of Python
5. Flavours of Python
6. Python Variations
7. Identifiers
8. Reserved words
9. Datatype
   - 14 Datatypes
   - Typecasting

IDE - Integrated development Environment

Python Introduction

→ Python is Programming language. (Ex: desktop calculator)
(You can develop application)

Ex: Gmail Application

→ It is a High level language
(Programmer friendly, not Machine friend
low-level language features)
(memory management features)

Ex: a = 10 (test.py)
b = 20
c = 30 if a > b Else 40
print(c) ⇒ The value of C will be 40
(destroying objects, security)

→ to run the code > py test.y or Python test.y
> 40

→ It is a General Purpose PL.
(It is not specific to a domain)

→ Who developed Python — Guido Van Rossum.

Java came in 1995
Officially Released in 1997

while working in Python (1989)
NRI (National Research Institute)

Dob of Python in Netherland.
(Feb, 20th, 1991) — officially made available in market

→ Current Market Situation: Python
why now, discussion about

① Simple language and easy to understand
② Very less code | Concise code.
③ AI, ML, DL, NN, DS.
④ IOT.

Top Companies Using Python
Google, Yahoo, facebook, instagram, NASA, Dropbox
Quora, Netflix
Bittorrent, Youtube
Video chat

Quotient → use the floor division (//)

remainder → modulo (%)

```
>>> 20//6
3
>>> 7%2
1
>>> 7%(5//2)
1
```

Strings

If you want to use text in python, you have to use a string

String can be written in " " or ( )

```
>> "python is fun"
'python is fun'
```

5, 6, 7, 8, 9, 11, 12
15, 16, 24, 46, 5

\n → represents a new line.

Simple input, output functions

```
>>> print(1+1)
2
>>> print("Hello\n world")
Hello
world
```

→ what is o/p of the code

```
>>> print('print("print")')
print("print")
```

Input

To get input from the user in python, we can use input function

```
>>> s = input("Enter something please")
print(s)
```

↳ words

String operations

Concatenation : strings in python can be added, using a process called concatenation.

```
>> "spam" + 'Eggs'
spam Eggs

>> "2" + "2"
22
```

```
print("a" + "5")
a+b
```

We can't add one datatype with another if concatenated

```
1 + '2' + 3 + 4
```

1) which line of code produces an Error

→ 3+4
→ "one" + "2"
→ '5' + 6
→ "7" + 'Eight'

Unsupported operated types

multiplied

```
>>> print("spam" * 3)
spam spam spam

>>> 4 * '2'
'2222'

>> '17' * '87'
Error

print('spam' * 0)
```

no output

(2) print(3 * '7')

'777'

Type Conversions

In python, it impossible to complete certain operations due to the types involved.

For instance., You can't add two strings containing the numbers 2 and 3 to produce integer 5; as operation on strings will result in '23'

The solution to this type [illegible]

Comparision with others PLs. (Consice Code)

No Pre Requisites to learn Python.

Comparisions to other PLs.

Ex.1: Hello World to print Helloworld

C

```
#include <stdio.h>
main()
{
printf("Hello World");
}
```

⇒ Compile and Run.

Java

```
public class Test
{
public static void main(String[] args)
{
System.out.Println("Hello World");
}
}.
```

↳ Compile → Run
$ Javac test.java
$ Java test ↳ Hello world.

Python

```
print("Helloworld");
```

→ ; optional

. run → python test.py → or → py test.y ↳ Hello world

Ex 2: Sum of two Numbers

C

```
#include<stdio.h>
main()
{
int a, b;
a=20;
b=30;
printf("The sum is %d", (a+b));
}
```

Java

```
class test
{
PSVM(String[] args)
{
int a,b;
a=20;
b=30;
S.O.P("The Sum" + (a+b));
}}.
```

Python

type not specified

```
1. a=20
2. b=30
3. print("The sum", a+b)
```

we can combine into a single line

```
a,b = 20,30
```

Statically typed PL
C, C++, Java

Dynamically typed PL
python, Java Script

Java:

int a;

→ Error.

Python

1. a = 10
2. print (type(a)) : o/p → int class
3. a = 10.5 now onwards → float class
4. print (type(a)) → bool. class
5. a = "lordi"
6. print (type(a)) → string class:

Why the Name Python ⇒

→ Why Guido van Rossum selected Python

He is attracted with fun show

"The Complete Monty Python's Flying Circus" since 1969 to 1974

Ex: Coffee with Karan.

Java

James Gosling

Oak (Embedded System)

Java (Coffee from Indonesia)

Hadoop

Yellow elephant with cap

Doug Cutting

His son was playing with doll (toy)

→ Python is All Rounder.

C → Functional PL.

C++, Java → OOPs.

Perl, Shellscript → Scripting Language

While developing Python Guido borrowed

Functional PL features from C

Object Language features from C++

Scripting Language features from Perl, Shell Script

Modular PL features from Modula-3.

Python is ~~An~~ All Rounder.

Applications of Python.

(1) Desktop Applications (The Applications which are running on Single System)

Ex: desktop calculator

(2) Web Applications (Gmail, flask, facebook)

(3) Network Applications

(4) Games

(5) Data Analysis

(6) ML, DL, NN, AI, IOT

frameworks:- Tensorflow

Variables and Assignment

* Python is Case sensitive.

→ Rules for Variables name is same as, most other High-level languages.

→ Python is dynamically typed, meaning that no pre-declaration of a variable or its type is necessary.

→ = → Assignment Sign

→ The type (and value) are initialized on Assignment

```
>> Counter = 0
>> miles = 1000.0
>> name = 'Bob'
>> Counter = Counter + 1
>> Kilometers = 1.609 * miles
>> print '%f miles is same as %f km' % (miles, Kilometers)
1000.0000 miles is same as 1609.000 km.
```

Python-2 onwards

`n = n * 10` can be written as `n *= 10`

NOTE: Python doesn't support `n++` or `--n`

Numbers

Python supports 4 different numerical types

* int (Signed integers)
* long (long integers) (may be octal and hexadecimal)
* float (floating point real values)
* Complex (Complex Numbers)

In this Exam, You would use the int function

```
>>> "2" + "3"
'23'
```

```
>>> int("2") + int("3")
```

(1) What is the output of this code

```
>>> int("3" + "4")
```

→ 34 ✓

Another Exam of type Conversions is trueing use input (which is a string) to numbers to allow for the performance of calculations.

```
>>> float(input("Enter a Number") + float
(input("Enter Another number"))
```

Enter a Number: 40

Enter another Number: 2

42.0

(2) >>> what is the o/p of this code?

```
float('210' * int(input("Enter a Numbe')))
```

Enter a Number: 2

210210.0

Variables

A Variable allows you to store a value by assigning it to a name, which can be used to refer to the value later in the program.

→ To assign a Variable, use one Equal Sign.

```
x = 7
>>> print(x)
>>> print(x+3)
print(x)
```

What is the output of this code.

```
>>> spam = "Eggs"
print (spam * 3)
```

A) Eggs Eggs Eggs

B) Spam Spam Spam

Variables can be reassigned as many times as you want, in order to change their value.

In python, variables don't have specific types, so you can assign a string to a variable, and later assign an integer to the same variable

```
>>> x = 123.456
>>> print(x)
123.456
>>> x = "This is a string"
>>> print(x + "!")
This is a string!
```

NOTE: Python is case-sensitive PL.

The only characters that are allowed in variable name. The only characters that are allowed are letters, numbers and underscore. Also they can't start with numbers

```
>> this_is_a_Normal = 7
>> 123abc = 7
```

S.E → invalid 'Syntax'

2) Which of these is a valid variable name in Python?

A) A_Variable_Name ✓ B) a_variable_name

C) a variable name

→ Trying to reference a variable you haven't assigned to cause an Error.

→ You can use the del statement to remove a variable, which means the reference from the name to the value is deleted, and trying to use the variable causes an error. Deleted variables can be re-assigned to later as new

```
>>> foo = "a string"
>>> foo
```

```
>>> bar
```

name 'bar' is

Operators

Python has following Arithmetic operators

+ → Addition , - → Subtraction , * , /

% → modulus , Exponential → **

>> print -2 * 4 + 3 ** 2

↳ Answer will be 1.

priorities → ** → top , then

*, /, % → then ← Unary +, Unary -

+, and -

3 ** 2 → 9

-2 * 4 → -8 } ⇒ 9 - 8 ⇒ 1

Comparision operators provided by python

<, <=, >, >=, ==, != , <>

↓ ↓

Not Equal C-style Not Equal ABC/pascal style

Exm:

>> 2 < 4
1.

>> 2 == 4
0.

>> 2 > 4
0

>> 6.2 <= 6
0

>> 6.2 <= 6.201
1.

Conjunction operators:

and or not

>> (2 < 4) and (2 == 4)
0

>> (2 > 4) or (2 < 4)
1.

>> not (6.2 <= 6)
1.

>> 3 < 4 < 5

↓ python valid

>> (3 < 4) and (4 < 5)

Program Input and the raw_input( ) Built-in Function

The easiest way to obtain user inputs from the command line is with raw_input( ): built-in function

```
>> user = raw_input ("Enter login Name: ")
   Enter login name : root
>> print 'your login is : ', user
      Your login is : root
```

This Example was strictly for text input.

A Numeric string input (with Conversion to a real integer) form follows below.

```
>>> num = raw_input ('Now Enter a Number')
   Now Enter a number : 1024
>> print 'Doubling Your Number : %d'
   % (int(num) * 2)
   ↳ Doubling Your Number : 2048.
```

The int( ) function Converts the string into a number.

Comments

As with most scripting and Unix-shell languages, the hash pound (#) sign signals Comment begin right from the # and Continue till the End of line.

```
>> print 'Hello World'   # The first Comment
   Hello World.
```

```
>>> del foo
>>> foo
```

NameError : name 'foo' is not defined

→ You can also take the value of the variable from the user input

```
>>> foo = input("Enter a Number")
Enter a Number : 7
>>> print(foo)
7
```

What is the o/p of this code.

```
>> spam = 2
>> Eggs = 3
>> del spam
>> Eggs = 4
>> spam = 5
print (spam * Eggs)
20
```

In-place Operators

In-place operators allow you to write code like 'x = x + 3' more concisely, 'x += 3'

The same thing is possible with other operators such as -, *, / and % as well.

```
>> x = 2
>> print(x)
2
>> x += 3
>> print(x)
5
```

③ What is the o/p

```
>> x = 4
>>> x* = 3
>>> print(x)
12
```

→ These can be used for strings also.

```
>>> x = 'spam'
>>> print(x)
Spam
```

```
>>> x += 'Eggs'
>>> print(x)
Spam Eggs.
```

④ What is the result

```
>> x = "a"
```

→ print(x) = a a a

Using an Editor

So far, we've only used Python with the console entering and running one line of code at a time.

→ Actual programs are written in a file, and then executed by the Python Interpreter.

→ In IDLE, this can be done by creating a new file, entering some code, saving the file, and running it. This can be done either with the menu or with the keyboard short-cuts Ctrl-N, Ctrl-S and F5.

p1.py → F5

```
x = 7
x = x + 2
print(x)
```

→ 9

Quiz-1

① What is o/p of this codes

```
spam = "7"
spam = spam + "0"
eggs = int(spam) + 3
print(float(eggs))
```

→ 73.0

② >> word = input("Enter a word")
Enter a word : cheese.
>> print(word + 'shop')
cheeseshop

③
```
>> x = 5
>> y = x + 3
>> y = int(str(y) + "2")
>> print(y)
```
82

④ Declare a variable, add 5 to it and print its value

```
x = 4
x += 5
print(x)
```

```
>>> x = 3
>>> num = 17
print(num % x)
```

Control Structures

Boolean → There are two boolean values: True and False.

```
>> my_boolean = True
>> my_boolean
True
```

```
>>> 2 == 3
False
```

```
>> "hello" == "hello"
True
```

Comparision: Another Comparision operator, the not Equal operator (!=), Equals to true if the items being compared aren't Equal, false if they are.

```
>> 1 != 1
False
```

```
>> "Eleven" != Seven
True
```

```
>> 2 != 10
True
```

```
>> 7 > 5
True
```

```
>> 10 < 10
false
```

If Statements

If an Expression Evaluates to true, some stmts are carried out. Otherwise they aren't carried out.

```
If Expression:
    Stmts
```

NOTE: Python use indentation (white spaces at the beginning of the line) to delimit blocks of code. Other languages such as C, use curly braces to accomplish this, but in Python indentation is mandatory

) A Python Program to check whether a given number is odd or Even.

```
n = int(input("Enter a Number"))
if n%2==1:
    print("odd number")
else:
    print("Even Number")
```

Exm: (1)

positive

```
n=9
if n>0:
print("positive
```

→ Nested - if

To perform more complex checks, if statements can be nested, one inside the other. This is the way to see whether multiple conditions are satisfied.

for Exm: num = 12

```
if num>5:
    print("Bigger than 5")
    if num<=47:
        print("Between 5 and 47")
```

O/P

```
num=7
if num>3:
    print("3")
    if num<5:
        print('5')
if num==7
print(7)
```

→ 3

Extra Ex

```
x=10
y=20
if x>y:
print("if"
else:
print("else"
```

Else statement

An Else statement follows an if statement, and contains code that is called when the if statement evaluates to false

Exm: (1)

```
num=1
if num==5:
    print("Number is 5")
else:
    if num==1:
        print("Number is 1")
    else:
        if num==7
            print("Number is 7")
        else:
```

O/P Number is

② if - else - chain

```
if 1+1==2:
    if 2*2==8:
        print("if")
    else
        print("else")
```

O/P Else

elif Statement

The elif (short for Else if) statement is a shortcut to use when chaining if and else statements.

A series of if elif statements can have a final else block, which is called if none of the if or elif expressions is True.

A shorter option for an Else if statement is "elif"

```
num = 7
if num == 5:
    print("Number is 5")
elif num == 11:
    print("Number is 11")
elif num == 7:
    print("Number is 7")
else:
    print("Number is not 5, 11 or 7")
```

o/p Number is 7

① What is the result

```
if (1 == 1) and (2+2 > 3):
    print("true")
else:
    print("false")
```

o/p : true

fill in the blank

```
age = 15
money = 500
if age > 18 or money > 100:
    print("welcome")
```

Boolean not

```
if not True:
    print("1")
elif not (1+1 == 3):
    print("2")
else:
    print("3")
```

o/p 2

Operator Precedence

'==' has higher precedence than "or"

```
if 1+1*3 == 6:
    print("Yes")
else:
    print("No")
```

o/p ⇒ No

**, → Exponentiation
~, +, - (Complement, Unary +, -)
*, /, %, //
+, -
>>, <<
&
^
|
in, not in, is, is not, <, <=, >, >=, !=, ==
not
and
or
=, %=, /=, //=, -=, +=, *=, **=.

While loop

An if statement is run once if its condition evaluates to true, and never if it evaluates to false.

→ A while statement is similar, except that it can be run more than once. The statements inside it are repeatedly executed as long as condition remains true. Once it evaluates to false, the next section of the code is executed.

Below is a while loop containing a variable that counts up from 1 to 5.

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
print("finished")
```

```
1
2
3
4
5
finished
```

How many nos does this code print?

```
i = 3
while i >= 0:
    print(i)
    i = i - 1
```

→ 4 times

Infinite loop is a special kind of while loop. It never stops running. Its condition always remains true.

Eg:

```
while 1 == 1
    print("In the loop")
```

(Ctrl+C)

Fill in the blanks to create a loop that increments the variable by 2 and prints the even values.

```
x = 0
while x <= 20 :
    print x
    x += 2
```

break: To end a while loop prematurely, the break statement can be used → When encountered inside the loop, the break causes the loop

```
i = 0
while 1 ==
    print(i)
    i = i + 1
    if i >= 5:
```

Python Basics

What is Python

Python is an Uncomplicated and Robust PL that delivers both the power and complexity of traditional Compiled languages along with the ease of use of simpler scripting and Interpreted languages.

History of Python

→ Work on Python began in late 1989 by Guido Van Rossum, in Netherlands and Released for public distribution in 1991.

Before python he was working on an Interpreted language python, ~~He~~ named ABC. He was not satisfied with that He want Extra methods in that, from that onwards, the seeds of python were sown.

Features of Python

(1) High-level (2) Object-oriented (3) Scalable
(4) Extensible (5) Portable (6) Easy-to-learn
(7) Easy-to-maintain (8) Robust
(9) Effective as a Rapid Prototyping Tool
(10) A memory Manager (11) Interpreted

obtaining Python

www.python.org → Ommunity home page.
www.pythonlabs.com → Commercial home page

Installing and Running Python.

Compailing Python

→ Python is compared with Jawas as oops

Python → perl → as a scripting language

JPython → Jawa version of python
↳ Scripting nature.

TCl → python
↳ scripting language.

GUI version of python → Tkinter

LISP or Scheme → Python
↳ functional PL

Program Input / Output

```
>>> print ("Hello Lords")
    Hello Lords
```

The print statement is used to display output on to screen.

```
>>> mystring = 'Hello Lords'
>> print mystring
   Hello Lords
>> mystring
   Hello lords
```

Features of Python.

1. Simple and easy to learn
2. Freeware and Open Source.
3. High level PL.
4. Platform Independent
5. Portability
6. Dynamically Typed
7. Both Procedure and OOPs.
8. Interpreted
9. Extensible
10. Embedded
11. Extensible library.

(1) Simple and Easy to learn

English — Crores of words

Java — 53

Python — 33 → Reserved words.

Oracle Java.

Ternary operator:

```
x = (10 > 20) ? 30 : 40;
```

→ Concise Code

```
x = 30 if 10>20 else 40
```

*** Reading from a text file and printing to a Console.

```
print(open(abc.txt).read())
```

abc.txt

```
python is a easy
programming language
```

C#, .Net — Microsoft — Java paid version

(2) Free Ware and Open Source.

No vendors for python

PSF (Non-profit) foundation

→ But you can donate (python.org)

Open Source

Source code of python is open to Everyone you can customize

1.2 Python Objects

Python objects have the following 3 characteristics.

1) Identity 2) Type 3) Value.

What 3 Values are associated with all objects

Identity → Unique identifier that differentiates an object from all others.

An object identifier can be obtained by using id( ) built-in function.

Type → It indicates what kind of values an object can hold, what operations can be applied and what behavioral rules these objects are subject to.

use type( ) built-in function to reveal the e of a python object.

Value: Data item that is represented by an object

The most familiar object attributes are functions, lasses, files.

1.3 - Standard Types (Primitive types)

I. Numbers (4 - Seperate - Sub types

↳ 1. Regular or "plain" integer
2. Long integers
3. Floating point Real Numbers
4. Complex Number

II. String
III. List
IV. Tuple
V. Dictionary
VI. set

We refer standard types as Primitive types

1.4 Other - built in types

1. Type
2. None
3. File
4. function
5. Module
6. class
7. class Instance
8. Method

type - built in function. Syntax

type(object)

The type() built-in function takes object and returns its type.

```
>> type(4)          # int type.
<type 'int'>
>> type('Hello World!')   # string type
<type 'string'>
>>> type(type(4))   # type type.
<type 'type'>
```

None

Python has a special type known as the Null object. It has only one value, None. The type of None is also None.

In C → void → value NULL.

1.5 Internal Types

1) Code 2) Frame 3) Traceback 4) Slice 5) Ellipsis 6) Xrange

Code Objects :- These are executable pieces of python source that are byte-compiled, usually as return values from calling the compile() method.

Frame objects :- These objects represents Execution stack frames in python. Frame objects contain all the information the python interpreter needs to know during a runtime execution Environment.

Traceback objects :- When you make an Error in python, an Exception is raised. If Exceptions are not handled, the interpreter Exits with some diagnostic information similar to the output shown below:

```
Traceback (innermost last):
  File "<stdin>", line N? in ???
Error Name : Error Reason.
```

Slice objects : Slice objects are created using the python Extended slice Syntax.

The Syntax for multi-dimentional indexing is:

Sequence [start1 : End1, start2 : End2] or

using the Ellipsis sequence.

[... start1 : end1]

```
>> var = "lords"
>> var[::-1]
>> var[::-2]
```

sdro l
srt^2 ggp

Ellipsis objects: These are used for Extended Slice notation.

Xrange objects :- These are sibling of range.

1.6 Standard Type Operators

Standard type value Comparison operators

1) Object Value Comparison

Op	function
Expr1 < Expr2	Expr1 less than Expr2
Expr1 > Expr2	Expr1 Greatthan Expr2
Expr1 <= Expr2	less than or Equal
Expr1 >= Expr2	Greater than or Equal
Expr1 == Expr2	Equal
Expr1 != Expr2	→ Expr1 is not Equal to Expr2 (C-type)
removed in py3* Expr1 <> Expr2	→ Expr1 is not Equal to Expr2 (pascal-style)

2) Standard type object identity Comparison operato

Op:	function
obj1 is obj2	obj1 is the same object as obj2
obj1 is not obj2	obj1 is not the same object as obj2

```
>> a = [5, 'hat', -9.3]
>> b = a
>> a is b
True.
>> b = 2.5e-5
>> a is b
false
>> a is not b
True.
```

1.6.3 Boolean

Standard Type Boolean Operators

Operator	Function
not Expr	logical NOT for Expr (Negation)
Expr1 and Expr2	logical AND of Expr1 and Expr2 (Conjunction)
Expr1 or Expr2	logical OR of Expr1 and Expr2 (disjunction)

Ex:
```
>> x = 3.14
>> not (x < 5.0)
   false
>> y = -1024
>> (x < 5.0) or (y > 2.71)
   True
>> (x < 5.0) and (y > 2.71)
   false
>>> not (x is y)
   True
```

1.7 Standard Type Built-in function

Function	Operation
cmp(obj1, obj2)	Compares obj1 and obj2, returns integer i where. i<0 if obj1 < obj2, i>0 if obj1 > obj2, i==0 if obj1 == obj2
repr(obj)	Returns evalmatable string representation of obj
str(obj)	Returns printable string rep. of obj
type(obj)	Determines type of obj and return type object

1.8. Categorizing the Standard Types (6)

Data type	Storage Model	Update model	Access Model
Numbers	Scalar	Immutable	Direct
Strings	Scalar	Immutable	Sequence
Lists	Container	Mutable	Sequence
Tuples	Container	Immutable	Sequence
Dictionaries	Container	Mutable	Mapping

(Composites)

type which holds a single literal object called as atomic or scalar storage

Multiple objects are called as container storage

1.9. Un Supported Types

following Types are not Supported by Python

1) Char 2) pointer 3) int versus short versus long

4) Float vs double.

Python does not Support Single precision floating point. (C long double Equal to float).

Model Questions

What 3 Attributes are associated with all python objects ? Briefly describe each one.

Write standard types, and other built-in types in python.

Explain Internal types in Python

Explain diff. standard type operators in Python

5. Write the following
   a) Standard type built-in Function b) Categorizing the standard type
   c) Un supported types.

Scalar storage: A type which holds a single literal object is called scalar or atomic type, and those which can hold multiple objects are known as container storage. (Compound or Composite)

All Python's Containers types can hold objects of different types.

3 different models to categorize the standard types.

1. Storage Model (Categorize by how many objects can be stored in an object of this type.)
2. Update Model (Can objects be changed or not?)
3. Access Model (How to do we access our stored data?)
   (direct, Sequence (sequential) and mapping)
   ↓ single Element ↳ unordered. and accessed with a key.

Mutable objects:- Whose values can be changed (lists, dictionaries)

Immutable objects:- whose values cannot be changed.

chapter-2 Numbers

→ Introduction to Numbers

→ Integers
- Boolean
- Standard Integers
- Long Integers

→ Floating Point Real Numbers

→ Complex Numbers

→ Operators → Built-in functions

→ Related Modules.

chapter-3 Sequences

1. Strings
2. List
3. Tuples.

chapter-4 Mapping and set types

Mapping Types: Dictionaries

- Operators
- Built-in functions
- Built-in methods
- Dictionary Keys

Set types

- Operators
- Built-in functions
- Built-in methods

Introduction to Numbers

Numbers provide scalar storage and direct access. It is an immutable type, meaning that changing or updating its value results in a newly allocated object.

Number objects

a = 1

a = −999.9 ...

afloat = 3.1415...

acomplex = 1.23 + 4.56j

Update

a = a + 1

If you want to delete a reference to a number object use `del` statement

>> a = 3

>> del a

>> print(a)

Name Error.

Integers
- Boolean
- Standard (Regular or plain) Integers
- Long Integers

Boolean: Objects of boolean type have two possible values ① True and ② False

Standard Integers:

a) Decimal b) Octal c) Hexadecimal

d) Binary

Apart from Base 10, remaining have leading indicator that determines base.

0O 101 → octal

84 → decimal

−237 → decimal

0b 1101 → for Binary

0X FFF7 → Hexadecimal

Long Integers :- (Greater than 2^{32})

This type is not there from python 2.2 onwards.

long Numbers previously represented by trailing L or 'l'

99999999L

Double Precision Floating point Real Number.

Floating point values are denoted by a decimal point (.) in the appropriate place and an optional "e" suffix representing Scientific Notation.

4.3e25 , 1.6 , 1.6e-2

Complex Number $i^2 = -1$

i.e Real + imaginary ~~part~~ Components j

64.575+3j

Complex number Attributes

Attribute	Description
num.real( )	Real Component of Complex number nu
num.img( )	Imaginary " " " " "
num.conjugate( )	Returns complex Conjugate of nu

```
>> a = 6.+3.4j
>> a.real( )
6
>> a.img( )
3.4
>> a.conjugate( )
(6+3.4j)
```

2. Operators

Numeric type supports a wide variety of operators, ranging from standard type of operators to operators created specifically for numbers

2.1. Mixed mode operations.

If Explicit coercion is desired, python does provide the coerce() built-in function.

Exm: In order to add the numbers (one integer, one float), both need to be converted to the same type. Since float is the superset, the integer is coerced to a float before the operation happens, leaving the result as a float

```
>> 1+4.5
5.5 ✓
```

2.2 Standard type Operators (Arithmetic)

Exm:

```
>> 5.2 == 5.2
True

>> -719 >= 833
False

>> .2<5<9   # same as (2<5) and (5<9)
True

>> 77.>66==66

>>> (-1<1) or (1<-1)
True
```

2.3 Numeric type (Arithmetic) Operators

Arithmetic Operation	Function
Expr1 ** Expr2	Expr1 raised to the power of Expr2
+Expr	(Unary) Expr sign unchanged
-Expr	(Unary) negation of Expr
Expr1 * Expr2	Expr1 times Expr2
Expr1 / Expr2	Expr1 divided by Expr2 (Classical or True division)
Expr1 // Expr2	" " " " (Floor division only)
Expr1 % Expr2	Expr1 modulo Expr2
Expr1 + Expr2	" plus "
Expr1 - Expr2	" minus "

```
>>> -442-77
-519
>>> 4**3
64
>>> 4.2**3.2
98.71831
>>> 8/3
2
>>> 8.0/3.0
2.6666.7
>>> 8%3
2
```

2.4 *Bit operators (Integers - only)

Bitwise Operator	Function
~num	(Unary) Invert the bits of num
num1 << num2	num1 left shifted by num2 bits
num1 >> num2	" Right shifted " "
num1 & num2	num1 bitwise AND num2
num1 ^ num2	num1 bitwise XOR with num2
num1 | num2	num1 bitwise OR with num2

```
>> 4&5
4
>>45<<1
90
```

```
>>60>>2  0100
&15      0101
         0100
>>30145
51
```

Built-in factory functions (II)

3.1 Standard type functions

(1) cmp( ) (2) str( ) (3) type( )

```
>> cmp(-6,2)
-1
>> cmp(0xFF, 255)
0
>> str(0xFF)
'255'
>> type(0xFF)
<type 'int'>
```

3.2 Numeric type functions

Class	Operation
bool(obj)	Returns the boolean value of obj
int(obj, base=10)	Returns integer representation of obj
long(obj)	" long " "
float(obj)	" floating point " "
complex(str)	" complex number " " str

```
>> complex(4) => 4+0j
>> float(4) => 4.0
>> int(4.22) => 4
```

3.3 Numeric type factory functions (Built-in Operational functions)

Function	Operation
abs(num)	→ Returns the absolute value of num
coerce(num1, num2)	→ Converts num1 and num2 to the same type and returns the converted pair as tuple
divmod(n1, n2)	→ returns (num1/n2, n1%n2) as a tuple
pow(n1, num2, mod)	→ Raises n1 to n2 power, quantity modulo mod if provided.
round(flt, ndig=0)	→ (floats only) takes a float and rounds it to ndig digits.

3.4 Integer-only functions

Python supports few functions, specific only to integers (plain and long)

2 types
1. Base Representation
2. ASCII Conversion

Integer Type Built-in functions

Function	Operation
hex(num)	Converts num to hexadecimal and returns a string
oct(num)	... to octal ...
chr(num)	Takes ASCII value num and returns ASCII character as string
ord(chr)	Takes ASCII chr and returns corresponding Ordinal ASCII value

Ex:
```
>> hex(255)
'0xff'
>> oct(255)
'0377'
>> ord('a')
97
>> ord('A')
65
>> chr(97)
'a'
```

4. Related Modules

Numeric type Related Modules

Module	Contents
decimal	→ Decimal floating point class decimal
array	→ Efficient arrays of Numeric values
math / cmath	→ Standard C library mathematical functions.
operator	→ Numeric Operator available as functions Calls operator.sub(m, n) ⇒ m-n
random	→ Various pseudo-Random number gener

Dictionary

- Dictionary
- Creating Dictionary
- Accessing components
- Modifying Components

Dictionary: It is an Example of Hash table data structure.

→ Works like key-value pair, where the keys are mapped to values.

→ Dictionaries are enclosed in, curly braces.

→ Dictionaries are mutable.

Creating a dictionary

```
fuel_type = {"petrol": 1, "diesel": 2, "CNG": 3}
```

keys are immutable. Value can be any Python object

To print the dictionary

```
print(fuel_type)
{'petrol': 1, 'diesel': 2, 'CNG': 3}
```

Accessing a Component of dictionary

```
print(fuel_type['petrol'])
```

O/P 1

To access keys

dictionary_name.keys()

~~print~~ fueltype.keys()

dict_keys ['petrol', 'diesel', 'CNG']

to access values from dictionary

fueltype.values()

dictionary_name.values()

dict_values [1, 2, 3]

To access both keys and values simultaniously from dictionary fuel-type.

dictionary_name.items()

fuel-type.items()

dict_items [('petrol': 1), ('Diesel' : 2), ('CNG' : 3)])

- It Returns Elements in a list format with (key, value) tuple pairs

Modifying a dictionary

Add a new key value pair

Syntax: dictionary-name[key] = value

fuel-type['Electric'] = 4

print(fuel-type)

{'petrol' : 1, 'diesel : 2, 'CNG':3, 'Electric': 4}

Update() BIF

dictionary-name = update({key: 'value'})

fuel-type.update({'Electric': 4})

Modify

fuel-type['CNG'] = 5

print(fuel-type)

'CNG': 5

Remove components from dictionary using del()

del dictionary-name[key]

Dropping the key petrol

del fuel-type['petrol']

print(fuel-type())

clear() → removes the all the key value pairs

dictionary-name.clear()

To access both keys and values simultaniously from dictionary fuel-type.

dictionary-name.items()

fuel-type.items()

dict-items [('petrol': 1), ('Diesel': 2), ('CNG': 3)])

- It Returns Elements in a list format with (key, value) tuple pairs

Modifying a dictionary

Add a new key value pair

Syntax: dictionary-name[key] = value

fuel-type['Electric'] = 4

print(fuel-type)

{'petrol': 1, 'deisel': 2, 'CNG': 3, 'Electric': 4}

update() BIF

dictionary-name = update({key: 'value'})

fuel-type.update({'Electric': 4})

modify

fuel-type['CNG'] = 5

print(fuel-type)

'CNG': 5

Remove components from dictionary using del()

del dictionary-name[key]

Dropping the key petrol

print(fuel-type())

del fuel-type['petrol']

clear() → removes the all the key value pairs

dictionary-name.clear()

Set

If we want to represent a group of unique values as single entity then we should go for set.

Duplicates are not allowed

Index and slicing are not allowed.

→ mutable.

→ Represents data in curly braces

Creation of set objects

s = {10, 20, 30, 40}

print(s) → {40, 10, 20, 30}

print(type(s)) → <class set>

functions of set

1) add(x)

s = {10, 20, 30}

s.add {40}

print(s)

{10, 20, 30, 40}

2) copy()

3) pop() → remove a random Element

4) remove() → removes a specified Element

5) clear() → remove all elements

Mathematical Operations on Sets

1) Union - x.union(y) or x | y → return all Elements present in both set

x = {10, 20, 30, 40}

y = {30, 40, 50, 60}

{10, 20, 30, 40, 50, 60}

2) Intersection

x.intersection(y) or x & y

print(x & y) → {30, 40}

3) difference() - x.difference(y) or x - y → returns Element present in x but in y

print(x-y) = {10, 20}

print(y-x) = {50, 60}

4) Symmetric_difference() returns Elements present in either x or y not both

x^y = {10, 50, 20, 60}

Dictionary and Mapping

If we want to represent a group of objects as key-value pairs then we should go for Dictionary.

→ Dictionaries are mutable.

→ indexing and slicing concepts not available.

Dictionaries are sole mapping type in python. Mapping objects have a one to many correspondance between Hashable values (keys) and the objects they represent (values)

→ The syntax of a dictionary Entry is key : value

→ Dictionary Entries are Enclosed in { } braces.

creating and Accessing dictionaries

d = { } or d=dict{ }

d[100] = "durga"

d[200] = "ravi"

d[300] = "shiva"

print(d) → { 100: 'durga', 200: 'ravi', 300: shiva }

print[d[400] → key Error :400

Updating dictionaries

d[key] = value.

d[400] = "pavan"

d[200] = "lords"

print[d] = { 100: 'durga', 200: 'lords', 300: 'shiva', 400: pavan }

deling a key

del d[100] → { 200: lords, 300: shiva, 400: pavan }

del d

=) delete total dictionary, we can't access after this

Lists

Lists provide sequential storage through an index offset and access to single or consecutive elements through slices.

→ Lists are mutable objects

→ Lists can contain different type of objects. [container]

→ Lists can be grown and shrunk.

How to create and Assign lists: lists are delimited by surrounding square brackets [ ]

```
lords = [123, 'abc', 4.56, ['inner', 'list'], 7+9j]
print(lords)
[123, 'abc', 4.56, ['inner', 'list'], 7-9j]
```

alist = [ ] → Empty list

How to Access Values in Lists

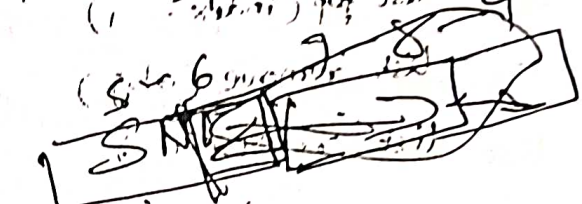
Access

```
lords[0]
123
```

```
lords[1:4]
['abc', 4.56, ['inner', 'list']]
```

```
lords[:3]
[123, 'abc', 4.56]
```

```
lords[3][1]
'list'
```



Updating a list

```
lords[1] = "welcome"
print(lords) => [123, "welcome", 4.56, ['inner', list], 7-9j]
```

Remove a list or list Elements

```
del lords[1]
print(lords)
[123, 4.56, ['inner', 'list'], 7-9j]
```

del lords → total list will be remov

List functions

len( ) max( ) min( )

sum

List Type Built-in Methods

Method	Operation
list.append(obj)	Adds obj to the end of list
list.count(obj)	Returns count of how many times obj occurs in list
list.extend(seq)	Appends contents of seq to list
list.insert(index, obj)	Insert obj into list at offset index
list.pop(index = -1)	Removes and returns obj at given or
list.remove(obj)	Remove obj from list
list.reverse( )	Reverse objects of list in place

Tuples

→ These are containers type → Tuples uses parentheses ( )

→ Immutable → We can't add elements

→ Duplicates are allowed

Ex: t = (10, 20, 10, 10, 20)

→ Read only version of list

print(t.index(10)) → 0

print(t.index(30)) → Value Error

sorted

t = (40, 10, 30, 20)

print(sorted(t)) ⇒ (10, 20, 30, 40)

(40, 30, 20, 10)

Sequences

1. Strings
2. List
3. Tuples

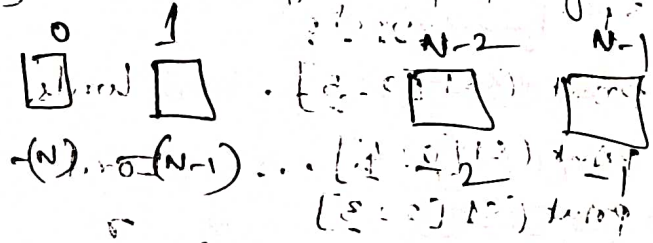
Sequences: The family of python type whose items are ordered sequentially and accessed via index offsets into its ct of Elements. This group is known as Sequences.

Sequences includes following types

1. Strings 2. Lists 3. Tuples

quence types all share the same access model : ordered et with sequentially indexed offsets to goto each Element.

Sequences:



String: A String represents a group of characters.

Concatenation operator (+) on string: We can concatenate two strings by using + operator

Exm: s_1 = "hello guys" s_2 = "Good morning"

print(s_1+s_2) ⇒ hello guys Good morning

s_3 = s_1 + s_2

print(s_3) ⇒ hello guys Good morning

Repetition (*) operator: It is used to repeat contents of a string multiple times:

s_1 = hello guys

print(s_1 * 5) ⇒ hello guys printed 5 times

Indexing: It is a process of accessing a character from a string based on index position.

s_1 = "0 1 2 3
Good Morning Guys"
-4 -3 -2 -1

print(s_1[3]) → d
print(s_1[5]) → M
print(s_1[-1]) → s.
print(s_1[20]) → Index Error

s_1 = "hello"
→ len(s_1)
5

Slicing:- It is a process of accessing a group of characters from a string

Syntax: data [starting index : Ending index]
It used to accessing a group of characters beginning from the specified starting index to Ending index − 1

s_1 = "Lords"
print(s_1[0:5]) Lords
print(s_1[0:4]) Lord
print(s_1[2:3]) r

Membership operator (in and not in): Using these operators we can check whether a particular data available or not in a given string.

Ex: msg = Good Morning Guys"
s_1 = "hello"
s_2 = "hai"
print(s_1 in msg) → True.
print(s_2 in msg) → False
print(s_2 not in msg) → True.

Python strings are a literal or a scalar type, a single value, not like containers
Strings are immutable
Strings are accessed sequentially via slicing

Unit-II

FILES, Exceptions, Modules

UNIT - 2

FILES: File objects, File Built-in Functions Built-in methods. File Built-in Attributes Standard files, Command line Arguments File system, File Execution, Persistent Storage modules, Related Modules.

Exceptions: Exceptions in python, Detecting and handling Exceptions, Context Management, Exceptions as strings, Raising Exceptions, Assertions, Standard Exceptions. Creating Exceptions Why Exceptions (now)? Why Exceptions at all? Exceptions and Sys modules, Related Modules

Modules: Modules and files, Name Spaces, Importing modules, Importing module Attributes, Module BIF, Packages, other packages of modules.

File objects: These can be used to access not only normal disk files, but also any other type of "file" that uses that abstraction.

① Text files ② Binary files → images, video, Audio.

File - Built-in Functions:

open() and file()

The open() BIF returns a file object on a successful opening of the file or Else returns an Error. when a failure occurs python raises an IOError Exception.

file_object = open(file_name, access_mode='r', buffering=-1)

file or full pathname

r, w, a

buffering is used to indicate the type of the buffering that should be performed when accessing a file.

0 → no buffering
1 → line Buffering
>1 → buffered I/O with the given value as the buffer size.

fp = open ('test', w)

file(·) : factory function

Both file() and open() do exactly same thing.

file() best used when you want to show that you are dealing with file objects. i.e if instance (f, file) ⟹ Universal new line UNS support

Access modes for files

file mode	operation
r	open for read
rU or U	open for read with Universal newline support
w	write
a	append (always work from EOF)
r+	read and write (previous data in the file, will not be deleted)
w+	read and write (Override existing data)
a+	read and write
rb	open for Binary read
wb	write
ab	append
rb+	binary r-w
wb+	r-w
ab+	r-w

File Built-in Methods (2)

File Attributes

Once we opened a file and we get a file object, we get various details related to that file by using its properties.

name → name of the file opened.
mode → mode in which file opened.
closed → returns boolean value. indicates the file
readable → returns boolean value whether readable or not
writable → " " " writable

file attributes

```
f = open("abc.txt", 'w')
print("Filename:", f.name)
print("File mode", f.mode)
print("Is file readable", f.readable())
print("Is file writable", f.writable())
print("Is file closed", f.closed)

f.close()
print("Is file closed", f.closed)
```

Output

```
File name : abc.txt
File mode : w
Is file readable : False
Is file writable : True
Is File closed : False
Is file closed : True
```

---

* File methods come in 4 categories.

1) Input 2) Output 3) movement within a file (Intra-file motions)
4) miscellaneous.

Standard Files (1) stdin (2) stdout (3) stdErr

3 types (1) Standard input (Keyboard)
(2) Standard output (Buffered output to the monitor to display)
(3) Standard Error (Un Buffered o/p to the screen)

Python takes these from sys module.

sys.stdin, sys.stdout, and sys.stderr

print → sys.stdout

raw_input → sys.stdin

Command line Arguments

Command line Arguments are those arguments given to the program in addition to the script on invocation.

The sys module provide access to any command-line Arguments via sys.argv.

argv variable contains an array of strings consisting of each argument from command line while the argc variable contains the no. of arguments entered.

In python

argc → no. of items in the sys.argv.list

len(sys.argv)

sys.argv[0] → program name

small program: argv.py (filename)

```
import sys
print 'you Entered', len(sys.argv), 'arguments...'
print ('they were:', str(sys.argv))
```

$ argv.py 76 tales 85 hawk

You Entered 5 arguments

they were: ['argv.py', '76', 'tales', '85', 'hawk']

Python has 2 modules to help process Command line arguments ① getopt ② optparse

File Built-in Methods

File Built-in Methods come in 4 different categories

1) Input 2) Output 3) Movement with in a file (Intra-file motion)

4) Miscellaneous.

Input : To read the data from a file, we have following 4 methods

1) read( ) :- This method will read the total data from the file.
2) readline( ) : will read only one line from the file
3) readlines( ) : Will read all the lines from the file into a list
4) read(n) : This method will read n characters from the file.

python.txt

```
welcome to
Python Programming
```

prog:1 : To read Entire file

```
f = open("python.txt", "r")
text = f.read( )
print(text)
f.close( )
```

prog:2 : To Read one line from a file

```
f = open("python.txt", "r")
text = f.readline( )
print(text)
f.close( )
```

prog:3 To Read all lines from a file

```
f = open("python.txt", "r")
list = f.readlines( )
print(list)
f.close( )
```

To read 10 characters from a file

```
f = open("python.txt", "r")
t = f.read(10)
print(t)
f.close
```

2) Output : To write the data into a file we have 2 methods

① write( ) : Used to write a string into a file

② writelines( ) used to write multiple lines into a file.

Prog 1: To write the contents into a file : (using write( ))

```
f = open("python.txt", "w")
f.write("Welcome to")
f.write("python Programming")
f.close()
print("data is stored into the file")
```

Using .writelines( )

Prog: 2

```
f = open("abc.txt", "w")
data = ["java\n", "python\n", "testing\n", "oracle\n"]
f.writelines(data)
f.close
print("data stored into the file.
```

3. Intra file Motions

seek( ) :- This method will specify the current location of the cursor

tell( ) :- This method will place the cursor in the specified position

Syntax :

f.seek(offset, whence → from where

offset in bytes given along with a relative offset

where attributes 0 → from beginning of the file (default)

1 → from the Current position

2 → from the End of the file

To move the file position

```
data = "All students are stupids"
f=open("abc.txt","w")
f.write(data)
```

```
with open("abc.txt","r+") as f:
    f=open("abc.txt","w")
    =f.read()
```

space →

```
    text = f.read()
    print(text)
    print("The current cursor position", f.tell())
    f.seek(17)
    print("The current cursor position", f.tell())
    f.write("GEMS!!!")
    f.seek(0)
    text = f.read()
    print("Data After modification")
    print(text)
```

O/P

All students are Stupids

The current cursor position : 24

The current cursor position : 17

Data After modification :

All students are GEMS!!!

4. Miscellaneous

close()

flush()

When we are writing data into the file and if we do not close the file, the content will not be placed into the file. If we want to place the content without closing, then we need to take the help of flush().

flush → It will clear the buffer and place the content into the file.

To clean the buffer

```
f = open("abcd.txt", "w")
f.write("Hello friend")
f.write("Good morning")
f.write("This is file handling concept in python")
f.flush()
print("data is stored into the file")
```

File System

:cess to your file system mostly through the python os odule. This serve as an interface to your os facilities and sevices from python.

Two modules in python file system

(1) os module (2) os.path module.

os module file/Directory Access functions

mkfifo/mknod	create a named pipe file system node.
remove	Delete file
symlink()	Create symbolic link
utime	update time stamp.
tories	
chdir()	Change working directory
listdir()	list files in directory
getcwd()	
makedirs()	
mkdir()	
access()	
chmod()	
chown	
umask	
open, dup, dup2()	

os.path module pathname Access functions

basename()	Remove directory path and return leaf name.
dirname()	Remove leaf name,
join()	join separate components
Status	
getatime()	Return last file access time.
getctime()	Return file creation time.
getmtime()	Return last file modification time
getsize()	Return file size in bytes
Inquiry	
Exists()	→ does pathname exist
isabs(), isdir(), isfile()	
islink(), ismount(), samefile()	

Persistent Storage Modules (5)

. It may be somewhat frustrating being required to Enter the same data repeatedly. So its better to have persistent storage, or a way archive your data so that you may access them at later time instead of re-Enter all that information.

→ The majority of the persistent storage modules deals with storing strings of data, but there are ways to archive python objects as well.

Pickle and Marshal Modules

Pickling is the process whereby objects more complex than primitive types can be converted to a binary set of bytes that can be stored or transmitted across the network, then be converted back to their original object forms. Pickling also known as flattening, serializing or marshalling.

(dbhash/bsddb, dbm, gdbm, dumbdbm) and their "manager" anydbm can provide persistent storage of python strings only.

→ The last module (shelve) can do both.

Set of modules Allows for Pickling of python objects

Marshal and Pickle.

→ The difference between marshal and pickle is that marshal can handle only simple python objects (numbers, sequences, mapping and code). while pickle, can transform recursive objects, objects that are multi-referenced from different places and user-defined classes and instances

Turbo c version of Pickle is cPickle.
dump() and load() are 2 main function of Pickle.

DBM Style - Modules.

The dbm series of modules writes data to the traditional DBM format.

dbhash / bsddb, dbm, gdbm and dumbdbm.

dumbdbm is the most limited one, and is the default one if none of the other packages is available.

drawback: They can store only strings, i.e. they do not serialize python objects.

Shelve module: The shelve module use the anydbm module to find a suitable DBM module, then uses it to perform the pickling process.

→ This module permits concurrent read access to the database file, but not shared read/write access.

The relationship b/w Pickling and other persistent storage modules, and how the shelve object appears to be best both worlds.

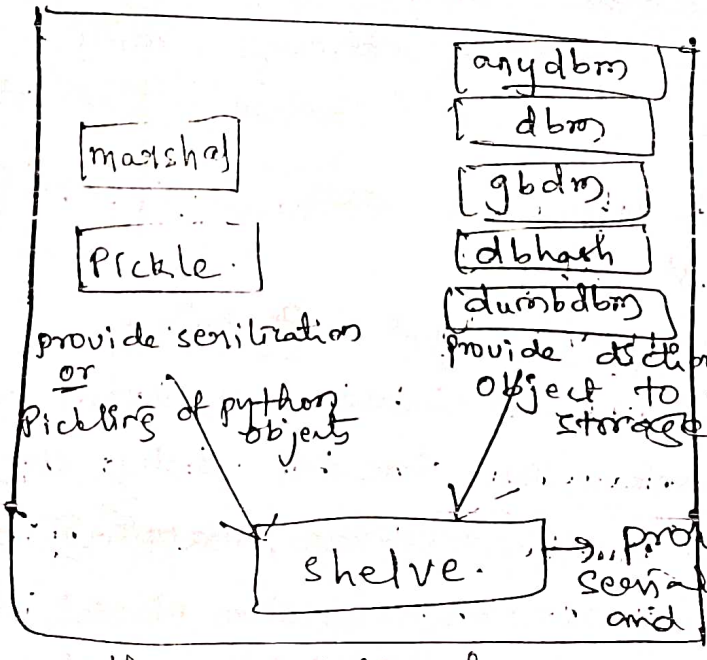


Fig: python modules for serialization and persistency

New Pickling and Unpickling of objects

→ Sometimes we have to write total state of object to the file and we have to read total object from the file.

→ The process of writing state of object to the file is called pickling. and the process of reading state of an object from the file is called unpickling.

→ We can implement pickling and unpickling by using pickle module of python.

Pickle module contains dump() and load()

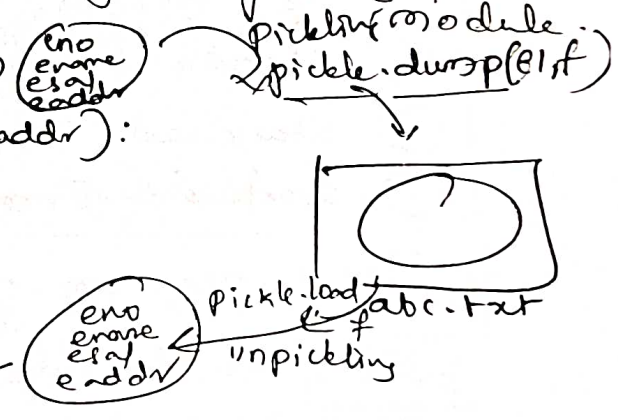
pickling

```
pickle.dump(object, file)
```

→ pickle module contains load() function to perform unpickling

```
obj = pickle.load(file)
```

Writing and Reading state of object by using pickle




```
import pickle
class Employee:
    def __init__(self,eno,ename,esal,eaddr):
        self.eno=eno;
        self.ename=ename;
        self.esal=esal;
        self.eaddr=eaddr;
    def display(self):
        print(self.eno"\t",self.ename,"\t",self.easal,"\t",self.eaddr)
with open("Emp.dat","wb") as f:
    e=Employee(100,"Durga",1000,"hyd")
    pickle.dump(e,f)
    print("Pickling of Employee object completed")

with open("emp.dat","rb") as f:
    obj=pickle.load(f)
    print("Printing Employee Info after Unpickling")
```

Related Modules

Module	Contents
base64	Encoding/decoding of binary strings to/from text strings.
binascii	Encoding/decoding of binary and ASCII-Encoded binary strings.
csv	Comma separated values
filecmp	Compares directories and files
gzip	Read and writes GNU zip files
shutil	offers high-level file access functionality
tarfile	read and writes TAR archive files
tempfile	Generates temporary file names
zipfile	Tools and Utilities to read and write zip file.

Standard Files

3 standard files

1. Standard input (Usually the Keyboard) — stdin
2. Standard Output (Buffered Output to the monitor) — stdout
3. Standard Error (Unbuffered " to the screen) — stdErr

These are pre-opened for you.

These are handled via python sys module

we have to access these files

sys.stdin, sys.stdout, sys.stdErr

The print statement normally outputs to sys.stdout while raw_input( ) built-in function receives its input from sys.stdin.

UNIT-3

Regular Expression - Characteristics (1)

(1) Matching more than one RE pattern with Alteration (|)

RE pattern	Strings Matched
at|home	at, home
r2d2|c3po	r2d2, c3po
bat|bet|bit	bat, bet, bit

(2) Matching any single character (.)

RE pattern	Strings Matched
f.o	Any character between f and o Eg: fao, f9o, f#o
..	Any pair of characters
.End	Any character before the string End.

(3) matching from the beginning or End of strings or Word boundaries (^/$)

RE pattern	Strings matched
^from	any string which starts with from Eg: from itself
/bin/tcsh$	Any string which ends with /bin/tcsh
^subject:hi$	Any string consisting solely of the string subject:hi

(4) Creating character classes ([ ])

RE pattern	Strings Matched
b[aeiu]t	bat, bet, bit, but
[cr][23][dp][o2]	'c' or 'r' then '2' or '3' followed by d or p finally either '0' or '2'

(4) Denoting ranges (-) and negation (^)

RE pattern	Strings Matched
z.[0-9]	z followed by any character then followed by single digit Eg: z#3
[r-u][env-y][us]	'r', 's', 't' or u follow En, then v or w or x or y then e or u or s.
[^aeiou]*	Zero or more Non-Vowels

(5) Multiple occurence / repetition using closure operators
(*, +, ?, { }) {m,n} m to n occurences

* — Zero or more occurences (kleene Closure)

\+ — one or " " (positive Closure)

? — Exactly 0 or 1 "

{ } — Single value or Comma Separated pair of values.

RE pattern	Strings Matched
[dn]ot?	d or n followed by o and atmost one t after that. Eg: dot, no, not, do
0?[1-9]	Any Numeric digit, possibly prepended with a "0" Eg: 01 → Jan or 1 02 → Feb or 2 whether single or double digit
[0-9]{15,16}	fifteen or sixteen digits E.g. credit card numbers

Important functions of re module.

1) match() 2) fullmatch() 3) search() 4) findall()
5) finditer() 6) sub 7) subn 8) split() 9) compile()

1) match() : We can use match function to check the given pattern at the beginning of the target string
→ If the match is available then we will get match object, otherwise we will get None.

```
import re
s=input("Enter pattern to check")
m=re.match(s,"abcdefg")
if m!=None:
    print("Match is available at the beginning of the string")
    print("start index",m.start(),"and end index",m.end())
else:
    print("Match is not available at the beginning of the string")
```

o/p
Enter pattern to check: abc
Match is available at the beginning of the string
start index: 0 and End index: 3

another
Enter pattern to check: bde.
Match is not available at the beging of the string.

2) Full match() : Complete string should be matched according to given pattern. if complete string → returns match object otherwise → None

```
import re
s=input("Enter pattern to check")
m=re.fullmatch(s,"ababab")
if m!=None:
    print("Full string matched")
else:
    print("Not Matched":
```

o/p
Enter string: ababab
Full string matched
Enter: ab
Not Matched

(5) search( ): To search for the given pattern in the target string.

Exm: string = "abaaaba"

string to searched = aaa

matched.

bbb

not matched

(4) findall( ): To find all occurences of the match

This function returns a list object which contains all occurences.

```
import re
lords = re.findall("[0-9]", "a7b9c5kz")
print(lords)
```

o/p:

['7', '9', '5']

3) finditer( ) Returs the iterator yielding a match object for each ~~match~~ match. On each match object we can call start( ), End( ), and group( ) functions.

```
import re
ftr = re.finditer("[a-z]", "a7b9c5k8z")
for m in ftr:
    print(m.start( ), "...", m.End( ), "...", m.group( ))
```

Output ⟹

```
0 ... 1 ... a
2 ... 3 ... b
4 ... 5 ... c
6 ... 7 ... k
8 ... 9 ... z
```

6) sub( ) : It means substitution or replacement

re.sub(regex, replacement, substring)

In the target string every matched pattern will be replaced with provided replacement.

```
import re
s = re.sub("[a-z]", "#", "a7b9c5k8z")
print(s)
```

Every alphabet symbol is replaced with # symbol

#7#9#5#8#

7) subn( ) Same as sub except, it can also return the no. of replacements

('#7#9#5#8#', 5)

(8) split() : If we want to split the given target string according to a particular pattern then we should go for split() function → This function returns list of all tokens.

```
import re
lords = re.split(",", "Sunday, Monday, Wedday, Thursday")
print(lords)
for t in lords:
    print(t)
```

```
l=re.split("\.", "www.lords.ac.in")
```

Sunday
Monday
Wed
Thu

www
lords
ac
in

(9) ^ symbol

```
import re
s="Learning Python is Easy"
res = re.search("^Learn", s)
if res!=None:
    print("Target String start with Learn")
Else:
    print("Target string not starts with Learn")
```

(10) $ symbol

Easy $

re.IGNORECASE

UNIT-3

Chapter-2

Multithreaded Programming

→ Introduction, Threads and Processes

→ Python, threads, and the Global Interpreter lock

→ Thread Module, Threading module, ~~Related~~ modules;

Introduction:

Single Tasking :- The process of executing a single task at a time is called as Single Tasking. In single tasking most of the processor time is wasted.

Ex: DOS.

Multi Tasking :- The process of executing multiple tasks at the same time is called Multitasking. In multi tasking the processor time utilized in an optimum way. Ex: Windows.

Context switching: The process of loading and unloading the process into the memory is called as Context Switching. In Multitasking, the processor time is divided among the tasks ~~the tasks~~ that are executed

Time slice: The small amount of processor time that is given to a particular task for execution is called as time slice.

Advantage of Multitasking: 1) Reduced Waiting time

2) Improved response time

3) Overall Performance improved.

Threads and Processes

Computer programs are just executables, binary which reside on disk. They do not take on life of their own until loaded into memory and invoked by OS.

Process

1. It is sometimes called as heavyweight process, is a program under Execution
2. Each process has its own address space, memory a data stack and other auxiliary to keep track of Execution.
3. The OS manages the Execution of all processes on the system, dividing the time fairly b/w all processes.
4. Processes can also create new processes to perform other tasks, but each new process has its own memory, data stack etc. and cannot generally share information unless IPC is employed.

Threads

1. These are sometimes called light weight processes are similar to processes except that they all execute within the same process, and thus share the same context.
   - They can be thought of as mini-processes running in parallel within a main process or "main thread"
   - It can be prempted and temporarily put on hold while threads are running. ("Yielding")
   - Multiple threads within a process share the same data space with the main thread and can share information.
   - If two or more threads access the same piece of data, inconsistent results may arise because of data access. This is commonly known as race condition.

(For this thread libraries come with synchronization)

Threads are generally executed in concurrent fashion and it is this parallelism and data sharing that enable the coordination of multiple tasks

Python, Threads and the Global Interpreter Lock (GIL)

→ Execution of Python code is controlled by Python Virtual Machine (interpreter main loop). In Python only one thread of control may be executing in the main loop, similar to how multiple processes in a system share a single CPU.

→ Multiple threads may be "running" within the Py interpreter, only one thread is being executed by the interpreter at any given time.

Access to the PVM is controlled by the Global Interpreter lock (GIL). This lock ensures exactly one thread is running.

The PVM executes in the following manner in an MT Environment (1) Set the GIL

(2) Switch to a thread to run.

(3) Execute either ...

a) for a specific number of bytecodes/instructions; or

b) If the thread voluntarily yield control (by time.sleep(0))

(4) Put the thread back to sleep

(5) Unlock the GIL and

(6) Do it all over again (repeat)

Exiting the Threads

When a thread completes execution of the function it was created for, it exits

It may also call thread.exit() or sys.exit() or SystemExit exception

Lock Type Lock object methods

acquire(wait = None)	attempts to acquire a lock object
locked()	Returns true if lock acquired, false otherwise.
release()	Releases lock.

Threading module

It is a higher-level threading module, which gives us not only a Thread class but also a wide variety of Synchronization mechanisms

Program to print Name of current Executing thread

```
1) import threading
2) print("Current Executing thread", threading.current_thread().getName())
```

Output

```
Current Executing thread : Main thread.
```

→ threading module contains function current_thread() which returns the current Executing thread object. On this object if we call getName() method then we will get current Executing thread Name.

** The ways of creating a thread in Python.

(3 Ways)

1. Creating a Thread without using any class
2. " " " by Extending a Thread class.
3. " " " without " " " .

Accessing threads from Python

By default, threads are enabled when building python from source (Python 2.0 onwards)

```
import thread
```

Life without threads

→ If we want to perform independent computations for a program, without threads, it takes a lot of time. So it's better to use multithreaded program (MT).

Python Threading modules.

① thread module ② threading module ③ Queue module

① These allow the programmer to create and manage threads.
- Basic thread and locking support

② Higher level, full featured

③ Queue module allows the user to create a queue data structure that can be shared across multiple threads

thread module

In addition to being able to spawn threads, the thread module also provides a basic synchronization data structure called a lock object.

thread module and lock objects

Function/method	Description
thread module functions	
start_new_thread(function, args, kwargs=None)	Spawns a new thread and execute a function with the given args and optional keyword args (kwargs)
allocate_lock()	Allocates lock type to lock object
exit()	Instructs a thread to exit

threading module objects

Objects	Description
Thread	Object that represents single thread of Execution
lock	Primitive lock object
RLock	Re-Entrant lock object provide ability (Recursive locking)
Condition	Condition variable object causes one thread to wait until a certain "condition" has been satisfied by another thread, such as changing of state or of some data value
Event	General version of condition variables whereby any no. of threads are waiting for some ~~time~~ Event to occur and all will awaken when the event happens.
Semaphore	provides a "waiting area" - like structure for threads waiting on a lock.
Bounded Semaphore	Similar to semaphore but ensures it never exceeds its initial value.
Timer	Similar to thread except that it waits for an allotted period of time before running

Thread object methods

Method	description
start( )	Begin thread Execution
run( )	Method defining thread functionality
join( ) (timeout = None)	Suspend until the started thread terminates; blocks unless timeout is given
getName( )	Return Name of thread
setName(name)	Set Name of thread
isAlive( )	Boolean flag indicating whether thread is still running
isDaemon( )	Return daemon flag of thread
setDaemon( )	set the daemon flag of thread as per boolean

Every thread in python has a name. It may be default. It may be a default name generated by python or customized name provided by programmer.

→ We can get and set name of thread by using following Thread class methods.

t.getName() → Returns Name of thread
t.setName(newName) → To set our own Name.

Note:- Every Thread has implicit variable "name" to represent name of Thread.

```
from threading import *
print(current_thread().getName())
current_thread.setName("PawanKalyan")
print(current_thread.getName())
print(current_thread.name())
```

Output:-

```
Main thread
PawanKalyan
PawanKalyan.
```

→ Thread Identification Number (Ident) :- for Every thread internally a thread Identification number is available. We can access this ID by using implicit variable "ident".

```
from threading import *
print("Main thread ID", current_thread().ident)
```

O/P

Main thread ID : 2492.

Related Modules

Module	Description
thread	→ Basic, lower-level thread module
threading	→ Higher level threading and Synchronization objects
Queue	→ Synchronized FIFO Queue for multiple threads
mutex	→ Mutual Exclusion objects
SocketServer	→ TCP ... UDP ... IO ...

REs and Python

Now we know all about REs., Here we will see. how python supports all these using re module

Functions of RE module

1 Compile( ) : This function can be used for specifying a pattern that has to be searched or Compile a regular Expression pattern, returning a pattern object. (or) Retuns module contains multiple( ) Function to compile a pattern into a Regex object

```
pattern = re.compile("ab")
```

2) Finditer( ) : Returns an iterator object which yields match object for Every match.

```
matcher = pattern.finditer("abababa")
```

On match object we can call the following methods

1) start( ) → Returns start index of the match
2) End( ) → Returns End+1 index of the match
3) group( ) → Returns the matched string

```
import re
count = 0
pattern = re.compile("ab")
match = pattern.finditer("abababa")
for match in matcher:
    count = count + 1
    print(match.start(),"...",match.End(),"...",match.group())
print("The no.of occurences:", count)
```

Output:

```
0 ... 2 ... ab
3 .... 5 .... ab
```

```
import re                    (passing pattern directly as arg to finditer)
count = 0
matcher = re.finditer("ab","abababa")
for match in matcher:
    count += 1
    print(match.start(),"...",match.end(),"...",match.group())
print("The no.of occurence", count)
```

Output:

```
0 ... 2 ... ab
3 ... 5 ... ab
5 ... 7 ... ab
The no.of occurences: 3
```

Special characters representing character set, (2)

"0-9" → \d → to indicate the match of any decimal digit.

\w → A-Z a-z 0-9

Alphanumeric class

Upper case versions of these string symbolizes a Non match

"\D" → Non-decimal digit [^0-9]

RE pattern	Strings matched
\w+-\d+	Alphanumeric string and numb seperated by a hyphen lords-123
[A-Z a-z]\w+	Alphanumeric first characta., additon characters (if present) can be alphanu a123, a1, MS, (Python Identifier)

Designating groups with parentheses (( ))

① Grouping RE ② Matching sub groups.

RE pattern	Strings matched
\d+(\.\d*)?	Strings representing simple floating point numbers i.e Any no. of digits followed optionally by a single decimal point and zero or more numeric digits Exm: 32.4, 23, 23.
(Mr?s?\.)?[A-Z][a-z]* [A-Z a-z]+	first Name and last name with restricted first name (must start with Uppercase. lower Case only for remaing letters of any flexible last name allowing m

The full name prepended by an optional title of

Mr., Mrs, Ms M

Unit- IV

GUI Programming, WEB Programming

UNIT - 4

4-(1) GUI Programming:- Introduction, Tkinter and Python Programming, Brief tour of other GUIs, Related Modules and other GUIs.

4-(2) Web Programming:- Introduction, Web Surfing with Python, creating simple web clients, Advanced web clients CGI - Helping Servers process client data, Building CGI Application, Advanced CGI, Web (HTTP) Servers.

Introduction

GUI stands for Graphical User Interface.

→ The primary GUI toolkit in Python is Tkinter (Tk in short).

→ Tkinter is based on the Tk toolkit, originally designed for Tool Command language (Tcl).

→ Tk is ported to variety of Programming languages (Perl/Tk) → Perl, Ruby/Tk → Ruby, Python → Tkinter

→ like threading, Tkinter is imported using

→ `import Tkinter`

Tkinter and Python Programming

5 steps that are required to get your GUI up and running

1. Import the Tkinter module (or from Tkinter import *)
2. Create a top-level windowing object that contains your GUI Application.

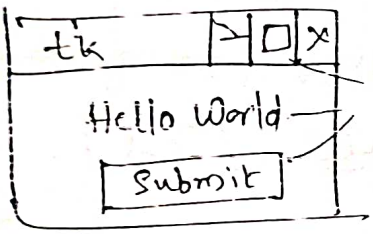
(3) Build all your GUI Components on top of your top-level windowing object.

(4) Connect these GUI Components to the underlying application code.

(5) Enter the main Event loop.

→ In GUI programming, a top-level root windowing object contains all of the little windowing objects that will be part of your complete GUI application.

These can be text labels, buttons, list boxes etc. These individual little GUI components are known as widgets.



In Python

```
top = Tkinter.Tk # just Tk() w
from Tkinter import *
```

The object returned by Tkinter.Tk() is referred as root window. You may have more than one top-level window for GUI, but only one of them should be root window.

If a widget contains other widget it is known as parent widget, 'contained' widget is known as child widget.

→ widgets have associated behaviours, such as when a button is pressed, or text is filled into a text field. These types of user behaviours are called events, and all the actions that the GUI takes to respond to such events are known as Callbacks.

Other Event are button press (release), mouse movement hitting the RETURN or Enter key.

The Entire System of Events that occurs from the beginning to the End of a GUI application is what drives it. This is known as Event-driven processing.

① Packer pack() and ② Grid are two geometry managers of Tk to position the widget in right place.

The final step to Enter the main loop once all the widgets are ready. (Server infinite loop)

```
Tkinter.mainloop()
```

Top-level Window : Tkinter.Tk()

All the main Widgets are built into the top-level window. object. This object is created by the Tk class in Tkinter

```
>>> import Tkinter
>>> top = Tkinter.Tk()
```

Within this Window we will place all widgets.

Tk Widgets

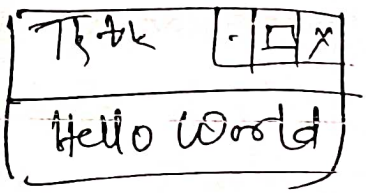
There are 15 types of widgets currently available.

Widget	Description
Button	Similar to a label, but provides additional functionality for mouse overs, presses, and releases as well as keyboard activity/Events
Canvas	provides ability to draw shapes (lines, ovals, polygons, rectangles). Can contain images or bitmaps
Checkbutton	Set of boxes of which any number can be checked (display)
Entry	single line text-field with which to collect keyboard input
Frame	pure container for other widgets
Label	Used to contain text or images
Listbox	present user list of choices to pick from.

Menu	Actual list of choices from "menubutton" that the user can choose from.
Menubutton	Provide infrastructure to contain menus (pulldown, cascading etc)
Message	Similar to Label, but displays multi-line text
Radiobutton	Set of buttons of which only one can be pressed
Scale	Linear slider widget providing an exact value at current setting; with defined starting and Ending values.
Scrollbar	Provides scrolling functionality to supporting widgets i.e. Text, Canvas, List-Box and
Text	Multi-line text field with which to collect text from user
Toplevel	Similar to a Frame, but provides a separate window Container

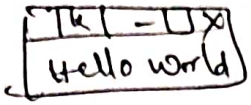
Exm:- Label Widget Demo (tkhello1.py)

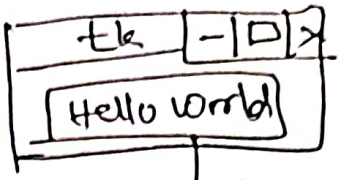
```
1. #! /usr/bin/env python
2  import Tkinter
3  top = Tkinter.Tk( ) #Creates a top-level window
4  label = Tkinter.Label(top, text = "Hello World")
5  label.pack( ) # packer to manage and display Hello world label text
6  Tkinter.mainloop( ) # to run GUI application widget
```



Tkinter Examples

(1) Label widget (2) Button widget





Button widget demo (tkhello2.py)

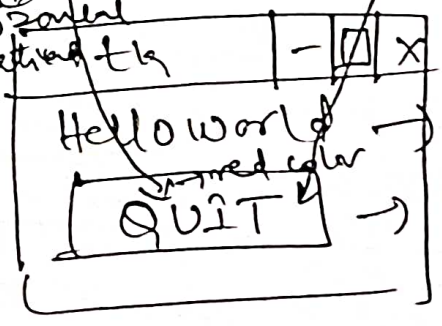
```
import Tkinter
top = Tkinter.Tk()
quit = Tkinter.Button(top, text = 'Hello World',
        command = top.quit )
# Button widget with Tkinter.
quit.pack
Tkinter.mainloop()
```

when pressed method quit()
(callbacks or releases)

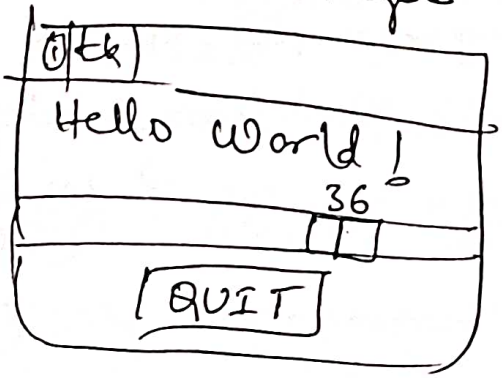
(3) Label and Button Widgets (tkhello3.py)

```
import Tkinter
top = Tkinter.Tk()
hello = Tkinter.Label(top, text = "Hello World")
hello.pack()
quit = Tkinter.Button(top, text = 'QUIT', command =
        top.quit, bg = 'red', fg = 'white')
quit.pack(fill = Tkinter.X, expand = 1) landscape
Tkinter.mainloop
```

Horizontal
Y-vertical



(4) Label, Button and scale widgets



Tix GUI Demo

(animalTix.pyw)

```
#!/usr/bin/env python
from Tkinter import Label, Button, END
from Tix import Tk, Control, ComboBox
top = Tk()
top.tk.eval('package require Tix')
lb = Label(top,
        text = 'Animals (in Pairs; min: pair, max: dozen)')
lb.pack()
ct = Control(top, label = 'Number:',
    integer = True, max = 12, min = 2, value = 2, step = 2)
ct.label.config(font = 'Helvetica -14 bold')
ct.pack()
cb = ComboBox(top, label = 'Type:', editable = True)
for animal in ('dog', 'cat', 'hamster', 'python'):
        cb.insert(END, animal)
cb.pack()
qb = Button(top, text = 'QUIT', command = top.quit,
        bg = 'red', fg = 'white')
qb.pack()
top.mainloop()
```

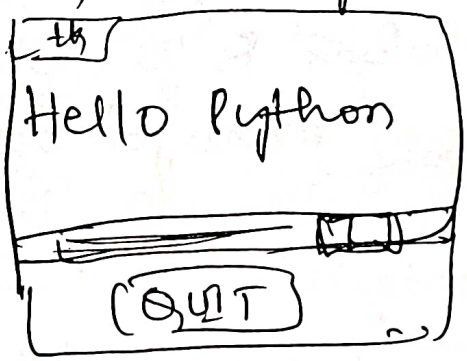
GTK (formerly GTK+, GIMP Toolkit) is a free and open source - cross platform widget toolkit for creating Graphical User Interface (GUIs)

Label, Button and Scale Demo

(tkhello4.py)

```
from Tkinter import *
def resize(ev=None):
    label.config(font='Helvetica -%d bold' %
                 scale.get())
top=Tk()
top.geometry('250x150')
label = Label(top, text='Hello Python', font='Helvetica
    -12 bold')
label.pack(fill=Y, expand=1)
scale = Scale(top, from_=10, to=40, orient=HORIZONTAL,
    command=resize)
scale.set(12)
scale.pack(fill=X, expand=1)
quit = Button(top, text='QUIT', command=top.quit,
    activeforeground='white', activebackground='red')
quit.pack()
mainloop()
```

vertical



Related Modules and other GUIs

(1) Tk-Related Modules (2) wx-widget Related modules

(3) GTK+/GNOME-Related modules (4) Qt/KDE-Related modules

(5) Other open source GUI toolkits

(I) Commercial.

Brief Tour of other GUIs

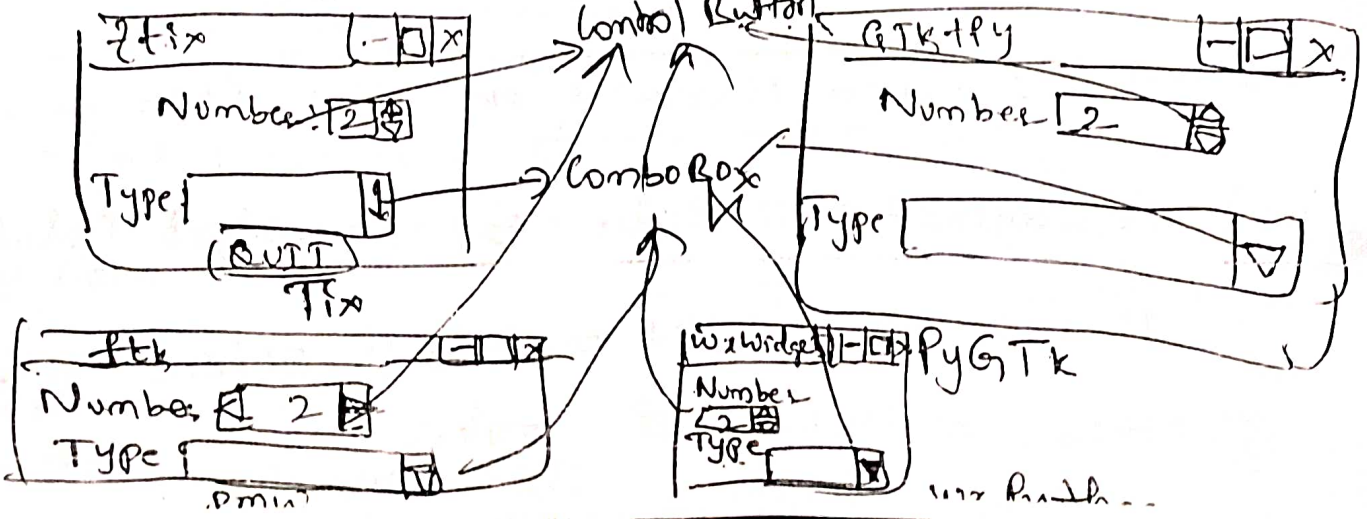
4 other popular toolkits are

1. Tix (Tk interface Extension) (Already available in Python standard Library)
2. Pmw (Python Mega Widgets Tkinter Extension)
3. wx Python (Python binding to wxWidgets written in C++ for Python)
4. PyGTK (Python binding to GTK+)

(3 & 4 are 3rd party extensions that must be downloaded)

Apart from Label and Button Widgets, we introduce Control or SpinButton and ComboBox.

→ Control Widget is a combination of text widget with a value inside being "controlled" or "spun up or down" by a set of arrow buttons close by, and the ComboBox is usually a text widget and a pulldown menu of options where the currently selected item in the list is displayed in the text widget



Chapter - 2 (UNIT - 4)

Web Programming

Introduction

In this chapter we will study, high-level overview of the kinds of things we can do with python on the Internet, from web surfing to creating user feedback forms, from recognizing URL to generate dynamic web page client.

Introduction - web surfing: Client/Server Computing

Web surfing falls under the same client/server architecture umbrella that we have seen repeatedly.

This time, web clients are browsers, applications that allow users to seek documents on the WWW.

On the other hand, Web servers processes it provides data.

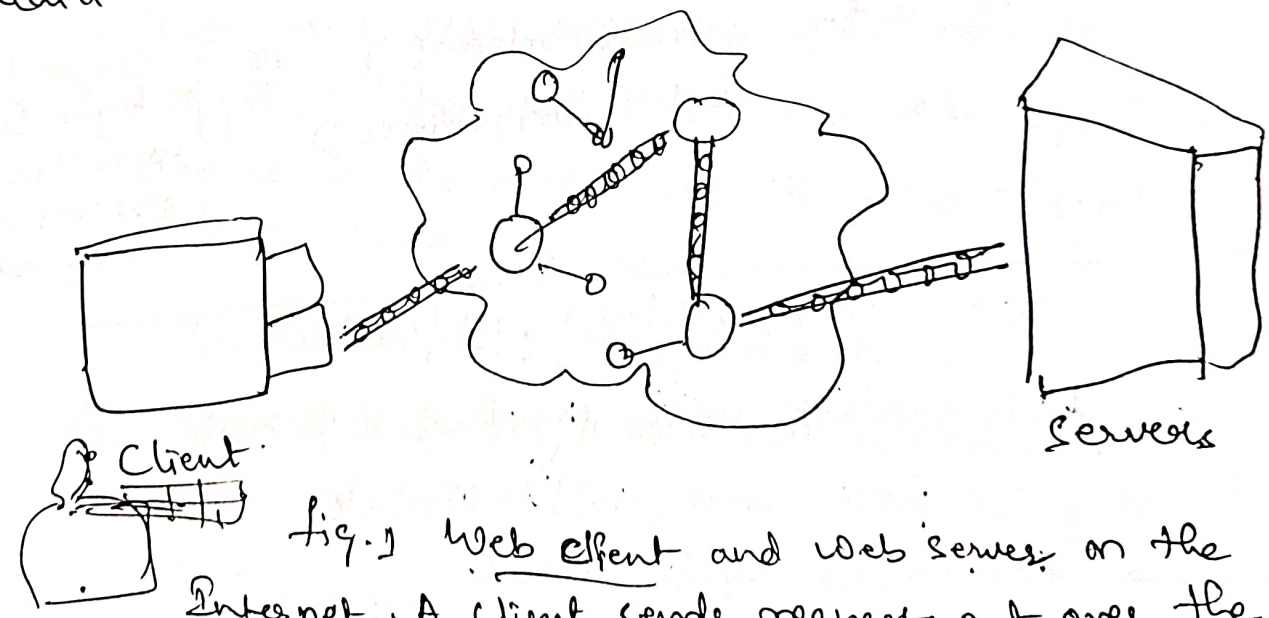


fig.1 Web client and web servers on the Internet. A client sends request out over the Internet to the server, which then responds with the requested data back to the client.

→ The language that is spoken by web clients, and servers, the standard protocol used for web communication is called HTTP. It is written on top of TCP/IP.

Web Surfing with Python : (Creating Simple web clients)

→ A browser is only one type of web-client. Any application that makes a request for data from a web server is considered as client.

→ It is possible to create other clients that retrive documents or data off the Internet.

→ Browser has limited capacity i.e. is used primarily for viewing and interacting with web sites.
On the other hand, the client program ~~does~~ ability to do more
- It can not only download the data, but it can also store it, manipulate it, or even transmit it to other location or application.

← Applications that use ~~that~~ ~~use~~ urllib module to download or access information on the web.

using either urllib.urlopen() or urllib.urlretrieve() } can be considered by a simple web client by providing valid web address

---

Web Programming: Introduction

→ Web Surfing with python : Simple web Client
- Urlparse and urllib Modules

→ Advanced Web Clients
- crawler / Spider / Robot

→ CGI : Helping web servers process client Data

→ Building CGI Applications

→ Advanced CGI → Web (HTTP) Servers

2.1 URL (Uniform Resource Locators) (Web Address)

URLs are used to locate a document on the web or to call a CGI (Common Gateway Interface) program to generate a document for your client.

A URL is normally made up of 3 or 4 components

1. **A scheme** The scheme identifies the protocol to be used to access the resource on the Internet. It can be
   HTTP (without SSL) or HTTPS (with SSL)
   or FTP (local network server)
2. **A host** The host name identifies the host that holds the resource.
   for Exm: www.example.com
   Hostnames can also be followed by a Port Number
   Most HTTP URLs omit the port number.
3. A path: The path identifies the specific resource in the host that the web client wants to access.
   for Exm
   /Softwares/http/cics/index.html
4. A query string: It follows path component. It is usually a string of name and value pairs
   Exm:
   term = bluebird
   Name and value pairs are seperated from each other by an ampersand (&)
   term = bluebird & source = browser-search

The Components of the URL are combined and delimeted as follows

scheme : // host : port / path ? query

Exm:

http : // www. Example-com : 1030 / Software / index.html

scheme — IP-addres host — port Number — path component

In python, a URL uses this format

prot_sch : // net_loc / path : params ? query # frag

Web address Components

URL Component	Description
prot_sch	Network protocol or download scheme
net_loc	location of server
path	Slash (/) delimited path to a file
params	Optional parameters
query	Ampersand (&) delimited set of "key = value" pairs
frag	fragments to a specific anchor within document

Network location Components

Component	Description
user	User name or login
passwd	User Password
host	Name or address of machine running web server
port	Port Number (if not so, the default

required only for FTP connection

iii) urlparse.urljoin() : It is useful in cases many related URLs are needed. for Example, the URLs for a set of pages to be generated for a website.

The syntax of the urljoin() is

urljoin(base url, newurl, allowfrag= None)

Ex:

urlparse.urljoin('http://www.python.org/doc/FAQ.html', 'current/lib/lib.html')

output

http://www.python.org/doc/current/lib/lib.html

Urllib Module

This module provides functions to download data from given URLs as well as Encoding decoding strings to make them suitable for including as part of valid URL strings.

Core Urllib Module functions

Urllib function	Description
urlopen(urlstr, postQueryData=None)	Opens the URL urlstr, sending the query data in a postquerydata if a POST request
urlretrieve(urlstr, localfile=None, downloadstatushook=None)	downloads the file located at the urlstr URL to a local file, downloadstatushook is a function that can receive download statistics
quote(urldata, safe='/')	Encodes invalid URL characters of urldata : characters in 'safe' string are not encoded.
quote_plus(urldata, safe='/')	Same as quote() Except Encodes spaces as plus (+) sign
unquote(urldata)	decodes encoded characters of urldata
unquote_plus(urldata)	Same as unquote() but converts plus signs to spaces
urlencode(dict)	Encodes the key-value pairs of dict into a valid string for CGI query and Encodes the keyword and value strings with

Python supplies 2 different modules for dealing with URLs.

① Urlparse ② Urllib

Urlparse module: This module provides basic functionality with which to manipulate URL strings. These functions include

i) urlparse() ii) urlunparse() iii) urljoin()

i) urlparse.urlparse(): urlparse break-up a URL string into some of the major components.

It has the following syntax:

urlparse(urlstr, defprotsch=None, allowFrag=None)

urlparse() parses urlstr into 6-tuple (prot-sch, net-loc, path, params, query, flag).

Exm: Urlparse() output given a URL

>> urlparse.urlparse('http://www.python.org/doc/FAQ.html')

output

('http', 'www.python.org', '/doc/FAQ.html', '', '', '')

ii) urlparse.urlunparse(): It does the exact opposite of urlparse() - It merges above 6-tuples into a single URL

We state the following Equivalance

urlunparse(urlparse(urlstr)) = urlstr

Syntax:

urlunparse(urltup)

How parse url into diff components using urlparse()

demo.py

```
from urllib.parse import urlparse

url = 'http://wikipedia.org'
obj = urlparse(url)
print(obj.scheme)
print(obj.netloc)
print(obj.hostname)
print(obj.geturl())
print(obj.port)
print(obj.path)
```

o/p

```
http
wikipedia.org
wikipedia.org
http://wikipedia.org
None
```

→ not print, on which port wikipedia is running

Advanced Web Clients

Web browsers are basic web clients. They are used primarily for searching and downloading documents from the web.

→ Advanced web clients are those applications that do more than download single documents from the Internet.

Exm: crawler (also known as spider, robot)
These programs explore and download pages from Internet for different reasons, some of which include

(1) Indexing into a large search Engine such as Google or Yahoo!
(2) Offline Browsing (3) Downloading and storing for historical purposes (4) Web page caching to save downloading time or website traffic.

few advanced web clients in the market

Bing Bot (msnbot) — Microsoft's Web Crawler

Google Bot — Google.

Xenon — Used by Govt. Tax authorities to detect fraud.

Yahoo! Slurp — Yahoo

Open Source

Frontera, GRUB, HTTrack, GNU Wget, Heritrix, Apache Nutch, PHP-crawler, BeautifulSoup

→ Scrapy, an open source web crawler framework written in Python

→ Xapian, a search crawler Engine, written in C++

Exm: Do a Web Crawler project using Python scrapy (Crawler for searches across multiple pages) for OLX's Electronics & Appliances items.

UNIT-5

Database Programming: (1)

→ Introduction

→ Python DataBase Application Programmer's Interface (DB-API) Application programming interface

→ Object Relational Managers (ORMs)

→ Related Modules.

→ In this chapter, we discuss how to Communicate with databases from python.

Introduction

Generally there are three basic storage mechanisms

1) files 2) Relational Database Systems (RDBMSs) or API

3) Object Relational Mapper (file manager, spreadsheet, Configuration file etc.)

Basic Database operations and SQL:

Quick Introduction of some Elementary database concepts

Example ORM Libraries

Java – Hibernate.
PHP – propel or Doctrine
Python – the Django ORM or SQLAlchemy
C# – NHibernate or Entity framework

Basic Database operations and SQL (2)

Before we dig into databases and how to use it with python, we introduce some elementary database concepts and the SQL.

1) Underlying Storage: Databases usually have a fundamental persistent storage using the file system i.e. Normal OS files, etc.

2) Uses interface: Most database systems provide a command line tool with which to issue SQL commands or queries. There are also some GUI tools that use the command line clients, giving users a much nicer interface (SQL Query Builder Tool) MySQL GUI)

3) Databases: An RDBMS can usually manage multiple databases, all on the same server.
Eg: MySQL is an example of a server-based RDBMS because there is a server process running continuously waiting for commands. (SQLite, Godfy) not having

4) Components.
The table is storage abstraction for databases. Each row of data will have fields that correspond to database columns. The set of table definitions of columns and datatypes per table all put together define the database schema.

SQL Commands are: create, drop (both for database and tables)
insert, update, ~~drop~~ insert
→ changing existing rows in a table is called updating and removing is called

21.1.3 Databases and python

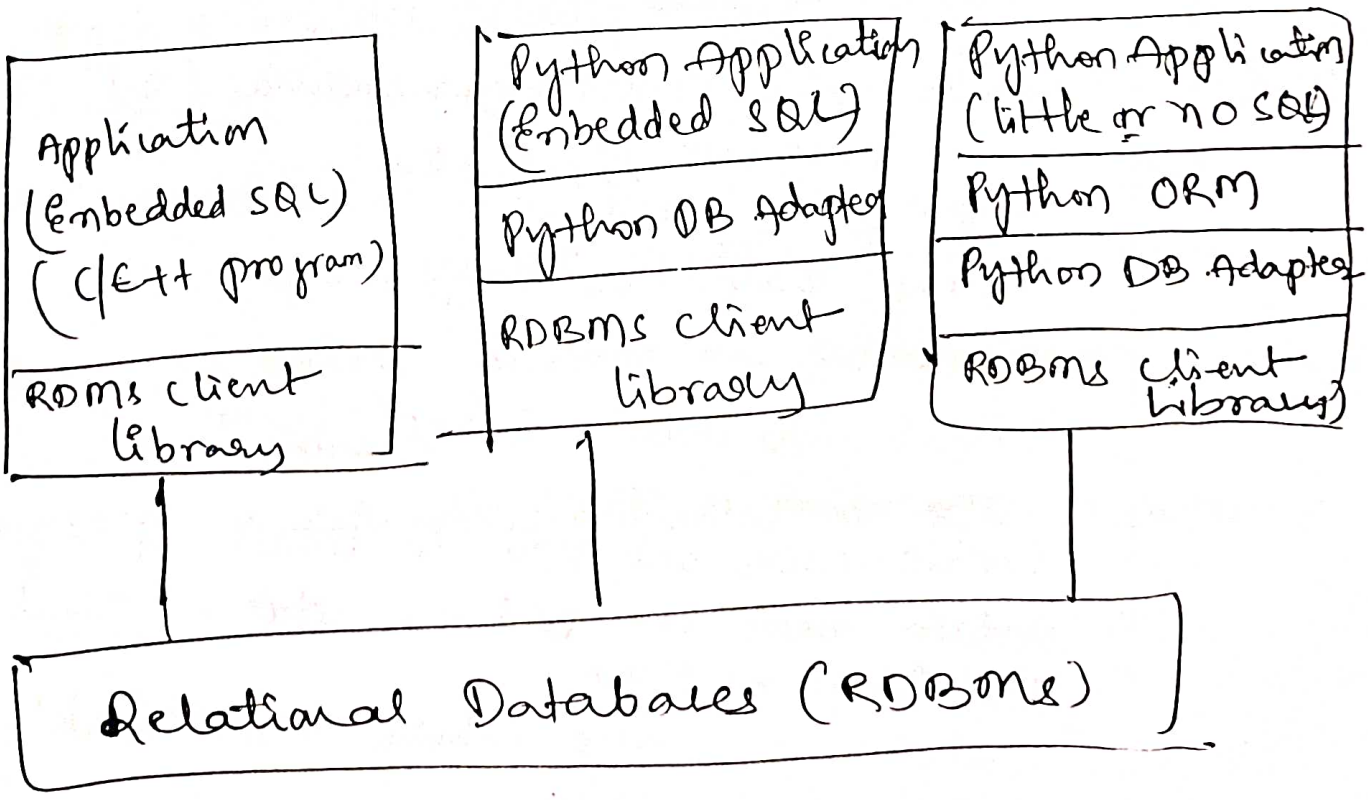
We can access relational databases from python either directly through a database interface, or via an ORM.

→ Aaron Watter's Gadfly, is a simple RDBMS written completely in Python.

→ The way to access a database from python is via an Adapter.

→ An Adapter is basically a python module that allows you to interface to a relational database's client library, usually in C.

→ The following fig illustrates the layers involved in writing a python database application, with and without an ORM.



Requesting rows from a database with optional criteria is called querying

SQL

Database commands and queries are given to a database by SQL. Commands are not case sensitive

Creating a Database

```
Create Database test;  → Assuming a DBA
Grant all on test.* to user(s);
```
↳ Granting permissions to specific users (or all)

Using a Database:

```
Use test;
```

Dropping a Database

```
DROP database test;  (Removes all the the tables and data)
```

Creating a Table:

```
create table users(login varchar(8), uid INT,
    prid INT);  project ID
```

dropping a table:

```
drop table users;
```

Inserting a row

```
Insert into users values('Leanna', 311, 1);
```

Updating a row

```
update users set prid = 4 where prid = 2;
update users set prid = 1 where uid = 311;
```

Deleting a row

```
Delete from users where prid = '1od';  → any number
Delete from users;  → All rows will be deleted
```

Limitations of Databases:

① Database cannot hold very huge amount of data (Terabytes of data)

② Database can provide support only for structured data (Tabular or Relational Data) and cannot provide support for semi structured data (like XML) and unstructured data (like Video files, Audio files, Images etc.)

To overcome these problems we should go for more Advanced storage Areas like Big data Technologies, Data Warehouses etc.

Python Database Programming:

-> We can use SQL language to talk to the database and we can use python to send those SQL commands to the database.

-> Python provides inbuilt support inbuilt support for several databases like Oracle, MySQL, Sqlserver Gadfy, sqllite etc.

-> Python has seperate module for each database.

Eg: cx-Oracle module for communicating with Oracle database

pymssql module for communicating with microsoft Sql server.

Standard Steps for Python Database Programming

1). Import database specific module.

Eg: import cx_Oracle.

2) Establish Connection b/w Python Program and database. We can create this connection object by using connect() function of the module.

```
con = cx_Oracle.connect('scott/tiger@localhost')
```

3) To execute our sql queries and holds results some special object is required, which is nothing but Cursor object. We can create Cursor object by using cursor() method.

```
cursor = con.cursor()
```

4) Execute SQL queries by using Cursor object. following methods are used for this.

* execute(sqlquery) → To execute a single SQL query
* executescript(sqlqueries) → To execute a string of SQL qu seperated by semicolon ';'
* executemany() → To execute a parameterized query.

Eg: cursor.execute("select * from Employees")

(5) Commit or Rollback changes based on requirement

commit() → Save the changes to the database.

rollback() → roll all temporary changes back

(6) fetch the result from the Cursor object

```
Eg: data = cursor.fetchone()
    print(data)
```

(7) Close the Resources (close in reverse order) → cursor.close() con.close()