

PYTHON

BY

NARAYANA

DURGA SOFT

FRIENDS XEROX

BEHIND MITRYVANAM,GAYATHRI NAGAR,AMEERPET

CELL: 9908909131

INDEX

1. Introduction to python language — 1.
2. Scripting and programming language. — 3
3. Features of python language. — 5.
4. Advantages & disadvantages of python language. — 11.
5. why should we learn python language. — 13.
6. python Datatypes compare to other languages. — 15.
7. python application or program development — 16.
8. python identifier. — 18.
9. python keywords. — 21
10. python indentation. — 22
11. Multi-Line statements. — 25
12. Quotations in Python — 25
13. Comments in python. — 26
14. Python Variables. — 28
15. First python program — 35

Python
by
Narayana

- 16. Reading data from keyword — 37.
- 17. Input and output statements — 41
- 18. Command line arguments — 47
- 19. Data types in python — 55
- 20. Type Conversion function. — 58
- 21. eval function. — 64

Assignment -1

- 22. String - Data Structure. — 80
- 23. List - Data Structure. — 103
- 24. Tuple - Data Structure — 135
- 25. Set - Data Structure. — 159
- 26. Dictionary - data structure: — 177

Assignment -2

- 27 Operators. — 223
- 28. Conditional statements. — 241
- 29. Iterative statements. — 268
- 30. Transfer statement — 299
- 31. Python functions. — 282

Python
by
Marayama

32 python format function — 329

33 Lambda functions — 337

34 Object Oriented programming. — 347

- a Introduction to OOPS
- b Static & class Variables
- c Constructor
- d Non static or instance variable
- e Data hiding.
- f Data abstraction
- g Encapsulation
- h Inheritance
- i polymorphism

35 Reading & writing files — 389

36 file handling. — 396 Assignment-3

37 Python modules — 432

38 Exception handling. — 461

39 Unit testing. — 494

40 Iterators. — 501

41 Generators — 505 "O"

42 Decorators — 511

Python by
Narayanan

Python
by
Narayana
= O =

- 43 Multi-threading — 517
- 44 Database Connection — 521
- 45 Numpy — 532
- 46 Working with Logging — 546
- 47 Comprehension — 549
- 48 Pickling & unpickling of objects — 557
- 49 Regular Expression. — 561

Assignment - 4

Python
by
Narayana

27/04/18 python

* Introduction to python * Python Narayana

①

1. under the Leadership of "Andrew Stuart Tarenbaum"
a group of employees developed distributed operating system.
2. Group of employees used ABC scripting language to develop Distributed Operating system.
3. 'ABC' scripting language is ~~half~~ very simple and easy to Learn and work.
4. "Guido van Rossum" is a member in that group and he likes ABC scripting language very well as it is very simple and easy.
5. In Christmas ~~Holidays~~, Guido van Rossum started developing a new language to be simpler and easy compare to ABC scripting language and all other existing languages.
6. finally he developed a new language.
7. He likes "Monty python's flying circus" english daily serial very well.
8. So finally Guido Van Rossum taken word 'PYTHON' from that serial and kept for his language

python

python Narayana

9. So finally Guido Van Rossum developed python ② Scripting language at the National Research institute for mathematics and computer Science in Netherlands in 1989 and it available to the public in 1991.
10. python is now maintained by a core development team at the institute, although Guido Van Rossum still holds a vital role in directing its progress.
11. Python 1.0 was released in January 1994.
12. Python 2.0 was released in October 2000 any python 2.7.11 in the last edition of python 2.
13. Meanwhile, python 3.0 was released in December 2008.
14. Python is a general purpose high level programming language.
15. Python is recommended as first programming language for beginners.
16. Guido Van Rossum has developed python language by taking almost all programming features from different languages.
17. The most of syntaxes in python have taken from C and ABC Languages.

Python Scripting and programming Languages; Python Narayana

Scripting Languages

(3)

1. Scripting Languages based applications are run by interpreter.
2. Scripting Languages based applications are not required explicit compilation.
3. We can run these scripting Language based Language directly.
4. Examples of Scripting Languages are shell script, python, perl, Ruby, powershell... These all languages are run by interpreter, that's why these all languages are called Scripting languages.

Programming Languages

1. programming Languages based applications are run by compiler.
2. programming languages based applications are required explicit compilation.
3. programming Languages based applications can't run directly without compilation.
4. Eg of programming Languages are c, C++, JAVA, .NET... These all programming languages are run

^{python}
by compiler. That's why these all languages ^{python Marayana} are called programming languages. ①

Note: Most of the people call python is a scripting language because the way of developing python applications and executing python applications are similar to scripting languages.

what is python?

1. python is a simple, powerful and high level, interactive, object oriented scripting language.
2. python is a general purpose and portable language.

Different Languages used to develop python:-

1. procedural oriented programming language---c
2. Object oriented programming language---C++, Java.
3. Scripting languages --- shell script, perl.
4. Modular programming Language :--- Modula-3.

^{Narayana}
Python

Python * Features of python * Python Norayana.

(5)

1. python is simple, easy and powerful language.
 - a. The syntax of python is very simple and easy to remember, so anybody can easily remember the python syntaxes without having any programming basics.
 - b. Even non-technical persons also can learn to work with python language. So here developers not required focusing on syntaxes.
 - c. It allows programmers to concentrate on the solution to the problem rather than the language itself.
 - d. Reading a python program feels almost like reading English, although very strict English. This pseudo-code nature of python is one of its greatest strengths.
 - e. This elegant syntax of python language makes the developers to express their business logic in very less lines of code. So when developers write less code then automatically application development time, application cost and also application maintenance time also will be decreased.

python

Python Narayana

f. that's why python is also called program friendly language.

⑥

g. python has few keywords, simple structure, and a clearly defined syntax. this allows the student to pick up the language quickly.

2. python is an Expressive Language.

(a) python language is more expressive. the sense of expressive is the code is easily understandable.

3. python is free, Open and Redistribution language.

a. python is an example of Floss(free) Libre and open source software.

b. FLOSS is a community, which itself is based on the concept of sharing knowledge.
floss are free for usage, modification and redistribution.

Examples of Floss are

1. Linux
2. ubuntu
3. Libre office
4. Mozilla firefox
5. Mono
6. Apache web Server & VLC player.

Python

4. python is an Interpretable Language Python Narayana
a. python is an interpreted language i.e., interpreter executes the python code line by line at a time. So, we don't need to compile our program before executing it. This makes debugging easy and thus suitable for beginners.
5. python is an extensible language.
a. python can completely integrate with components of Java and .Net, and also can invoke libraries of C and C++. So here python performs cross language operations.
b. if we need a critical piece of code to run very fast, you can code that part of your program in C (or) C++ and then use them from our python program.
6. python is very rich in Libraries
a) The python is very rich in libraries. They help us do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI (Common Gateway interface), ftp (file transfer protocol), email,

python

python Narayana

XML, WAV files, cryptography, GUI (Graphical user interface) using TK (Toolkit) and also other system-dependent stuff. Remember, all this is always available wherever python is installed.

b. this is called the "batteries included" philosophy of python.

7. python is oriented programming language

a. python supports "procedure-oriented programming" as well as "object-oriented programming" in procedure-oriented languages.

b. the program is built around procedures (81) functions which are nothing but reusable pieces of programs. In object-oriented languages,

c. The program is built around objects which combine data and functionality. python is very powerful in way of doing object-oriented programming, especially, when compared to languages like C++ (81) Java.

8. python is a portable language:

a. Due to its open source nature, has been ported to many platforms.

Python

Python Narayana

- b. All our python programs will work on any platform without requiring any changes at all So python is also called "Cross platform Language" ⑨
- c. we can use python on Linux, windows, Macintosh, Solaris, os/2, Amiga, AROS, AS/400, BEOS, OS/390, z/OS, palmos., QNX, VMS, psion, Acorn RISC OS, Vxworks playstation, Sharp Zaurus, Windows CE and pocket PC!

9. High-Level Language:-

- a. when you write programs in python, you never need to bother about the Low-level details such as managing the memory used by your program.
- b. The memory allocates dynamically when we run the application.

Some Other features of python Languages-

- 1. Interactive Language.
- 2. Beginners Language.
- 3. Easy - to - Learn.
- 4. Easy - to - maintain.
- 5. Easy - to - work.
- 6. Supports all major commercial databases.
- 7. Supports all major commercial databases.

Python

- 7. Supports GUI applications.
- 8. Scalable.
- 9. Supports multiple databases.
- 10. Supports inheritance.

python Narayana

(10)

* Installing python 3:-

Step 1 Goto "www.python.org" and download latest version of python, like below.

Step 2 :- goto downloads folder and select downloaded python and run and install

Step 3 :- Set variable path like below.

My Computer → properties → Advanced system Settings → environment variables → under user variables, create new variable with name PATH, like below.

path = c:\users\Narayana\AppData\Local\programs\Python
 \python 36-32\scripts;

c:\users\ Narayana \AppData\ Local \ programs\Python\python 36-32\

Python Narayana

Python

what can we do by using Python?

Python

Narayana

(11)

By using python we can develop;

1. GUI Applications.
2. Data Analytics Applications.
3. Gaming Applications.
4. Scientific Applications.
5. Task Automations.
6. Network Applications.
7. Animation Applications.
8. Test Cases Applications & so on.....

Simply Python can be used to make games, do data analysis
control robot, hardware, create GUIs or even to create
websites.

Advantages & Disadvantages of Python:

Advantages:-

- Open source - free and can edit source code.
- Dynamically typed - No need to specify the type of variable before/after using it.
- Interpreted Language - opposite to compiled language.
- Objected oriented language.
- Scripting Language.

Python

→ Indentation - whitespaces, No need of braces. Python Narayana.

→ Scripting Language.

(12)

→ Easy to Learn like normal English.

→ Developed using 'C' Language.

→ Interface to other programming Languages - Can include C, Java, .Net using cython, Jython, Ironpython interface respectively.

→ platform independent - works on multi platforms.

→ Less code Compared to other Languages.

→ No strict typing on Variables & Containers.

→ Used by lot many firms and worlds top most organizations like Google, NASA, MST and Gaming apps etc.

→ Huge library.

→ Multi-threading & Multi processing.

Disadvantages:

→ slow compared to other programming Languages like C, C++ (or) Java.

→ Threading not fully implemented.

→ All Strings are not unicode by default.

→ Indentation - if mix tabs & spaces.

Python Narayana

Python

→ Not using for mobile Applications. Python Narayana

(13)

some other points about python:

1. python is a general-purpose Language.
2. It has wide range of applications from web development, scientific and mathematical computing to develop desktop graphical user interfaces (Pygame, panda3D).
3. The syntax of the language is clean and length of the code is relatively short. It's fun to work in python because it allows you to think about the problem rather than focusing on the syntax.

why should we learn python?

There are four reasons to learn python,

1. Very Simple Syntax.

python programming is very fun, it is very is to understand & write the python code. The syntax feels very natural.

Eg:-

$$a = 10$$

$$b = 20$$

$$\text{sum} = a + b$$

print(sum).

Python Narayana

Python
So here if you even don't have any python knowledge before, we can easily guess that this program is adding two numbers and print result. (14)

2. Not Overly Strict:-

- you don't need to define the type of variable in python. Also, it's not necessary to add semicolon at the end of the statement.
- python enforces you to follow good practices. These small things can make learning much easier for beginners.

3. Expressiveness of the language:-

- python allows you to write programs having greater functionality with fewer lines of code.
- If we write 1000 lines of code by using any programming language, the same functionality can be achieved in python with just 20% to 30% code of that programming language. This is just an example, you will be amazed how much you can do with python once you learn the basics.

Python Narayan

4. Great Community & Support

Python Narayana

(15)

python has Large supporting community. These are numerous active forums online which can be handy if you are stuck. Some of them are:

- a. Learn python
- b. Google forum
- c. python questions - Stack overflow.

How python's datatypes Compare to Other Languages?

Statically typed languages A Language in which types are fixed at compile time. Most statically typed Languages enforce this by requiring, you to declare all variables with their data types before using them. Java and C are statically typed languages.

Dynamically typed Languages A Language in which types are discovered at execution time, the opposite of statically typed. VB Script and python are dynamically typed; because they figure out what type a variable is when you first assign it a value.

python
Narayana

python

strong typed Language A Language in which types are always enforced. Java and python are strongly typed. If you have an integer, you can't treat it like a string without explicitly converting it.

weakly typed Language: A Language in which types may be ignored, the opposite of strongly typed: VBscript is weakly typed.

→ In VBscript, you can concatenate the string '12' and integer 3 to get the string '123' then treat that as the integer 123, all without any explicit conversion.

So python is both dynamically typed (because it doesn't use explicit datatype declarations) and strongly typed. (because once a variable has a datatype, it actually matters).

Python Application (81) program development

→ python is considered an interpreted language because python programs are executed by an interpreter. There are two ways to use the interpreter.

1. Interactive mode.
2. Script mode

python

Interactive mode: command Line shell, we type python programs and the interpreter prints the result.

>>> 10+20

30.

→ The chevron, >>> is the prompt, that is used by interpreter to indicate that it is ready, so that's why when we enter 10+20 and click enter then immediately interpreter return 30.

→ This interactive mode is not used to develop any business applications. It is just used to test the feature of python.

→ In script mode, we write python program in a file and use the interpreter to execute the content of that file. The extension of this file is .py

→ Here we can write our python program in any of the editors (SI) IDE's

→ Different editors used for python programs are notepad, Notepad++, Editplus, Editplus, nano, gedit, IDL...

Different IDE's used for python program are pycharm, Erid, Eclipse, pycscript, Net beans...

Python
This script mode is used to develop business applications

(18)

Eg:-

Open python → file → New file.

Write your python program in the new file and save the file with an extension .py.

Like.

A = 10

B = 5

print (A+B)

print (A+B).

Save this file with firstprogram.py.

→ Open the file firstprogram.py, click on run tab and then run module or press F5

Now we can see the o/p in python shell.

* python identifiers *

→ A python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or underscore (-) followed by zero or more letters, underscores and digits (0 to 9).

→ Python doesn't allow punctuation characters such as @, \$ and % within identifiers.

python
Python Narayana Language.
→ This Application and application are two different
identifiers in Python. (19)

→ Any identifier starting with two leading underscores indicates a strong private identifier.

1. An identifier should not start with digit but it can contain digits, like.

>>> 12ab = 100 SyntaxError: invalid syntax.

>>> ab12 = 100

2. An identifier can contain underscores (single, double and also triple)

>>> -total = 10

>>> to-tal = 20

>>> total- = 30

>>> -to-tal- = 40

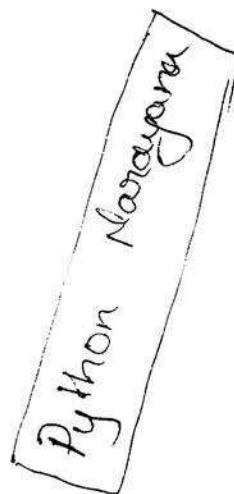
>>> _total = 'python'

>>> total_ = 'Narayana'

>>> to--tal = 'Nani'

>>> --total = 'Santhoshi'

>>> --to--tal- = 100.



3. An identifier must not contain special characters (except underscore).

>>> @total = 100 SyntaxError: invalid syntax

python
 >>> total = 200 }

 SyntaxError: invalid Syntax

(20)

>>> to@tal = 200

SyntaxError: Can't assign to operator

>>> total% = 200

SyntaxError: invalid Syntax.

>>> total! = 10

NameError: name 'total' is not defined

>>> ! total = 10

SyntaxError: invalid Syntax.

>>> total* = 10

NameError: name 'total' is not defined

>>> & to-tal = 200

SyntaxError: invalid Syntax.

>>> * total = 20.

SyntaxError: starred assignment

target must be in a list or tuple

>>> total% = 200. NameError: name 'total' is not defined

4. Identifiers are case-sensitive, so python language is a case sensitive language.

>>> a = 100

>>> A = 200

>>> print(a) 100

>>> print(A) 200

>>> var = 'python'

NameError: name var is not defined.

5. Keywords or reserved words can't be used as identifiers.

>>> if = 100

SyntaxError: invalid Syntax.

```

python
>>> return = 'python'      Syntax Error : invalid Syntax
>>> def = 'dev'           "          "
>>> del = 200.             "          "

```

Keywords

the keywords in python are having some predefined functionality. These keywords may not be used as Constant (or) variable (or) any other identifier names.

If we use help then interpreter displays all list of available keywords for python.

And Assert Break Class Continue Def Del Elif Else Except	Python	Exec finally For From Global if Import In Is Lambda	Not Or Pass. Print. Raise Return Try While With Yield.
---	--------	--	---

python

or

>>> import keyword.
>>> keyword.kwlist.

python Narayana.

(22)

['false', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with',
'yield'].

python contains totally 32 keywords. Out of 33,
only True, False and None are capitalized.

* Indentation *

1. One of the most distinctive features of python is its use of indentation to mark blocks of code.
2. Generally we use curly braces in C, C++ and Java Languages but in python curly braces are not allowed to indicate block of code for class, function definition or flow control. Blocks of code are denoted by indentation.
3. Instead of curly braces {}, indentations are used to represents a block.

python
Im-a-parents:

X Im-a-child:

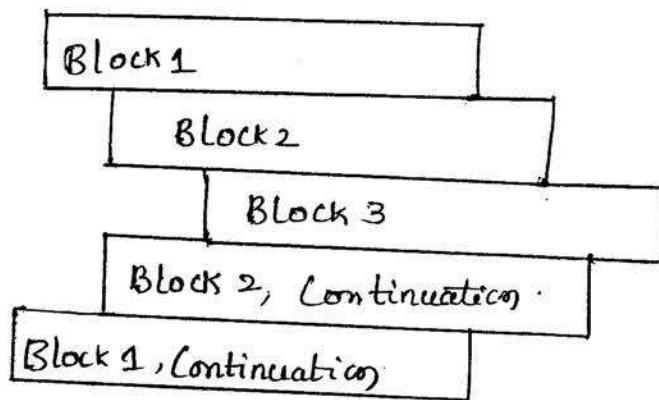
Im-a-grand-child.

Im-another-child:

Im-another-grand-child :

python Narayana

(23)



→ The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. Both blocks in this example are fine:

If True:

```
    print ("This Condition is True")
```

else:

```
    print ("This Condition is False")
```

However, the second block in this eg will generate an error because first print in the else part is not following indentation:

python

Python Narayana

(84)

if True:

 print ("Hello, python programmer")

 print ("This condition is True")

else:

 print ("Hello, python programmer")

 print ("This condition is false").

Example:

a = 10

b = 20

if a>b:

 print ('Hi')

 print (a)

 print ('you go a, right?')

 if a==b:

 print (a)

 print (b)

 print ('both are equal')

 if a<5:

 print (a)

 print ('value of a is Less than 5')

else:

 print (a, 'and', b, 'are not equal')

 print ('different values')

else:

 print ('hello')

 print (b)

 print ('byee...')

 print ('Thank you---!') {

hello

20

bye..

Thank you :- !

* Multi-line statements *

→ Statements in python typically end with a newline.
python does however, allow the use of the line continuation character (\) to denote that the line should continue for eg.

total = item-one

· item-two + \n

item-three.

→ statements contained within the [], {}, () brackets do not need use the line continuation character. For eg,

days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'].

* Quotations in Python *

→ python accepts single ('), double ("") and triple ("\" or "") quotes to denote string literals, as long as the same type of quote starts and ends the string.

Python
→ Generally triple quotes are used to write the string across multiple lines. For example, all the following are legal:

(26)

word = 'word'

Sentence = """this is a sentence".

paragraph = """This is a paragraph. It is made up of multiple lines & sentences."""

* Comments in python *

→ Comments are used in programming to describe the purpose of the code. This helps you as well as other programmers to understand the intent of the code.

→ Comments are completely ignored by compilers & interpreters.

→ A hash sign (#) that is not possib' inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the python interpreter ignores them.

#!/usr/bin/python.

first comment.

Python Narayana

Python
print "Hello, python!"; #second comment.

(27)

Python Narayana

This will produce following result:

Hello, python!

A comment may be on the same line after a statement or Expression:

name = "parlapalli" # declaring variable name
you can comment multiple lines as follows.

#python is easy.

#python is simple, too.

#python is powerful, too

These all are comments.

Python Narayana

Some of the basic functions:

Python supports so many functions. Some of those are, print(), type(), id()

Print(): This function displays the output on the Screen.

Type(): This function checks the data type of Specific Variable.

Id(): This function finds address of Variable.

* python - Variables *

(28)

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of variable, the interpreter memory and decides what can be stored in the reserved memory. Therefore ~~by~~ ^{By} assigning different types to variables, ~~you can~~ ^{you can} store integers, decimals or characters in these variables.
- We use these ~~variable~~ ^{for} names to make program code read than ~~like~~ English. If we didn't use good names for things in our software, we would get lost when we tried to read our code again.

* Assigning values to Variables *

We can assign single value to single variable or multiple variables at a time. And also we can assign single value to multiple variables.

Assigning single value to single variable

The operand to the left of the = operator is the variable, and the operand to the right of the = operator is the value stored in the variable.

```
python  
>>> m=10  
  
>>> print(m)  
  
>>> type(m)  
  
>>> id(m)
```

```
>>> n = 20.4
```

```
>>> print(n)
```

```
>>> type(n)
```

```
>>> id(n)
```

```
>>> r = "Narayana"
```

```
>>> print(r)
```

```
>>> type(r)
```

```
>>> id(r)
```

Assigning multiple values to multiple variables:-

```
>>> a,b,c = 10, 10.5, 'satya'
```

```
>>> print(a)
```

```
>>> type(a)
```

```
>>> id(a)
```

10

<class 'int'>

1624729872

```
>>> print(b)
```

10.5

```
>>> type(b)
```

<class 'float'>

```
>>> id(b)
```

49688512

```
>>> print(c)
```

Satya

```
>>> type(c)
```

<class 'str'>

```
>>> id(c)
```

54580890

Python Narayana

(29)

10

<class 'int'>

1624729872.

-20.4

<class 'float'>

48638592.

Narayana.

<class 'str'>

54591690.

Python Narayana

Python Narayana

python Assigning same value to multiple variables Python Narayana

(30)

```
>>> x = y = z = 10  
>>> print(x) 10  
>>> print(y) 10  
>>> print(z) 10
```

Some Examples on variables:

Eg:1

```
my-name = 'Ranuya'  
my-age = 28 # not a lie.  
my-height = 74 # inches.  
my-weight = 75 # kgs.  
my-eyes = 'Blue'.  
my-teeth = 'white'  
my-hair = 'black'  
print("Let's talk about %.s." % my-name).  
print("She's %.d inches tall." % my-height)  
print("She's %.d kgs heavy." % my-weight)  
print("Actually that's not too heavy").  
print("She got %.s eyes and %.s hair." % (my-eyes,  
                                              my-hair)).  
print("Her teeth are usually %.s depending on  
the coffee." % my-teeth)  
# this line is tricky, try to get it exactly  
right.
```

```
python  
print("Hi this is %s, if I add %.d %.d, and  
%.d I get %.d." % (my-name, my-age, my-height,  
my-weight, my-age + my-height + my-weight)).
```

Output

Let's talk about Ranuya.

She's 74 inches tall.

She's 75 kgs heavy.

Actually that's not too heavy.

She's got Blue eyes and black hair.

Her teeth are usually white depending on the coffee

Hi this is Ranuya, if I add 28, 74, and 75 I get

177.

Eg:2%

Cars = 100.

Space-in-a-Car = 4'0

drivers = 30.

passengers = 90.

car-not-driven = Cars-drivers

Car-driven = drivers.

Carpool-Capacity = Cars-driven * Space-in-a-Car.

Average-passengers-per-car = passengers / car-driven

```
print("There are", cars, "cars available")
```

python

print("There are only", drivers, "drivers available").
Print ("There will be", cars-not-driven, "empty
cars today").

Point ("we can transport", carpool-capacity, "people
today")

Print ("We have", passengers, "to carpool today")

Printf ("We need to put about", average-passengers-per-
car, "in each car").

Output:

There are 100 cars available

There are only 30 drivers available

There will be 70 empty cars today

We can transport 120.0 people today.

We have 90 to carpool to today.

We need to put about 3.0 in each car.

Eg38-

print("Mary had a little lamb").

print ("its fleece was white as 'tis 'snow')

print ("And everywhere that Mary went")

print ("*10) # what'd that do?

end1 = "c"

end2 = "h"

end3 = "e"

end4 = "e"

python
end5 = "s"
end6 = "e"
end7 = "B"
end8 = "U"
end9 = "x"
end10 = "g"
end11 = "e"
end12 = "r"

Python Narayana

(32)

watch that comma at the end. try removing it to
see what happens.

```
print(end1 + end2 + end3 + end4 + end5 + end6,  
      )  
print(end7 + end8 + end9 + end10 + end11 + end12)
```

Output:

Mary had a little lamb

Its fleece was white as snow

Anywhere that Mary went

cheese

Burger.

Eg:4

days = "Mon Tue Wed Thu Fri Sat Sun"

months = "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"

```
print("Here are the days:", days)
```

```
print("Here are the months:", months)
```

python
Output:

python Narayana,

(34)

Here are the days: Mon Tue wed Thu Fri Sat Sun

Here are the months : Jan

Feb

~~Feb~~

Mar

Apr

May

Jun

Jul

Aug.

Escape Sequences:

this is the list of all the escape sequences python supports. you may not use many of these, but memorize their formats and what they do anyway. Also try them out in some strings to see if you can make them work.

Escape

what it does

\ \

Backslash (\)

\ '

single quote (')

\ "

Double-quote ("")

\ a

ASCII bell (BEL)

\ b

ASCII backspace (BS)

\ f

ASCII formfeed (FF)

\ n

ASCII Linefeed (LF)

\ N

{name} character named name
in the unicode database (unicode only)

python		Python Narayana
\r		ASCII carriage return (CR).
\t		ASCII horizontal tab (TAB) (35)
\uxxxx		character with 16-bit hex value xxxx (unicode only)
\Uxxxxxxxxx		character with 32-bit hex value xxxxxxxx (unicode only)
\v		ASCII vertical tab (VT)
\ooo		character with octal value oo.
\xhh		character with hex value hh.

* First python program *

However, python is one of the easiest languages to learn, and creating "Hello, python Developer!" program is as simple as writing `print ("Hello, python Developer!")`. So, we are going to write a different program.

program to Add Two Numbers

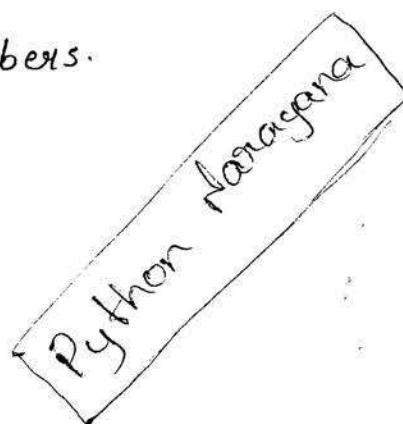
```
>>> # Add two Numbers.
```

```
>>> x = 10
```

```
>>> y = 20.
```

```
>>> sum = x+y
```

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



python

How this program executes?

python Narayana

(36)

Line 1: # Add two numbers.

Any line starting with # in python programming
is a comment.

Line 2: x=10

Here, x is a variable, you can store a value in
a variable. Here, 10 is stored in this variable.

Line 3: y=20.

Similarly, 20 stored in y variable.

Line 4: sum=x+y.

The variables x and y are added using + operator.
the result of addition is then stored in another
variable sum.

Line 5: print(sum)

the print() function prints the o/p to the screen
In Our Case, it prints 30 on the screen.

Python narayana

python

* Reading data from keyboard * (37)

- 1' Input to the program can come in various ways, for example from a database, from another computer, from mouse clicks and movements (81) from the internet.
2. generally in most cases the input comes from the keyboard. For this purpose, ^{python} provides the function `input()`
3. The input of the user will be returned as a string without any changes.
4. If this ~~string~~ input has to be transformed into data type needed by the algorithm, we can use either Casting (typecasting) functions (or eval()).

Eg:-

Direct input from user (default datatype is string):

```
>>> n = input('Enter number: ')      Enter number: 10
>>> print(n)                      10
>>> type(n)                      <class 'str'>
```

python
Converting type data by using casting function python Narayana

(38)

>>> n = int(input('Enter Number')) Enter Number: 10

>>> print(n) 10

>>> type(n) <class 'int'>

Converting type of data by using eval function

>>> n1 = eval(input('Enter Number')) Enter Number: 10

>>> print(n1) 10

>>> type(n1) <class 'int'>

Eg: In Interactive mode

>>> name = input('What is your Name?')

What is your Name: Narayana.

>>> loc = input("What is your Location?")

What is your location : Guntur.

>>> print("""Hello"""+Name+", How are you. "+loc+" How are you.")

Hello Narayana, How are you, How are you Guntur.

In Script mode

Open file and write the following code & save the file

name = input("What is your Name").

Python
loc = input("what is your location:") Python Narayana
print ('Hello'+name+,' How are you. How is your '+loc') (39)

If we run file, we can see the following in the python screen:

what is your Name : Narayana.

what is your Location: Guntur.

Hello Narayana, How are you, How is your Guntur.

Difference b/w python 2 and python 3.

→ In python 2 has two versions of input functions, input() and raw_input(). The input() function treats the received data as string if it is included in quotes "or", otherwise the data is treated as number.

→ In python 3, raw_input() function is deprecated. further, the received data is always treated as string.

In Python 2:

>>> x = input('Enter any values:')

'Enter any value:10'

>>>x

'10'

entered data is treated as string.

python

```
>>> x = input('Enter any value')  
'Enter any value: 100'  
(40)  
>>> x  
100  
# entered data is treated  
as number.
```

```
>>> x = raw_input("Enter any value:")
```

```
Enter any value: 10 # entered data is treated as  
>>> x  
string even without  
'10'
```

```
>>> x = raw_input("Enter any value:")
```

```
'Enter any value: '10' # entered data treated as  
>>> x  
string including"  
    "10"
```

In python 3:

```
>>> x = input("Enter any value")
```

```
'Enter any value: 10'
```

```
>>> x
```

```
'10'
```

```
>>> x = input("Enter any value:")
```

```
'Enter any value: '10' # entered data treated as  
string with (S) without',
```

```
>>> x
```

```
" "10" "
```

```
>>> x = raw_input("Enter any value") # will result Name  
Error
```

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

```
File "", Line 1, in
```

```
x = raw_input("Enter any value")
```

```
NameError: name 'raw_input' is not defined.
```

python * Input And Output StatementS *

Reading dynamic input from the keyboard (41)

In python 2 the following 2 functions are available to read dynamic input from the keyboard.

1. raw-input()

2. input()

1. raw_input() This function always read the data from the keyboard in the form of string format. We have to convert that string type to our required type by using the corresponding type casting methods.

Eg:- `x = raw_input ("Enter first number: ")`

It will always print str type only for any input type.

2. Input()

Input() function can be used to read data directly in our required format. we are not required to perform type casting.

`a = input ("Enter value")`

`type(a)`

python

$10 == \Rightarrow \text{int}$

"durga" == $\Rightarrow \text{str.}$

10.5 == $\Rightarrow \text{float.}$

True == $\Rightarrow \text{bool.}$

python Narayana

(42)

Note: But in python 3 we have only `input()` method
and `raw_input()` method is not available.

python 3 `input()` function behaviour exactly same
as `raw_input()` method of python 2, i.e., every
`input()` function of python 2 is renamed as `input()`
in python 3.

Eg:- 1. `>>> type(input("Enter value:"))`

2. Enter value: 10

3. <class 'str'>

4.

5. Enter value: 10.5

6. <class 'str'>

7.

8. Enter value: True

9. <class 'str'>

Q. write a program to read 2 numbers from the
Keyboard & print Sum.



Python

1. $x = \text{input}(\text{"Enter First Number: "})$ Python Narayana (13)
 2. $y = \text{input}(\text{"Enter Second Number: ")}$
 3. $i = \text{int}(x)$
 4. $j = \text{int}(y)$
 5. $\text{print}(\text{"The sum: "}, i+j)$
 - 6.
 7. Enter First Number : 100.
 8. Enter Second Number: 200.
 9. the Sum: 300:
-

1. $x = \text{int}(\text{input}(\text{"Enter first Number: "}))$
 2. $y = \text{int}(\text{input}(\text{"Enter Second Number: "}))$
 3. $\text{print}(\text{"The sum: "}, x+y)$
-

1. $\text{print}(\text{"The sum: "}, \text{int}(\text{input}(\text{"Enter first Number: "})) + \text{int}(\text{input}(\text{"Enter second Number: ")))$

Q. Write a program to read employee data from the keyword and print that data.

1. $eno = \text{int}(\text{input}(\text{"Enter Employee No: "}))$
2. $ename = \text{input}(\text{"Enter Employee Name: "})$
3. $esal = \text{float}(\text{input}(\text{"Enter Employee Salary: "}))$

Python

4. eaddr = input ("Enter Employee Address: ")
5. married = bool (input ("Employee Married ? [True/False]: "))
6. print ("please confirm information")
7. print ('Employee No:', ero)
8. print ("Employee Name : ", ename)
9. print ('Employee Salary : ', esal)
10. print ("Employee Address : ", eaddr)
11. print ("Employee Married ? : " married)
12.
13. D:\python-classes>py test.py
14. Enter Employee No : 100.
15. Enter Employee Name : Sunny.
16. Enter Employee Salary : Mumbai 1000.
17. Enter Employee Address : Mumbai
18. Employee Married ? [True/False] : True.
19. please confirm Information.
20. Employee Salary NO : 100.
21. Employee Name : Sunny.
22. Employee Salary : 1000.0
23. Employee Address : Mumbai
24. Employee Married ? : True.

Python Narayana

(44)

Python
How to read multiple values from the keyboard in a single line :-

(45)

1. `a,b = [int(x) for x in input("Enter 2 numbers:").split()]`
2. `print("product is:", a*b)`
- 3.
4. `D:\python-classes>py test.py`.
5. Enter 2 numbers: 10 20
product is : 200.

Note:- `split()` function can take space as separator by default. But we can pass anything as separator.

- Q. Write a program to read 3 float numbers from the keyboard with, separator and print their sum.

1. `a,b,c = [float(x) for x in input("Enter 3 float numbers:").split(',')]`
2. `print("The sum is:", a+b+c)`

- 3.
4. `D:\python-classes>py test.py`.
5. Enter 3 float numbers : 10.5, 20.6, 20.1
6. the sum is : 51.2

python eval() :-

python Narayana

(46)

eval function take a string and evaluate its result.

Eg:- $x = \text{eval}("10+20+30")$

print(x)

O/p:- 60.

Eg:- $x = \text{eval}(\text{input}("Enter Expression"))$

Enter Expression: $10+2*3/4$.

Output 11.5.

eval() can evaluate the input to list, tuple, set based the provided input.

Eg:- 1. $l = \text{eval}(\text{input}("Enter List"))$

2. $\text{print}(\text{type}(l))$.

3. $\text{print}(l)$

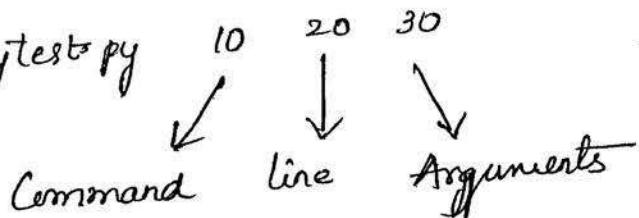
Python Narayana

Python * Command Line Arguments Python Narayan

(H7)

- argv is not Array It is a list. It is available sys module.
- the Argument which are passing at the time of execution are called Command line Arguments.

Eg:- D:\python-classes\py-test.py



→ within the python program this command line Arguments are available in argv. which is present in sys module.

test.py	10	20	30
---------	----	----	----

Note:- argv[0] represents Name of program. But not first Command line Argument.

argv[1] represent first Command line Argument.

Program To check type of argv from sys.

```
import argv  
print(type(argv))
```

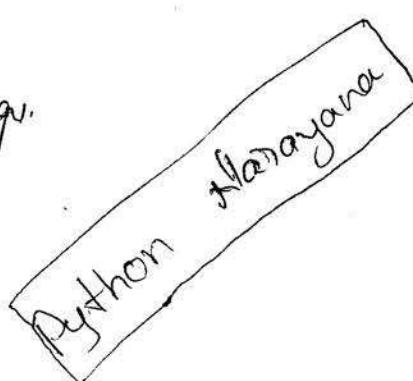
D:\python-classes\py-test.py.

python write a program to display Command Line Arguments

(48)

1. from sys import argv.
2. print ("The number of Command line Arguments:", len(argv))
3. print ("The List of Command line Arguments:", argv)
4. print ("Command Line Arguments one by one:")
5. for x in argv:
6. print(x)
- 7.
8. D:\python-classes> pytest.py 10 20 30
9. The Number of Command line Arguments: 4
10. The List of Command line Arguments ['test.py', '10', '20', '30']
11. command line arguments one by one:
12. test.py
13. 10
14. 20
15. 30.

1. from sys import argv.
2. Sum = 0.
3. args = argv[1:]
4. for x in args:
5. n = int(x)
6. Sum = Sum + n
7. print ("The sum:", Sum)



Python

8.

Python Narayana

(49)

9. D:\python_classes>pytest.py 10 20 30 40

10. The sum 100.

Note:1 Usually space is separator between command line arguments. If our command line argument itself contains space that we should enclose within double quotes (but not single quotes)

Eg:-

1. from sys import argv.

2. print(argv[1])

3.
4. D:\python_classes>pytest.py Narayana python Narayana

5. python

6. D:\python_classes>pytest.py 'Python Narayana'

7.
8. Python

9.

10. D:\python_classes>pytest.py "Python Narayana".

11. Python Narayana.

Note:2 Within the python program command line arguments are available in the string form. Based on our requirement, we can convert into corresponding type by using type casting methods.

python

- Eg:-
1. from sys import argv
 2. print(argv[1] + argv[2])
 3. print(int(argv[1]) + int(argv[2]))
 - 4.
 5. D:\python-classes> py test.py 10 20
 6. 1020
 7. 30

python Narayana

(50)

Note 3 If we are trying to access Command line arguments with out of range index then we will get Error.

Eg:-

1. from sys import argv
2. print(argv[100])
- 3.
4. D:\python-classes> py test.py 10 20
- 5) IndexError : List index out of range.

Note:- In python there is argparse module to parse Command line arguments & display some help messages whenever end user enters ~~too~~ wrong input.

input()

raw_input()

Command line arguments.

Output Statement

We can use print() function to display output

Form1: print() without any argument
just it prints new line character.

Form2:

1. print(string):
2. print('Hello world')
3. We can use escape characters also.
4. print ("Hello\n world")
5. print ("Hello\t world")
6. We can use repetition operator (*) in the string
7. print (10 * "Hello")
8. print ("Hello" * 10)
9. We can use + operator also
10. print ("Hello" + "world").

Note:

→ If both arguments are string type then + operator acts as Concatenation operator.

→ If one argument is string type & second is any other type like int then we will get error.

→ If both arguments are number type then + operator acts as arithmetic addition operator

Python

1. `print("Hello" + "World")`
2. `print("Hello", "World")`
- 3.

4. `HelloWorld.`

5. `Hello World.`

python Narayana

(52)

Form 3: `print()` with variable number of arguments

1. `a, b, c = 10, 20, 30`

2. `print("The values are:", a, b, c)`

3.

4. Output the values are 10 20 30

By default output values are separated by space. If we want we can specify separator by using "sep" attribute.

1. `a, b, c = 10, 20, 30`

2. `print(a, b, c, sep=',')`

3. `print(a, b, c, sep=':')`

4.

5. `D:\python-classes>py test.py`

6. - 10, 20, 30

T 10 : 20 : 30.

Form 4: `print()` with end attribute:

1. `print("Hello")`

O/P: 1. Hello

2. `print("Python")`

2. Python

3. `print("Narayana")`

3. Narayana

If we want O/P in the same line with space.

Python
1. print("Hello", end=' ')
2. print("Python", end=' ')
3. print("Narayana")

python Narayana
(53)

O/p: Hello Python Narayana.

Note: The default value for end attribute is \n, which is nothing but new line character.

Form-5: print(object) statement:

We can pass any object (like list, tuple, set etc) as argument to the print() statement.

Eg:- 1. l = [10, 20, 30, 40]

2. t = (10, 20, 30, 40)

3. print(l)

4. print(t)

Form-6: print(String, Variable list):

We can use print() statement with string and any number of arguments.

Eg:- 1. s = "Narayana"

2. a = 28

3. s1 = "python"

4. s2 = "Django"

5. print("Hello", s, "Your Age is", a)

6. print("You are teaching", s1, "and", s2)

Python O/p

python Narayana

(54)

1. Hello Narayana your age is 28
2. You are teaching python and Django.

Form-7 print (formatted String):

%i ==> int

%d ==> int

%f ==> float

%s ==> String type.

Eg: 1. s = "Narayana"

2. list = [10, 20, 30, 40]

3. print ("Hello %s... the list of items are %s", %s, list))

#

5. Output Hello Narayana... the list of items are [10, 20, 30, 40]

Form-8 print () with replacement operator

1. name = "Durge"

2. Salary = 10000

3. gf = "Sunny"

4. print ("Hello %s salary is %d and your friend %s is waiting."
format (name, salary, gf))

6.

7. output

8. Hello Durge your salary is 10000 and your friend
Sunny is waiting.

* Data types in python *

- 1. Data types are some of the keywords in any programming languages.
- 2. Data types are used to define what type of data that we are passing to the specific variable.
- 3. Generally variables with ~~not~~ stores the values without having ~~datatype~~ in other languages.
- 4. But in python language, variables allow values without specifying data types also because python allocates the data types dynamically at runtime when python application is running.
- 5. In python language, data types are decided by interpreter based on the value that is given to Variable by user.
- 6. Programmers should not specify the data type of Variable when developing the program, If we specify explicitly, then interpreter will throw error

python

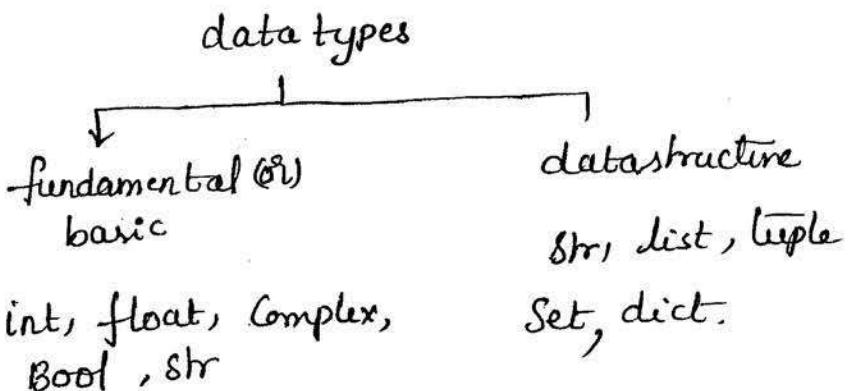
python supports two types of data types: Python, Narayana
56

1. Basic data types or fundamental data types.
2. composite datatypes (or) collection datatypes.

Basic data types or fundamental data types:

python supports different of basic data types,

1. int
2. float
3. Complex
4. Bool.
5. Str.



Int datatype

If we assign integer value to any variable then interpreter decides that variable as int type

Variable.

Eg:-

```
>>> a = 10
>>> print(a)
10
>>> type(a)
<class 'int'>
>>> id(a)
1625909520.
```

Float Data types

If we assign float value to any variable then interpreter decides that variables as float type variable.

```
>>> b = 10.5
>>> print(b)
10.5
```

```
python  
>>> type(b)  
>>> id(b)
```

```
python Narayana  
<class 'float'>  
47460096. (57)
```

String Data types

If we assign string value to any variable then interpreter decides that variable as string type variable

Ex:-

```
>>> c = 'Narayana'  
>>> print(c) Narayana  
>>> type(c) <class 'str'>  
>>> id(c) 52367056.
```

Bool Data types

If we assign either True or false to any variable then interpreter decides that variable as Bool type variable.

Ex:-

```
>>> d = True  
>>> print(d) True  
>>> type(d) <class 'bool'>  
>>> id(d) 1625727600
```

Complex Data types

If we assign complex number to any variable then interpreter makes that variable as Complex type variable

Python
Eg:-

```
>>> e = 3+4j
```

```
>>> print(e)
```

(3+4j)

```
>>> type(e)
```

< class 'complex' >

```
>>> id(e)
```

47539552

python Narayana

(58)

* Type Conversion functions *

We can convert datatypes into two different types:

1. type casting functions
2. eval function

Type casting functions

These type conversion functions are used to convert string type data into required types.

1. int(): This int() function is used to convert into int data type.

Eg:- Get two integers from the user and perform addition on those user values.

Way 1 :-

```
>>> a = input ("Enter first Number: ") Enter first no: 10
```

```
>>> b = input ("Enter Second Number: ") Enter second no: 2
```

```
>>> print(a) 10
```

```
>>> print(b) 20
```

```
>>> type(b) < class 'str' >
```

```
python  
>>> type(a)  
>>> id(a)  
>>> id(b)  
  
>>> c=a+b  
  
>>> print(c)  
>>> type(c)  
>>> id(c)
```

```
>>> x=int(a)  
>>> y= int(b)  
  
>>> print(x)  
>>> print(y)  
>>> type(x)  
>>> type(y)  
  
>>> id(x)  
>>> id(y)
```

>

```
>>> z=x+y  
>>> print(z)  
>>> type(z)  
>>> id(z)
```

python Narayana
<class 'str'>
52351712
629536.

(59)

adding two str variable
1020
< class 'str'>
5235116

converting str 'a' into int
converting str 'b' into int 'y'

10
20
< class 'int'>
< class 'int'>

1625909520

1625909680

adding two int variables.

30
< class 'int'>
1625909840.

python
way-2 (shorter way):

```
a = int(input("Enter first Number:"))      Enter first number: 10
>>> b = int(input("Enter Second number:"))   Enter Second no: 20
>>> print(a)                                10
>>> print(b)                                20
>>> type(a)                                 <class 'int'>
>>> type(b)                                 <class 'int'>
>>> c = a+b
>>> print(c)                                30
>>> type(c)                                 <class 'int'>
```

way-3 (shortest way):

```
>>> print("The sum is "+str(int(input("Enter first
numbers:"))+int(input("Enter second numbers:"))))
```

```
>>> Enter first number: 10
>>> Enter second number: 20
>>> the sum is 30.
```

2. float(): This float() conversion function is used to convert other types into float type

ways:

```
>>> a = input("Enter first number: ")    Enter first: 10.5
>>> b = input("Enter second number: ")   Enter Second Number
20.5
```

python Narayana
⑥

python

```
>>> print(a)  
>>> print(b)  
>>> type(a)  
>>> type(b)  
>>> id(a)  
>>> id(b)
```

python Narayana

10.5

(61)

20.5

<class 'str'>

<class 'str'>

52354240

53354144

```
>>> c=a+b  
>>> print(c)  
>>> type(c)  
>>> id(c)
```

10.520.5

<class 'str'>

52367096.

```
>>> x = float(a)  
>>> y = float(b)  
>>> print(x)  
>>> print(y)  
>>> print type(x)  
>>> type(y)  
>>> id(x)  
>>> id(y)
```

10.5

20.5

<class 'float'>

<class 'float'>

47459172

47460096

```
>>> z = x+y  
>>> print(z)  
>>> type(z)  
>>> id(z)
```

31.0

<class 'float'>

46410368.

Python way 2 (shorter way):-

```
>>> a = float(input("Enter first Number:")) Enter first no: 10.5  
>>> b = float(input("Enter second number")) Enter Second number 20.3  
>>> print(a) 10.5  
>>> print(b) 20.3  
>>> type(a) <class 'float'>  
>>> type(b) <class 'float'>  
  
>>> c = a + b  
  
>>> print(c) 30.8  
>>> type(c) <class 'float'>
```

Way 3 (shortest way):-

```
>>> print(float(input("Enter first number:")) + float(  
    input("Enter second Number:")))  
>>> Enter first number: 10.5  
>>> Enter Second number: 20.5  
>>> 31.0
```

3) Complex () :- This complex() conversion function is used to convert string type data into complex type

Way 1:

```
>>> a = input("Enter Number: ") Enter Number: 2+3j  
>>> print(a) 2+3j.
```

Python Narayana

(6)

python
>>> type(a)
<class 'str'>
Narayana
(62)

>>> x = complex(a)
>>> print(x) (2+3j)
>>> type(x) <class 'complex'>
>>> id(x) 47539168.

way:2

```
>>> print( complex( input("Enter Number:")))  
>>> Enter Number: 4+6j  
>>> (4+6j)
```

4. Bool(): This bool conversion fn is ~~not~~ used to convert string type data into Boolean type

way t:-

```
>>> a = input("Enter either True (or) False")
Enter either True (or) False
>>> print(a)
True
>>> type(a)
<class 'str'>
>>> id(a)
52354242.
>>> x = bool(a)
>>> print(x)
True
>>> type(x)
<class 'bool'>
>>> id(x)
1625727600.
```

python
ways

```
>>> print(bool(input("Enter Either True or False: ")))  
>>> Enter Either True or False: True.  
>>> True.
```

Python Narayana
(64)

* Eval function *

This function is also used to convert the data type into required type. But generally this function converts the type of the variable based on the value.

Converting int values

```
>>> a = eval(input('Enter value for a:'))  
Enter value for a: 10
```

```
>>> print(a) 10  
>>> type(a) <class 'int'>
```

Converting float values

```
>>> b = eval(input('Enter value for b:'))  
Enter value of b: 10.6
```

```
>>> print(b) 10.6  
>>> type(b) <class 'float'>
```

Python Converting string Value:

```
>>> c = eval(input("Enter value for c:"))
```

enter value for c: 'Narayana'

```
>>> print(c)
```

Narayana

```
>>> type(c)
```

<class 'str'>

Converting Complex Value:

```
>>> d = eval(input('Enter value for d:'))
```

enter value for d: 2+5j

```
>>> print(d)
```

(2+5j)

```
>>> type(d)
```

<class 'complex'>

Converting bool Value:

```
>>> e = eval(input('Enter value for e:'))
```

enter value for e: True

```
>>> print(e)
```

True

```
>>> type(e)
```

<class 'bool'>

Python Narayana

Python

python Narayana
66

Narayana
Python

Python

Assignment - 1 python Naraef ^{one}(67)

1. Who is the father of python?

A.

2. What is the language used to develop distributed operating system?

A.

3. Why did Guido Van Rossum keep the name 'python' to his language?

A.

4. When was the python developed?

A

Python

5. When did Python available to public? Python Narayana
⑥8

A

6. What is a Python?

A

7. What are the main features of Python?

A.

8. What are the differences b/w programming & Scripting languages?

A.

9. Is Python programming language or Scripting Language?
Why?

A.

Python
10. what are the different types of languages used to develop the python language? (69)

A.

11. Is python portable Language or not? if yes then why?

A.

12. If we use python language to develop a project, will it take less time or more time to develop the project compare to other computer languages? if no then why?

A.

13. Is python free source or needs to purchase?

A.

python
14. what are the different top Companies using python?

(70)

A.

15. why python is also called cross plat from language?

A

16. what are different types of OSS where we can run python?

A.

O
D

17. why python is also called interpretable Language?

A.

18. why debugging of python program is very easy for developer?

A.

python

python

Narayana

19. Give me any four reasons to learn/work on Python?

(71)

A.

20. Do developers need to focus more on python syntax?
if no then why?

A.

21. Do we need to think about memory management while
application is developing?

A.

22. When memory will be allocated to variable in
Python language?

A.

python

Q3. what are the different types of modes to develop python application?

Pythag Narayana

(72)

A.

Q4. what are the differences b/w interactive mode & Script mode?

A

Q5. what is the name of >>> symbol in python?

A

Q6. what is indentation in python?

A

python
27. what is the variable and what is the purpose of variable in python?

(7.3)

A.

28. who will decide the type of variable in Python Language ? Developer or interpreter ?

A.

29. what are the different types of quotes supported in python?

A.

30. what is the purpose of Comments in any programming language?

A.

31. Python
Can we write our programming code after comment
in same physical line? if no why? (74)

A.

32. Can we write comment after our programming code in
same physical line?

A.

33. How to write comment in multiple lines?

A.

34. What is identifier in python? Can a single identifier
contain combination of upperCase, lowerCase, underscore
& number?

A.

35 Python
Is python case sensitive language or not? If yes, give one example?

(75)

A.

36. Interpreter decides the type of variable in python then how can a programmer know the type of variable?

A.

37 How to see the value in the variable? And also the address of variable in python?

A

38. How to assign multiple heterogeneous values to multiple variables? Give one example?

A

Python
39. How to assign single value to multiple Variables? (76)

A.

40. How to give input to the program from key board?

A.

41. What is the default data type of input()’s value?

A.

42. Can we convert string type data to our required data type? Different ways?

A

43. What are the different types of type conversion fn’s?

A.

Python what is eval()

Python Narayana

(77)

A.

45. what is the difference b/w type conversion functions & eval()?

A.

46. How to read values from user? Give one example?

A

47 what is the purpose of datatype in any language?

A.

48. Can variable allow the value without specifying its type in python Language?

A

49. Can a programmer specify type of variable explicitly while developing application? Python Narayana (78)

A.

50. On what basis, interpreter decides the type variable in python language?

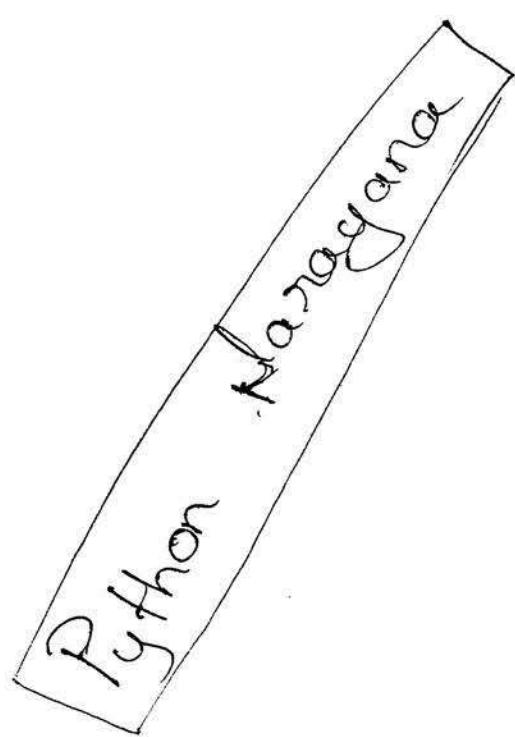
A.

python

Narayana

Python Narayana

(79)



python

Narayana

Python

* Composite Datatypes Python Narayana (or) 80

Collections (or) Data Structures *

python supports different type of data structures.

1. String
2. List
3. Tuple
4. Set
5. Dictionary.

* String - Data Structure * Python

1. A sequence of characters is called string.
2. python supports ~~for~~ data type to represent string type data.
3. String objects are immutable objects that we can't ~~modify~~ modify the existing string object
4. Insertion order is preserved in string objects.
5. A string allows multiple duplicate characters also
6. Every character in the string object is represented with unique index.

python

Python Narayana.

7. python supports both forward and backward
indexes. (81)

8. forward index starts with 0 and negative index
starts with -1

9. python string supports both concatenation & multipli-
cation of string objects.

10. strings can be created by enclosing characters
inside a single quote (or) double quotes. Even
triple quotes can be used in python but generally
used to represent multiline string & docstrings.

Eg: Creating a string with single quotes

```
>>> name = 'Narayana'
```

```
>>> print(name)
```

Narayana.

Eg: Creating a string with double quotes

```
>>> name = "Narayana"
```

```
>>> print(name)
```

Narayana.

Eg: Creating a string with triple quotes.

```
>>> name = """ Narayana """
```

```
>>> print(name)
```

Narayana.

```
Python  
>>> name = "||| Narayana |||"  
Python Narayana  
(82)  
>>> print(name) Narayana.
```

Using triple quotes for multiple line comments

```
name = """Hello, welcome to  
world of Python  
working with python is very fun"""
```

print(name)
Hello, welcome to
world of python
working with python is very fun.

1. We can access individual characters using indexing and a range of characters using slicing.
 2. Index starts from 0. If we try to access a character out of index range then interpreter will raise as IndexError.
 3. The index must be an integer. If we try to access the data with non-integer indexes values then interpreter raises TypeError.
 4. We can also give bool values as index values below interpreter treats True as 1 and False as 0.

String Indexing: It is nothing but fetching a specific character by using its index number

Eg.

0	1	2	3	4	5	6	7
N	A	R	A	Y	A	N	A
-8	-7	-6	-5	-4	-3	-2	-1

```
>>> x = 'narayana'  
>>> print(x)           narayana  
>>> type(x)          <class 'str'>  
>>> id(x)            52418004
```

forward indexes

>>> print(x[0])	n
>>> print(x[1])	a
>>> print(x[2])	r
>>> print(x[3])	a
>>> print(x[4])	y
>>> print(x[5])	a
>>> print(x[6])	n
>>> print(x[7])	a

python allows negative indexing for its sequences.

The index of '-1' refers to the last item, -2 to the second last item and so on.

python

Backward index

```
>>> print(x[-1])
>>> print(x[-2])
>>> print(x[-3])
>>> print(x[-4])
>>> print(x[-5])
>>> print(x[-6])
>>> print(x[-7])
>>> print(x[-8])
```

python Narayana

(84)

a
n
a
y
a
r
a
n.

→ If we modify the content of the existing object then
the indexes also will be changed.

Eg:-

```
>>> x = "narayana"
```

```
>>> print(x)
```

narayana

```
>>> type(x)
```

<class 'str'>

```
>>> id(x)
```

52419584.

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

n

a

g

a

y

a

a

Space is there here, because
x contains Space as
first character.

Python
>>> print(x[7])
>>> print(x[8])

Python Narayana
85

>>> print(x[-1]) a
>>> print(x[-2]) n
>>> print(x[-3]) a
>>> print(x[-4]) y
>>> print(x[-5]) a
>>> print(x[-6]) r
>>> print(x[-7]) a
>>> print(x[-8]) n
>>> print(x[-9])

#space is there here, because
x contains space as first
character.

String slicing:-

We can access a range of characters using
slicing.

The slicing operator is colon (:

Eg:-

0	1	2	3	4	5	6	7
N	A	R	A	Y	A	N	A
-8	-7	-6	-5	-4	-3	-2	-1

Python
```>>> a = 'NARAYANA'

Python Narayana

|                     |          |
|---------------------|----------|
| >>> print(a[0:1])   | N        |
| >>> print(a[0:2])   | NA       |
| >>> print(a[0:3])   | NAR      |
| >>> print(a[0:4])   | NARA     |
| >>> print(a[0:5])   | NARAY    |
| >>> print(a[0:6])   | NARAYA   |
| >>> print(a[0:7])   | NARAYAN  |
| >>> print(a[0:8])   | NARAYANA |
| >>> print(a[2:8])   | RAYANA.  |
| >>> print(a[2:4])   | RA       |
| >>> print(a[7:8])   | A        |
| >>> print(a[-8:-6]) | NA       |
| >>> print(a[-8:-4]) | NARA.    |
| >>> print(a[-8:-3]) | NARAY    |
| >>> print(a[-5:-4]) | A.       |

(86)

### String Concatenation

1. We can concatenate two or more strings into

single string is called concatenation.

2. The + operator is used in python for concatenation.

Eg: >>> string1 = 'python'.  
>>> string2 = 'Developer'.

Python  
>>> print('string1+String2:', string1 + string2) Python Narayana  
string1+String2: Python Developer (87)

## String Multiplication:-

1. Python supports multiplying (or) repeating the given string into n number of times.
2. The \* operator can be used to repeat the string for a given number of times.

Eg:-

```
>>> string1 = 'python' python python python
>>> print(string1 * 3)
```

String is immutable object because we can't replace (or) alter the existing string object.

Eg:-

```
>>> st = 'python'
>>> print(st) python.
>>> id(st) 42221972.
```

```
>>> st = st + 's' # altering the string object
>>> print(st) Pythons
>>> id(st) 41800640.
```

<sup>Python Narayana</sup>  
→ id is changed to before modification and after  
modification of the string str that's why string  
is a immutable object. 88

→ Here when we try to edit the existing string  
object (str is 42221492), then the new object (str is  
41200640) is created.

### String packing:

It is nothing but packing all values of defined  
of defined variables as a single String.

```
>>> a = 'x'
>>> b = 'y'
>>> c = 'z'
>>> st = " ".join([a,b,c])
>>> print(st) xyz
>>> type(st) <class 'str'>
```

### String unpacking:

1. String unpacking allows extracting all characters  
of string into different variables automatically
2. The number of variables must be equal to  
number of characters in the string.

Python  
```str1 = "python"```

Python Narayana
89

```print(str1)``` python

```type(str1)``` <class 'str'>

```id(str1)``` 23941472

```a, b, c, d, e, f = str1``` #string unpacking

```print(a)``` p

```type(a)``` <class 'str'>

```print(b)``` y

```type(b)``` <class 'str'>

```print(c)``` t

```type(c)``` <class 'str'>

```print(d)``` h

```type(d)``` <class 'str'>

```print(e)``` o

```type(e)``` <class 'str'>

```print(f)``` n

```print(f)``` <class 'str'>

Python Narayana

Python string functions:-

Python Narayana
⑩

1. Capitalize() :- This function converts first letter of first word in the given string into uppercase.

>>> Str1 = 'python developer'

>>> Str1. Capitalize()

'Python developer'.

2. Title() :- This function first character of each word in the given string into uppercase.

>>> Str1 = 'python developer'

>>> Str1. Title()

'Python Developer'

3. islower() :- This function checks whether the given string contains all lower case letter or not. If all are Lowercase then it will return True else false.

>>> Str1 = 'python developer'

>>> Str1. islower()

True.

>>> Str2 = 'Python'

>>> Str2. islower()

false.

4. isupper() :- This function checks whether the given string contains all uppercase letter or not. If all are Upper Case then it will return True else false.

python
 >>> str1 = 'python developer'
 >>> str1.isupper()

Python Narayana .
 (91)

>>> str3 = 'PYTHON'
>>> [str3.isupper()]

True.

5. Lower(): This function "converts all letters" of given string into "lowercase".

>>> str3 = 'PYTHON'
>>> [str3.lower()]

'python'

6. Upper(): This function "converts all letters" of given string into "uppercase".

>>> str1 = 'python developer'
>>> [str1.upper()]

'PYTHON DEVELOPER'

7. Len(): This function "counts" the number of characters in the given string.

>>> str1 = 'python Developer'
>>> Len(str1)

16

8. Count(): This function "Counts no. of occurrences of a specific character" in a given string

>>> str1 = 'python developer'
>>> str1.count('o')

2.

9. Python
find(): This function "finds the index position of specific character" in the given string. (92)

```
>>> str1 = 'python developer'  
>>> str1.find('o')           4 ... for first occurrence of 'o'  
>>> str1.find('o', 5)       12 ... for second occurrence of 'o'.
```

10. Split(): This function splits the given strings into multiple strings.

```
>>> str1 = 'python developer'  
>>> str1.split()           ['python', 'developer']
```

Note: the default delimiter is space.

```
>>> str2 = 'python developer in Tcs'  
>>> str2.split()           ['python', 'developer', 'in', 'Tcs']
```

```
>>> str3 = 'python-developer.in.Tcs'  
>>> str3.split('.')         ['python', 'developer', 'in', 'Tcs']
```

```
>>> str4 = 'developer'  
>>> str4.split('e')        ['dove', 'oper'].
```

11. strip(): This function removes specific special character to left side of given string.

python
str1 = '!!! python Developer !!!', Python Narayana
(93)
>>> str1.lstrip('!')
'python Developer !!!'

12. rstrip() :- this function removes specific special character to the right side of given string.

str1 = '!!! python Developer !!!'
>>> str1.rstrip('!')
'!!! python Developer'

13. strip() :- this function removes specific special character from both sides to delete the given string.

str1 = '!!! python Developer !!!'
>>> str1.strip('!')
'python Developer'.

14. SwapCase() :- this function swaps all lower case letters into uppercase and vice versa.

>>> str1 = 'pyThOn'
>>> str1.swapcase()
'PYTHON'.

- 15) reversed() :- This function reverses the string

>>> str = 'python'.
>>> print(str)
>>> str = ''.join(reversed(str))
>>> print(str)
nohtyp.
→ two single quotes

Python Narayana

15. replace(): this function replaces an existing character(s) with new character(s) (Q4)

```
>>> str1 = 'python Learner'  
>>> print(str1) Python Learner.  
>>> str2 = str1.replace('Learner', 'developer')  
>>> print(str2) python developer.
```

Note: We can remove any character(s) with non-empty space

```
>>> str1 = 'python'.  
>>> print(str1) python  
>>> type(str1) <class 'str'>  
>>> str2 = str1.replace('thon', '')  
>>> print(str2) py  
>>> type(str2) <class 'str'>.
```

del Command

We can't delete a specific character or range of characters by using del command.

We can delete the entire string objects permanently by using del command.

Python Narayana

Eg:1:-

```
>>> str1 = "python Narayana" # trying to remove specific character.
```

```
>>> del str1[0]           TypeError: 'str' object doesn't support item deletion
```

(95)

Eg:2

```
>>> str1 = "python Narayana" # trying to remove specific range
```

```
>>> del str1[1:5]
```

TypeError: 'str' object doesn't not support item deletion.

Eg:3

```
>>> str1 = "python Narayana".
```

```
>>> print(str1)           python Narayana.
```

```
>>> type(str1)            <class 'str'>
```

```
>>> id(str1)              63806464.
```

```
>>> del str1               # deleting entire string str1 object
```

```
>>> print(str1)            # after deleting.
```

NameError : name 'str1' is not defined.

Python Narayana

Python Narayana
→ How to display the given string in ascending order?

Eg:1
>>> st = "python" (96)

>>> ".join(sorted(st))" 'hnopy'

Eg:2
>>> str1="pyTHON developer" # if string contains both upper and lower cases.

>>> ".join(sorted(st1))" 'ELNOPPTV deehony'.

Q How to display the given String in descending order?

>>> st = "python"

>>> ".join(reversed(sorted(st)))" 'ytponh'

Q How to display the given String with three dots.

between each character.

>>> s1 = 'python'.

>>> s10 = '...'.join(s1)

>>> s10
'P...y...t...h...o...n'

Q How to display the given String with space below each character.

>>> st = 'Narayana'

>>> st = '!'.join(st)

>>> print(st)

N a r a y a n a

Python Narayana
How to reverse a given string?

(97)

```
>>> st = 'hyderabad'  
>>> st3 = '.'.join(reversed(st))  
>>> st3  
'dabredyh'.
```

Q. How to get 'durga' from st = durga soft.

Way 1:-

```
>>> st = 'durga soft'  
>>> st.replace('soft', '')  
'durga'.
```

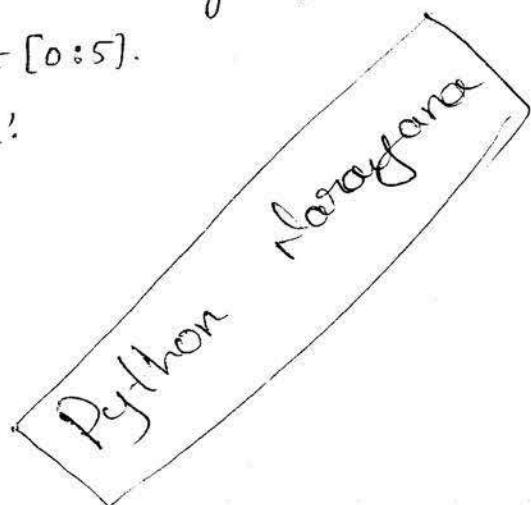
Way 2:-

```
>>> st = 'durga soft'  
>>> a = st.split('s')[0]  
>>> print(a)  
'durga'.
```

Way 3:- >>> st = 'durga soft'

```
>>> st[0:5].
```

'durga'.



Using :: :-

Python Narayana

(98)

```
>>> st = 'python narayana'  
>>> st[::]  
>>> st[0::]  
>>> st[1::]  
>>> st[2::]  
>>> st[12::]  
>>> st[13::]  
>>> st[15::]  
>>> st[0::5]  
>>> st[0::6]  
>>> st[0::10]  
>>> st[0:-10]  
>>> st[0:-16]  
>>> st[18::0]  
>>> st[18::5]  
>>> st[18::10]  
>>> st[18::15]  
>>> st[2::1]  
>>> st[2::7]  
>>> st[2::8]
```

'python narayana'
'python narayana'
'python narayana'
'thon narayana'
'ana'
'na'
"
'pna'
'na narayana'
'na'
'pa'
'py'
'p'
ValueError: slice step cannot be zero.
'yy'
'yy'
'y'
'thon narayana'
'ta'
'ta'

```

>>> st[3::1]          ' har narayana'
>>> st[3::5]          ' har'
>>> st[3::6]          ' hr'
>>> st[3::10]         ' hr'.
>>> st[4::1]          ' on narayana'
>>> st[4::3]          ' onan'.
>>> st[4::4]          ' ooa'          Narayana
>>> st[4::9]          ' naya'        Narayana
>>> st[5::3]          ' na'.          naya
>>> st[5::5]          ' n!'.          na
>>> st[5::10]         ' y'.           n!
>>> st[6::5]          ' '.
>>> st[6::9]          ' han'.        na
>>> st[7::3]          ' '.
>>> st[7::7]          ' na'.        na
>>> st[8::1]          ' y'.
>>> st[0::-1]          ' p'.
>>> st[1::-1]          ' yp'.
>>> st[4::-1]          ' ohtyp'.
>>> st[5::-1]          ' nohtyp'.
>>> st[15::-1]         ' anayaran nohtyp'

```

```
>>> st[6:-2]  
>>> st[7:-2]  
>>> st[9::-2]  
>>> st[13::-2]  
>>> st[-1::-1]  
>>> st[-2::-2]  
>>> st[-3::-3]  
>>> st[-4::-4]  
>>> st[-4::-5]
```

'otp' Python Narayana
'nnhy'. (100)
'rnnhy'.
~~a~~ 'y nnnhy'.
'a'
'n'
'a'.
'ana'.
'y'.

String Comparisons

Eg:1

str1 = 'b'

str2 = 'a'

if str1 > str2:

 print(str1, 'is bigger than', str2)

elif str2 > str1:

 print(str2, 'is bigger than', str1)

else:

 print('both', str1, 'and', str2, 'are equal')

Output:-

b is bigger than a.

Note: In the above, we have set $st1 = 'b'$ and $st2 = 'a'$,
when we see the alphabet in ascending order then
here, 'b' is bigger than 'a' 101

Eg:-2

$st1 = 'a'$

$st2 = 'b'$

if $st1 > st2 :$

 print(st1, 'is bigger than', st2)

elif $st2 > st1 :$

 print(st2, 'is bigger than', st1)

else:

 print("both", st1, "and", st2, "are equal")

Output

b is bigger than a.

Eg:-6

$st1 = 'python'$

$st2 = 'narayana'$

if $st1 > st2 :$

 print(st1, "is bigger than", st2)

elif $st2 > st1 :$

 print(st2, "is bigger than", st1).

else:

 print('both', st1, 'and', st2 'are equal') (102)

python Narayana

Output

python is bigger than narayana.

Note: 'p' in the st1 is bigger than 'n' in the st2 string. So st1 variable value is bigger than st2 variable value.

Eg:

st1 = '123'.

st2 = 'a123'

if st1 > st2:

 print(st1, 'is bigger than', st2)

elif st2 > st1:

 print(st2, 'is bigger than', st1)

else:

 print('both', st1 'and', st2 'are equal').

Python Narayana

* List - Data Structure * Python Narration

1. A list is a collection of elements. These elements may be homogenous (or) heterogeneous.
2. A list also allow duplicate elements.
3. "Insertion" order is preserved in list
4. List elements are separated by commas and enclosed within square brackets []
5. Every element in the list has its own unique index number.
6. List supports both "forward indexing" and "backward indexing", forward index starts from 0 and backward index starts from -1.
7. we access either specific element by using "indexing" or set of elements by using "slicing" from the list
8. we can create list in different ways. like by using `list()` function, by using square brackets "`[]`" and also by using `range()` function.
9. List object are ~~immutable~~.

Creating list by using List():

python Narayana

(OH)

1. the list() allows only one string value with set of character
 2. If we give int type data in the list() function then interpreter will throw "TypeError"
- Eg:1
~~list~~. Eg:1 lst = list([1, 2, 'a'])
 print(lst) = [1, 2, 'a']
 type(lst) <class 'list'>
- ```
>>> list1 = list() # creating empty list
>>> print(list1)
[]
>>> type(list1) <class 'list'>.
```

- Eg:2  
 >>> list1 = list('python') # creating list with set of characters  
 >>> print(list1)
 ['p', 'y', 't', 'h', 'o', 'n']
 >>> type(list1) <class 'list'>.

## Creating list by using square brackets "[]" :-

Eg:1  
 >>> list1 = [] # creating empty list
 >>> print(list1)
 []

Eg:2  
 >>> list1 = [1, 2, 3, 4, 5]. # creating list with homogeneous elements.

>>> print(list1)
 [1, 2, 3, 4, 5].
 >>> type(list1) <class 'list'>.

Eg:3  
 >>> list1 = [10, 11, 'python', 5.5, True, 2+3j] # creating list with heterogeneous elements
 >>> print(list1)
 [10, 11, 'python', 5.5, True, (2+3j)].
 >>> type(list1) <class 'list'>.

## Creating list by using range() function

we can also use range function to create list (105)

Syn: range ( starting index value, Lastvalue, Rangevalue)

Here, both starting index value and Rangevalue are optional.

The default starting index value is '0'.

The default RangeValue is 1.

Eg:-1 In python 2.

```
>>> list = list(range(10))
>>> print(list)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9].
<class 'list'>.
```

Eg:-2

```
>>> list1 = list(range(0,10))
>>> print(list1)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9].
<class 'List'>.
```

Eg:-3

```
>>> list1 = list(range(2,8))
>>> print(list1)
[2, 3, 4, 5, 6, 7]
<class 'List'>.
```

Eg:-4

```
>>> list1 = list(range(-4,4))
>>> print(list1)
[-4, -3, -2, -1, 0, 1, 2, 3].
<class 'List'>.
```

>>> list1 = list(range(0, 10, 2)) Python Narayana  
(166)  
>>> print(list1)  
>>> type(list1)

### Creating a list with split() function

>>> str1 = 'python is very easy and simple Language'.  
>>> lst = str1.split()  
>>> print(lst) ['python', 'is', 'very', 'easy', 'and', 'simple',  
>>> type(lst) 'language'].  
<class 'list'>

### Creating an empty list:

By using []:

>>> lst = [].  
>>> print(lst) []  
>>> type(lst) <class 'list'>.

By using list():

>>> lst1 = list().  
>>> lst1 []  
>>> type(lst1) <class 'list'>.

By using range():

>>> lst2 = list(range(0))  
>>> lst2 []  
>>> type(lst2) <type 'list'>.

## List Indexing:

Python Narayana

(107)

1. By using list indexing we can fetch specific element from the list, by using its index number.
2. It supports both forward and backward indexing.

Eg:-

```
>>> List1 = [10, 20, 30, 'python', True, 1.5, 2+3j].
```

[ 0      1      2      3      4      5      6  
  10, 20, 30, 'python', True, 1.5, 2+3j ]  
-6 -5 -4 -3 -2 -1

```
>>> print (List1[0]) 10
```

```
>>> print (List1[1]) 20
```

```
>>> print (List1[2]) 30
```

```
>>> print (List1[3]) python.
```

```
>>> print (List1[4]) True.
```

```
>>> print (List1[5]) 1.5
```

```
>>> print (List1[6]) (2+3j).
```

```
>>> print (List1[-1]) (2+3j)
```

```
>>> print (List1[-2]) 1.5
```

```
>>> print (List1[-3]) True.
```

```
>>> print (List1[-4]) python.
```

```
>>> print (List1[-5]) 30
```

```
>>> print (List1[-6]) 20
```

```
>>> print (List1[-7]) 10.
```

## Python Narayana

### Python 3

(108)

```
>>> lst = range(6)
range(0, 6)
>>> lst = range(2, 6)
>>> lst
range(2, 6)
>>> lst = range(5, 15, 2)
range(5, 15, 2)
>>> lst = range(3, 6)
>>> lst
range(3, 6)
```

### Python 2

Eg 1  

```
>>> lst = range(6)
[0, 1, 2, 3, 4, 5].
Eg 2

>>> lst = range(2, 6)
[2, 3, 4, 5].
Eg 3

>>> lst = range(5, 15, 2)
[5, 7, 9, 11, 13].
Eg 4

>>> lst = range(3, 6)
[3, 4, 5].
```

### List Slicing:

1. By using slicing we can fetch set of elements from list.
2. It also supports both forward and backward indexing
3. colon (:) is the slicing operator.

Eg 5  

```
>>> list1 = [10, 20, 30, 'python', True, 1.5, 2+3j].
```

$\begin{matrix} 0 & 1 & 2 & 3 \\ 10 & 20 & 30 & \text{'python'} \end{matrix}$	$\begin{matrix} 4 & 5 & 6 \\ \text{True} & 1.5 & 2+3j \end{matrix}$
$\begin{matrix} -7 & -6 & -5 & -4 \end{matrix}$	$\begin{matrix} -3 & -2 & -1 \end{matrix}$

```
>>> print(list1[0:2])
[10, 20]
>>> print(list1[2:5])
[30, 'python', True]
>>> print(list1[2:])
[30, 'python', True, 1.5, (2+3j)]
```

[10, 20]

[30, 'python', True]

[30, 'python', True, 1.5, (2+3j)]

python Narayana.

```

>>> print(list1[2:-1]) [30, 'python', True, 1.5].
>>> print(list1[-4:5]) ['python', True]. (109)
>>> print(list1[-4:]) ['python', True, 1.5, (2+3j)].
>>> print(list1[-5:]) [30, 'python', True, 1.5, (2+3j)]
>>> print(list1[-5:-2]) [30, 'python', True].

```

list is a mutable object that means we can alter (or) replace the existing list object

```

>>> list1 = [10, 20, 30, 'python', True, 1.5, (2+3j)].
>>> print(list1) [10, 20, 30, 'python', True, 1.5, (2+3j)]
>>> id(list1) 52330784
>>> list1[0] = 100 # modifying the content of list
>>> print(list1) [100, 20, 30, 'python', True, 1.5, (2+3j)]
>>> id(list1) 52330784.

```

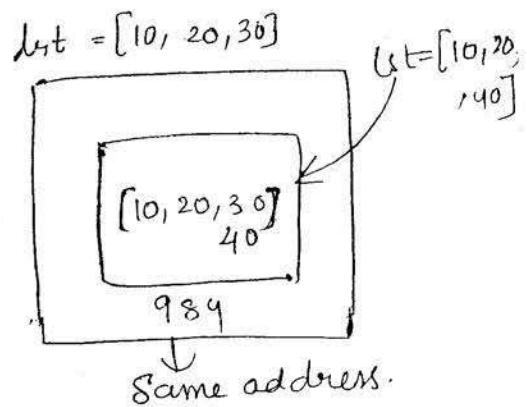
→ Here the address of list, list1 is not changed before and after modification that's why list is a mutable object.

→ list is a "mutable object" because we can't modify the existing list object

```

>>> for lst = [10, 20, 30]
>>> lst = [10, 20, 30]
>>> id(lst). 984

```



```
>>> lst.append(40)
```

Python Narayana.

```
>>> lst
```

[10, 20, 30, 40]

(110)

```
>>> id(lst)
```

984,

## List Concatenation:

→ Python supports concatenating two or more lists into single list.

Eg:- >>> list1 = [10, 'python', 5.5].

```
>>> list2 = [20, 30, 'Narayana', 3+4j].
```

```
>>> print(list1) [10, 'python', 5.5]
```

```
>>> print(list2) [20, 30, 'Narayana', (3+4j)]
```

```
>>> type(list1) <class 'list'>
```

```
>>> type(list2) <class 'list'>
```

```
>>> list3 = list1 + list2
```

```
>>> print(list3) [10, 'python', 5.5, 20, 30, 'Narayana']
```

```
>>> type(list3) <class 'list'> (3+4j)
```

Eg:- >>> l1 = [10, 20]

```
>>> l2 = [30, 40]
```

```
>>> l3 = [True, 2.5]
```

```
>>> l4 = l1 + l2 + l3.
```

```
>>> l4 [10, 20, 30, 40, True, 2.5]
```

```
>>> type(l4) <class 'list'>
```

## List Multiplication / List Repetition

Python supports multiplying the given list into N number of times.

Eg:-

```
>>> list1 = [10, 'python', 5.5].
```

```
>>> list5 = list1 * 8.
```

```
>>> print(list5). [10, 'python', 5.5, 10, 'python', 5.5].
```

## List functions :-

1. len() :- This function "counts no. of elements" in the list.

Eg:-

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]
```

```
>>> len(list1)
```

8

2. Count() :- This function counts the no. of occurrences of specific element in the list.

Eg:-

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]
```

```
>>> list1.count(10)
```

2

Eg:-

```
>>> lst = [1, 2, 3, True, False, 1, 0, False]
```

```
>>> lst.count(1)
```

python Narayana

## Python Narayana.

Eg:-

>>> lst = [1, 2, 3, True, False, 1, 0, False, 1+9j, 1, 1]. (112)

>>> lst.count(0) 3

>>> lst.count(False) 3.

3. Index() :- This function finds the index value for Specific element.

Eg:1

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]

>>> list1.index(10) 0

Eg:2

>>> list1 = [1, 2, 3, True, False, 1, 0, False, 1+9j, 1, 1]

>>> list1.index(1, 1) 3.

Eg:3

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]

>>> list1.index(10, 1) 6

Index on nested list

>>> lst [100, True, 1, 2, 3, 2, 5, 0, [8, 9], 10, 20, 4]

>>> lst[8].index(9) 1

>>> lst[8].index(8) 0

- 4) Append() :- This function adds new element at the end of the existing list

python Narayana

Eg:-1 >>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]  
>>> list1.append(50) (13)

>>> print(list1) [10, 20, 'python', 30, True, 'Narayana',  
10, 3+4j, 50].

Eg:-2 >>> list1 = [10, 20, 'python', 30, True, 'Narayana',  
10, 3+4j].

>>> list1.append(60)

>>> print(list1) [10, 20, 'python', 30, True, 'Narayana',  
10, (3+4j), 60].

→ We can also add multiple elements in the list by  
using append method but those multiple elements work  
like nested list (or) sub-list in the existing list.

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].

>>> list1.append([0, 1, 2])

>>> print(list1) [10, 20, 'python', 30, True, 'Narayana',  
10, (3+4j), [0, 1, 2]]

5. Extend () :- this function adds multiple elements at  
the end of the existing list.

Eg:-1

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]

>>> list1.extend([70, 80, 90])

>>> print(list1) [10, 20, 'python', 30, True, 'Narayana',  
10, (3+4j), 70, 80, 90].

## python Narayana

(11H)

Eg:2

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].
>>> list1.insert(3, 'Durga')
>>> print(list1) [10, 20, 'python', 'Durga', 30, True,
 'Narayana', 10, (3+4j)]
```

### 6. Insert()

→ we can also add multiple elements in the list at required place by using insert method but those multiple elements work like nested list (or) sub list in the existing list.

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].
>>> list1.insert(0, [0, 1, 2, 3])
>>> print(list1) [[0, 1, 2, 3], 10, 20, 'Python', 30,
 True, 'Narayana', 10, (3+4j)]
```

### 7. Remove():

- This function removes specific elements in the existing list.
- this function allows one argument and that should be element name.

Eg:1

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10,
 3+4j].
```

>>> list1.remove(10).

Python Narayana.

>>> print(list1)

[20, 'python', 30, True, 'Narayana',  
10, (3+4j)].

(115)

Eg:2

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].

>>> list1.remove(True)

>>> print(list1) [10, 20, 'python', 30, 'Narayana', 10  
(3+4j)].

8. pop() :-

a. This function removes specific element based on its index position.

b. The function also allows only one argument and that should be index number of an element.

Eg:1

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].

>>> list1.pop(1)

20

>>> print(list1)

[10, 'python', 30, True, 'Narayana',  
10, (3+4j)]

Eg:2

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].

>>> list1.pop(2)

'python'

>>> print(list1)

[10, 20, 30, True, 'Narayana', 10, (3+4j)].

9) Reverse():— This function reverses the existing list.

116

Eg:-

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].
```

```
>>> list1.reverse()
```

```
>>> print(list1) [(3+4j), 10, 'Narayana', True, 30, 'python', 20, 10].
```

10) Copy():— This function copies the existing list into new variable.

Eg:-

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].
```

```
>>> x = list1.copy()
```

```
>>> print(list1) [10, 20, 'python', 30, True, 'Narayana', 10, 0]
```

```
>>> print(x) [10, 20, 'python', 30, True, 'Narayana', (3+4j)].
```

10, (3+4j)].

11. clear():— this function clears & removes all the elements of the entire list

Eg:-

```
>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j].
```

```
>>> list1.clear()
```

```
>>> print(list1) [].
```

12. Max():— this function finds the maximum value in the given list.

Eg    >>> list2 = [10, 20, 30, 40].      Python Narayana  
      >>> max(list2)                          40. (117)

13. Min(): This function finds the minimum value in the given list.

Eg:-    >>> list2 = [10, 20, 30, 40]  
                >>> min(list2)                              10.

14. Sort (): this function sorts the elements.

```
>>> l8t = [1, 9, 5, 11, 2].
```

```
>>> lst. sort()
```

```
>>> print(lst) [1, 2, 5, 9, 11].
```

```
>>> l1 = [1, 2, 5, 8, 7, 4, 2, True].
```

```
>>> L1.Sort()
```

```
>>> print(l1) [1, True, 2, 2, 3, 4, 5, 7].
```

Note: by default this function sorts in ascending order, we can also get in descending order by setting True for `reverse`.

Eg:-

```
>>> lst = [1, 9, 5, 11, 2].
```

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

```
>>> print(list)
```

[11, 9, 5, 2, 1].

(118)

Eg:2

```
>>> l1 = [1, 2, 5, 3, 7, 4, 2, True].
>>> l1. sort(reverse = True)
>>> print(l1) [7, 5, 4, 3, 2, 2, 1, True].
```

del command:

This command is used to remove any specific element in the list or remove entire list object permanently.

→ It will give "NameError" after deleting the entire object.

NameError : List is not defined.

Removing Specific element by using del Command:

Eg:1 >>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]

```
>>> del list1[0].
>>> print(list1) [20, 'python', 30, True, 'Narayana', 10,
(3+4j)].
```

Eg:2

>>> list1 = [10, 20, 'python', 30, True, 'Narayana', 10, 3+4j]

```
>>> del list1[4]
>>> print(list1) [10, 20, 'python', 30, 'Narayana', 10,
(3+4j)].
```

## Removing entire list object by using del command

(119)

Eg:-

```
>>> list1 = [10, 20, 'python', 30, true, 'Narayana', 10, 3+5j].
```

```
>>> del list1
```

```
>>> print(list1)
```

# deleting list

# after deleting.

NameError: Name 'list1' is not found (it's defined).

### Difference between remove() and pop():

remove()	pop()
→ It removes the specific element from the existing list.	→ It also removes the specific element as remove() function.
→ It can't remove multiple elements.	→ It can't remove multiple. → It will return the removed value.

### Difference between clear() and del command

clear()	del command.
→ clear is a function used to clear (it's) remove the elements from the list (it's) object but not complete.	→ del is a command is used to remove any specific (it's) entire object.
→ It will return empty list after clear.	→ It will entirely remove (it's) deleted the list from database.

Eg:- `>>> lst = [10, 20, 30, 40]`

`>>> lst.clear()`

`>>> lst` [ ]

→ It will not support  
string and tuple.

Python Narayana  
→ `>>> lst = [10, 20, 30, 40]`

`del lst` (120)

→ It will throw NameError

→ It will not support for  
string, tuple.

Q. How to display the existing list in ascending  
order?

`>>> lst = [10, 20, 11, 12, 15].`

`>>> lst.sort()`

`>>> lst` [10, 11, 12, 15, 20].

\* To display ascending order we can use `sort()` function.

Q. How to display the existing list in descending  
order?

`>>> lst = [10, 20, 11, 15, 12].`

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

`>>> lst` [20, 15, 12, 11, 10].

Q. How to display the existing list in reverse  
order?

`>>> lst = [10, 15, 11, 20].`

`>>> lst.reverse()`

`>>> lst` [20, 11, 15, 10].

Q. How to take the clone/copy of existing list? (121)

By using copy() function or Assignment operator to take the clone existing list into new list.

```
>>> lst = [10, 20, 30].
>>> lst = lst.copy()
>>> lst1 [10, 20, 30].
>>> lst [10, 20, 30].
>>> lst = [10, 20, 30].
>>> lst2 = lst.
>>> lst = [10, 20, 30].
>>> lst2 = [10, 20, 30].
```

Q. How to reverse a list by using slicing operator?

```
>>> lst = [10, 20, 30, 40, 50].
>>> lst[::-1]. [50, 40, 30, 20, 10]
```

Q. How to get [20,50] from lst = [10, 20, 30, 40, 50]?

```
>>> lst = [10, 20, 30, 40, 50]
 0 1 2 3 4
>>> lst[1:3] [20, 50],
```

Hint: [20→1 index 1+3=4 ∵ 50]

python Narayana.

## Nested List :-

python Narayana

(22)

→ python supports Nested lists also, it means a list contains another lists.

```
>>> list1 = [10, 'Python', 5.5].
>>> list2 = [20, 30, 'Narayana', 3+4j].
>>> list3 = [1, True, 2, 'Durga'].

>>> print(list1) [10, 'Python', 5.5].
>>> print(list2) [20, 30, 'Narayana', (3+4j)].
>>> print(list3) [1, True, 2, 'Durga'].

>>> Nestlist = [list1, list2, list3] # creating a list by
using existing list

>>> print(Nestlist). [[10, 'Python', 5.5], [20, 30, 'Narayana',
(3+4j)], [1, True, 2, 'Durga']].

>>> type(Nestlist) <class 'List'>.
>>> print(Nestlist[0]) [10, 'Python', 5.5].
>>> print(Nestlist[1]) [20, 30, 'Narayana', (3+4j)]
>>> print(Nestlist[2]) [1, True, 2, 'Durga'].

>>> print(Nestlist[0][0]) 10
>>> print(Nestlist[0][1]) 'Python'
>>> print(Nestlist[0][2]) 5.5
>>> print(Nestlist[1][0]) 20
>>> print(Nestlist[1][3]) (3+4j)
>>> print(Nestlist[2][0]) 1
```

```
>>> print(Nestlist[2][1])
```

python Narayana.

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

Tree.

```
>>> print(Nestlist[2][3])
```

2

(123)

'Durga'.

### Working with index method on nested lists

```
>>> list1 = [10, 'python', 5.5]
```

```
>>> list2 = [20, 30, 'python' 'Narayana', 3+4j].
```

```
>>> list3 = [1, True, 'Durga'].
```

```
>>> Nestlist = [list1, list2, list3]
```

```
>>> print(Nestlist) [[10, 'python', 5.5], [20, 30, 'Narayana',
(3+4j)], [1, True, 2, 'Durga']].
```

0

1

2

[10, 'python', 5.5], [20, 30, 'Narayana', (3+4j)], [1, True, 2, 'Durga']  
0 1 2 0 1 2 3

```
>>> Nestlist.index([10, 'python', 5.5]) 0
```

```
>>> Nestlist[0].index(10) 0
```

```
>>> Nestlist[0].index('Python') 1
```

```
>>> Nestlist[0].index(5.5) 2
```

```
>>> Nestlist.index([20, 30, 'Narayana', (3+4j)]) 1
```

```
>>> Nestlist[1].index(20) 0
```

```
>>> Nestlist[1].index(30) 1
```

```
>>> Nestlist[1].index('Narayana') 2
```

```
>>> Nestlist[1].index((3+4j)) 3
```

```
>>> Nestlist.index([1, True, 2, 'Durga']) 2
```

```
>>> Nestlist[2].index[1] 0
```

python Narayana

```
>>> Nestlist[2].index(true) 0
>>> Nestlist[2].index(True, 1) 1 (124)
>>> Nestlist[2].index(2) 2
>>> Nestlist.index('Durga') 3.
```

## Conversions:-

### Converting a String into List

```
>>> str1 = 'python is very simple and easy language.'
>>> print(str1) python is very simple and
 easy language.
>>> type(str1) <class 'str'>

>>> list1 = str1.split()
>>> print(list1) ['python', 'is', 'very', 'simple', 'and',
 'easy', 'language'].
>>> type(list1) <class 'list'>.
```

### Converting a list to string:-

```
>>> list1 = ['python', 'is', 'very', 'simple', 'and', 'easy',
 'language'].
>>> print(list1) ['python', 'is', 'very', 'simple', 'and', 'easy',
 'language'].
>>> type(list1) <class 'list'>.
```

```
>>> str2 = " ".join(list1)
```

```
>>> print(str2) python is very simple and
 easy language.
```

```
>>> type(str2) <class 'str'>.
```

→ the main difference between string and list is mutation.

- 1. String is immutable whereas list is mutable.
- 2. Mutable Objects can be altered whereas immutable objects can't be altered.

Eg: String :-

```
>>> str1 = 'python'
```

```
>>> print(str1) python
```

```
>>> id(str1) 43143520.
```

```
>>> str1[0] = 's' # trying to replace 'p' with
 's' in 'python' string.
```

Error: TypeError: 'str' object doesn't support item assignment

We can add new letter to the existing string str1 but now it will create a new string str1 variable in the memory.

```
>>> str1 = str1 + 's'.
```

Python Narayana

```
>>> print(str1)
```

pythons

⑩

```
>>> id(str1)
```

53595200.

Eg: List:

```
>>> lst1 = [10, 20, 30, 40, 'Guido', 50].
```

```
>>> print(lst1)
```

[10, 20, 30, 40, 'Guido', 50]

```
>>> id(lst1)
```

53663224.

```
>>> lst1[4] = 'Rossum'
```

# trying to replace 'Guido'  
with 'Rossum' in lst1

```
>>> print(lst1)
```

[10, 20, 30, 40, 'Rossum', 50]

```
>>> id(lst1)
```

53663224.

List packing:

A list can be created by using a group of  
variable, it is called list packing.

```
>>> a = 10
```

```
>>> b = 20
```

```
>>> c = True
```

```
>>> d = 'py'
```

```
>>> list1 = [a, b, c, d]
```

```
>>> print(list1)
```

[10, 20, True, 'py'].

```
>>> type(list1)
```

<class 'list'>

## List Unpacking:

python Narayana

(127)

1. List unpacking allows to extract all list elements automatically into different variables.
2. The number of variables must be equal to number of elements in the List.

```
>>> lst = [10, 20, 'Python', True].
```

```
>>> print(lst)
```

[10, 20, 'Python', True]

```
>>> type(lst)
```

<class 'list'>

```
>>> a, b, c, d = lst.
```

# list unpacking

```
>>> print(a)
```

10

```
>>> type(a)
```

<class 'int'>

```
>>> print(b)
```

20

```
>>> type(b)
```

<class 'int'>

```
>>> print(c)
```

'Python'

```
>>> type(c)
```

<class 'str'>

```
>>> print(d)
```

True.

```
>>> type(d)
```

<class 'bool'>

Q. How to generate a list as per user requirement? Python Narayana

```
start_val = int(input('Enter start value:')) (128)
end_val = int(input('Enter end value:'))
step_size = int(input('Enter increment value:'))
userlist = range(start_val, end_val, step_size)
print(userlist).
```

Output in Python 2

```
Enter start value: 10
Enter end value: 21
Enter increment value: 2
[10, 12, 14, 16, 18, 20].
```

Output in Python 3.

```
Enter start value: 10
Enter end value: 21
Enter increment value: 2
range(10, 21, 2).
```

Adding different lists:

We can add elements of different lists by using lambda and map functions.

```
s = [1, 20, 3]
s1 = [1, 2, 3]
print(list(map(lambda x, y: x+y, s, s1)))
```

```
s = [1, 20, 3]
s1 = [1, 2, 3, 8]
print([x+y for x+y in zip(s, s1)]).
```

`s = [1, 20, 3, 6, 8]`

python Narayana

(129)

`s1 = [1, 2, 3, 8].`

`print ([x+y for x,y in zip(s, s1)]).`

Output :-

`[2, 22, 6]`

`[2, 22, 6].`

`[2, 22, 6, 14].`

List Comprehension :-

It provides an easy way to create list objects from any iterable objects based on some conditions.

Syntax :- `list = [expression for item in list if Condition].`

Q. How to display squares for all elements in the given list?

`>>> lst = [1, 2, 5, 4, 3].`

`>>> lst1 = [x*x for x in lst].`

`>>> print (lst1)`

`[1, 4, 25, 16, 9]`

`>>> type (lst1)`

`<class 'list'>`

Q. How to display squares for all even elements in the given list?

`>>> lst = [1, 2, 5, 4, 3]`

>>> lst3 = [i for i in lst1 if i not in lst2] Python Narayana

>>> lst3

[4, 5]

(130)

>>> type(lst3)

<class 'list'>

Eg:-

name = ['narayana', 'sai', 'krishna', 'Veni'].

>>> list4 = [[n.upper(), n.capitalize(), n.title(), len(n)]  
for n in name]

>> print(list4)

[['NARAYANA', 'Narayana', 'Narayana', 8], ['SAI', 'Sai', 'Sai', 3], ['KRISHNA', 'Krishna', 'Krishna', 7], ['VENI', 'Veni', 'Veni', 4]]

By Using :::-

Syn : [ starting-number :: increment-value ]

the default increment value is 1

>>> lst = [10, 20, 30, 40, 50, 60, 70, 80, 90]

>>> lst[1::] [20, 30, 40, 50, 60, 70, 80, 90]

>>> lst[2::] [30, 40, 50, 60, 70, 80, 90]

>>> lst[8::] [90]

>>> lst[-7::] [30, 40, 50, 60, 70, 80, 90]

>>> lst[-9:-1]	[ 10, 20, 30, 40, 50, 60, 70, 80, 90] (131)
>>> lst[-9:-3]	[ 10, 40, 70]
>>> lst[-9:-2]	[ 10].
>>> lst[-9:-1]	[ 10].
>>> lst[5:-3]	[ 80, 30]
>>> lst[5:-4]	[ 60, 20]
>>> lst[5:-1]	[ 60, 50, 40, 30, 20, 10]
>>> lst[::-9]	[ 90]
>>> lst[::-8]	[ 90, 10]
>>> lst[::-5]	[ 90, 40]
>>> lst[::-4]	[ 90, 50, 10].

### \* List Comparison \*

When comparing lists, the elements will compared from both lists parallelly.

- If the first elements from both lists are same then interpreter will compare the second elements from both lists,
- If second elements from both lists are also same then interpreter will compare the third elements from both lists.

Syn: cmp(list1, list2)

- If list1 is bigger than list2 then interpreter will 132 return 1.
- If list1 is smaller than list2 then interpreter will return -1.
- If list1 and list2 are same then interpreter will return 0.

Eg:1

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [1, 3, 5, 6]
>>> print cmp(lst1, lst2)
```

Explanation:- here the first element of lst1 is '1' and first element of lst2 is '1' so both are same. Now interpreter will compare Second elements, like 2 in the ~~list~~ lst1 and 3 in lst2,

2 from lst1 is smaller than 3 from lst2, So interpreter returned -1.

Eg:2

```
>>> lst1 = [10, 20, 30]
>>> lst2 = [10, 11, 12, 13]
>>> print cmp(lst1, lst2).
```

Explanation:- here first elements both list are same.

20 from lst1 is bigger than 11, So interpreter returned 1

Eg:3

```
>>> lst1 = [10, 20, 30]
>>> lst2 = [10, 20, 30]
>>> print cmp(lst1, lst2)
```

Explanation:- here first elements from both lists are same.  
 So interpreter checked second elements from lists second  
 elements are also same from both lists, interpreter checked  
 third elements, but third elements are also same.  
 finally all elements are same for both lists so interpreter  
 returned 0.

Eg:4

```
>>> lst1 = [10, 20, 30, 40]
>>> lst2 = [10, 11, 12]
>>> print cmp(lst1, lst2)
```

Explanation first elements are same, 20 from lst1 is bigger  
 than 11 from lst2. So interpreter returned 1.

Eg:5

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [10, 20]
>>> print cmp(lst1, lst2)
```

Explanation 1 from lst1 is smaller than 10 from lst2, so interpreter returned -1.

(13H)

Eg:6

```
>>> lst1 = [10]
>>> lst2 = [1, 2, 3, 14]
>>> print cmp (lst1, lst2)
```

Explanation 10 from lst1 is bigger than 1 from lst2, so interpreter returned 1.

Eg:7

```
>>> lst1 = [1, 2]
>>> lst2 = [1, 2, 3]
>>> print cmp (lst1, lst2).
```

Explanation first two elements (1, 2) are same from both lists, lst1 has no third element but lst2 has 3rd element, that means lst1 is smaller than lst2. So interpreter returned -1.

Eg:8

```
>>> lst1 = ['a', 1, 2]
>>> lst2 = [10, 20]
>>> print cmp (lst1, lst2)
```

# \* Tuple - Data Structure \* Python Narayana

(135)

1. Tuple is used to represent a set of homogenous (or) heterogenous elements into a single entity.
2. Tuple object are immutable that means once if we create a tuple later we can't modify that tuple object.
3. All elements are separated by Commas (,) and enclosed by parentheses. ( ) parentheses are optional.
4. Tuple allows duplicate elements.
5. Every element in the tuple has its own index number.
6. Tuple supports both forward indexing and also backward indexing, forward indexing.
7. If we take only one element in the tuple then we should use comma(,) after that single element.
8. Tuples can create a tuple in different ways, like with tuple(), with () (or) without () also.
9. Tuples can be used as keys to the dictionary.
10. The main difference between lists and tuples is List are enclosed in brackets ([ ]) and their elements

and size can be changed, while tuples are Python Narayana  
enclosed in parentheses ( ) and cannot be updated.

### Creating a tuple with tuple():

(36)

Eg:-

```
>>> tup = tuple([10, 20, 30, True, 'python'])
>>> print(tup) (10, 20, 30, True, 'python')
>>> type(tup) <class 'tuple'>
>>> id(tup) 52059760.
```

### Creating an empty tuple:-

Eg:-

```
>>> tup = ()
>>> print(tup) # creating empty tuple.
>>> type(tup) ()
>>> id(tup) <class 'tuple'> 2313456
```

### Creating a tuple with ():

Eg:-

```
>>> tup2 = (10, 20, 30, 40, 50) # creating homogenous
>>> print(tup2) tuple
>>> type(tup2) (10, 20, 30, 40, 50)
>>> id(tup2) <class 'tuple'> 63484864.
```

### Creating a tuple without ():

Eg:-

```
>>> tup = 10, 20, True, 'py' # Creating tuple without
parathensis.
```

```
>>> print(tup)
Python Narayana
(10, 20, True, 'py')
>>> type(tup)
<class 'tuple'>
>>> id(tup)
67086688. (137)
```

### Creating a tuple with heterogeneous elements:

Eg:-

```
>>> tup1 = (10, 20, 30, True, "python", 10.5, 3+5j)
creating heterogeneous
tuple.
(10, 20, 30, True, 'python', 10.5,
<class 'tuple'> (3+5j))
>>> print(tup1)
58963648.
```

### Creating a tuple with single elements

→ creating a tuple with a single element is tricky, if we take only one element then the type of that object will be based on specified element.

```
>>> t2 = (1)
>>> t2
1
>>> type(t2) < type 'int'>
>>> t2 = (True)
>>> print(t2)
True.
>>> type(t2) < type 'bool'>
```

To solve the above problem we should use  
Comma(,) after the elements in the tuple, like,

(138)

```
Eg>>> t2 = (1)
>>> print(t2) (1)
>>> type(t2) <type 'tuple'>
Eg>>> t2 = (False)
>>> print(t2) False
>>> type(t2) <type 'bool'>
```

### Tuple Indexing

Tuple indexing is nothing but fetching a specific element from the existing tuple by using its index value.

Eg:-

```
>>> tup = (10, 20, 30, True, "python", 10.5, 3+5j, 10)
>>> print(tup) (10, 20, 30, True, 'python', 10.5, (3+5j), 10)
>>> type(p) <class 'tuple'>
>>> id(tup) 63560624.
```

0	1	2	3	4	5	6	7
-8	-7	-6	-5	-4	-3	-2	-1

(10, 20, 30, True, "python", 10.5, 3+5j, 10)

True.

'python'.

10.5

(3+5j)

10

10

(3+5j)

10.5

'python'.

True.

30.

20.

10.

```
>>> tup[0]
>>> tup[1]
>>> tup[2]
>>> tup[3]
>>> tup[4]
>>> tup[5].
>>> tup[6]
>>> tup[7]
>>> tup[-1]
>>> tup[-2]
>>> tup[-3]
>>> tup[-4]
>>> tup[-5]
>>> tup[-6]
>>> tup[-7]
>>> tup[-8]
```

## Tuple Slicing:-

Tuple slicing is nothing but fetching a sequence of elements from the existing tuple by using their index values.

```
>>> tup = (10, 20, 30 ,True, "python", 10.5, 3+5j, 10)
```

```
>>> print (tup)
10, 20, 30, True, 'python', 10.5
(3+5j), 10)
```

```
>>> type(tup)
```

Python Narayana  
<class 'tuple'>

63560496.

(1310)

0 1 2 3 4 5 6 7

(10, 20, 30, True, "python", 10.5, 3+5j, 10)

-8 -7 -6 -5 -4 -3 -2 -1

```
>>> tup[0:4]
```

(10, 20, 30, True).

```
>>> tup[0:0]
```

()

```
>>> tup[0:1]
```

(10,)

```
>>> tup[0:5]
```

(10, 20, 30, True, 'python').

```
>>> tup[3:5]
```

(True, 'python').

```
>>> tup[2:-2]
```

(30, True, 'python', 10.5)

```
>>> tup[-5:-2]
```

(True, 'python', 10.5 (3+5j), 10)

```
>>> tup[6:]
```

((3+5j), 10).

Eg: tuple1 = ('Narayana', 1037, 1000, 'python', True)

tinytuple = ("Super", 'Django', True)

print(tuple1) # prints complete tuple.

print(tuple1[0]) # prints first element of the tuple.

print(tuple1[1:3]) # prints elements starting from 2nd and till 3rd.

```

print(tuple1[2:]) # prints elements starting
 from 3rd element (111)
print(tinytuple * 2) # prints tuple two times.
print(tuple1 + tinytuple) # prints concatenated
 tuple.

```

### Output's

('Narayana', 1037, 1000, 'python', True)

& "Narayana"

(1037, 1000)

(1000, 'python', True)

('Super', 'Django', True, 'Super', 'Django', True).

('Narayana', 1037, 1000, 'python', True, 'Super', 'Django', True).

### Tuple Concatenation

We can concatenate two or more tuples in python

Ex:-

```
>>> tup1 = (1, 2, 3, 'a', True) # creating first tuple
 tup1
```

```
>>> print(tup1) (1, 2, 3, 'a', True)
```

```
>>> type(tup) < class 'tuple'>
```

```
>>> tup2 = (10, 20, False, 'b') # creating second
 tuple tup2
```

```
>>> print(tup2) (10, 20, False, 'b')
```

```
>>> type(tup2)
```

Python Narayana  
<class 'tuple'>

(142)

```
>>> tup3 = tup1 + tup2 # concatenating tup1 and tup2
```

```
>>> print(tup3) (1, 2, 3, 'a', True, 10, 20, False, 'b')
```

```
>>> type(tup3) <class 'tuple'>
```

## Tuple multiplication (or) repetition:-

we can multiply (or) repeat a tuple n number of times.

```
>>> tup1 = (1, 2, 3, 'a', True)
```

```
>>> print(tup1) (1, 2, 3, 'a', True)
```

```
>>> type(tup1) <class 'tuple'>
```

```
>>> tup1 * 3 (1, 2, 3, 'a', True, 1, 2, 3, 'a', True, 1, 2, 3, 'a', True)
```

## Tuple functions:-

1. All(): this function returns True if all elements are true, if any elements is false (either 0 (or) False) then it will return False. For empty tuple also it will return True.

Eg:-

```
>>> tup = (1, 2, 3)
```

```
>>> print(all(tup)) True.
```

```
>>> tup = (1, 2, 3, 0)
```

```
>>> print(all(tup)) False
```

>>> tup = (2,3)

python Narayana

>>> print(all(tup))

True.

(143)

>>> tup = ()

True.

>>> print(all(tup))

.

>>> tup = (True,)

True.

>>> print(all(tup))

2. Any() :- This function returns true if any one element is true in the tuple.

Eg :-

>>> tup = (1,2,3)

True.

>>> print(any(tup))

>>> tup = (1,2,3,0)

True.

>>> print(any(tup))

>>> tup = (False,2,3)

True.

>>> print(any(tup))

>>> tup = (False,0,0,False)

False.

>>> print(any(tup))

>>> tup = ()

False.

>>> print(any(tup))

3. `len()`:— this function returns no. of elements in the tuple.

(144)

`>>> tup = (1, 2, 3, 4, 'a', 5, 5)`

`>>> len(tup)`

6

4. `Count()`:— this function counts the number of occurrences of a specific elements.

Eg: `>>> tup = (1, 10, 20, True, 0)`

`>>> tup.count(1)` 2

`>>> tup.count(0)` 1.

5. `Index()`:— this function is used to find the index value of specific element.

Eg: `>>> tup = (1, 10, 20, True, 0)`.

`>>> tup.index(0)` 4

`>>> tup.index(10)` 1

`>>> tup.index(20)` 2.

6. `Max()`:— this function returns maximum value from the tuple elements.

Eg:

`>>> tup = (1, 32, 55, 3, 5, 23)`

`>>> max(tup)` 55.

7) `Min()`: This function returns minimum value from the tuple elements.

(145)

Eg:- `>>> tup = (1, 3, 2, 55, 3, 5, 23)`

`>>> min(tup).` 1.

8. `Sorted()`: This function sort the data.

Eg:-

`>>> tup = (1, 3, 2, 55, 3, 5, 23)`

`>>> sorted(tup)` [1, 2, 3, 3, 5, 23, 55].

Note:- by default this function sorts the data in ascending order. we can also get in descending order by setting True for Reverse.

Eg:- `>>> tup = (1, 3, 2, 55, 3, 5, 23)`

`>>> sorted(tup, reverse = True)` [55, 23, 5, 3, 3, 2, 1].  
(Q)

`>>> t1 = tuple([1, 2, 3, 7, 4])`

`>>> t1` (1, 2, 3, 7, 4)

`>>> t2 = reversed(t1)`

`>>> tuple(t2)` (4, 7, 3, 2, 1).

9) `Sum()`: This function returns sum of all the elements.

Eg:- `>>> lst = [1, 9, 5, 11, 2]`

`>>> sum(lst)`

Del Command:

We cannot delete the elements of existing tuple but we can delete the entire tuple object by using del command.

Eg:-

```
>>> tup = (10, 20, "python", 1.3)
```

```
>>> print(tup)
```

(10, 20, 'python', 1.3)

```
>>> type(tup)
```

<class 'tuple'>

```
>>> del tup
```

# deleting tuple by using del command

```
>>> print(tup)
```

# after deleting.

NameError: name 'tup' is not defined.

We can replace the elements of list but not tuple, like

```
>>> lst = [10, 20, 30, 'py', True].
```

```
>>> lst[4] = False
```

# it is possible in list

```
>>> print(lst)
```

[10, 20, 30, 'py', False].

```
>>> tup = (10, 20, 30, 'py', True)
```

```
>>> tup[4] = False.
```

# it is not possible in tuple.

Traceback (most recent call last):

File "<pyshell #15>", line 1, in <module>

tup[4] = False

`TypeError: 'tuple' object doesn't support item assignment`

(147)

Q. How to display the given tuple in ascending order?

```
>>> tup = (10, 20, 5, 3, 30)
>>> sorted(tup) [3, 10, 20, 30].
>>> print(tup)
```

Q. How to reverse the given tuple?

```
>>> tup = (10, 20, 5, 3, 30)
>>> reversed(tup) (30, 3, 5, 20, 10).
>>> print(tuple(tup))
```

### Nested tuples:-

1. Python supports nested tuple, i.e., tuple in another tuple.
2. Tuple allows list as its element.

Eg:-

```
>>> t1 = (1, 'a', True)
>>> print(t1) (1, 'a', True)
>>> type(t1) <class 'tuple'>.
```

```
>>> t2 = (10, 'b', False)
>>> print(t2) (10, 'b', False)
>>> type(t2) <class 'tuple'>
```

>>> t3 = (t1, 100, 'python', t2) # creating a tuple  
with existing tuples t1  
and t2. (14)

>>> print(t3) ((1, 'a', True), 100, ('python', (10, 'b', False)))

>>> type(t3) <class 'tuple'>

>>> print(t3[0]) (1, 'a', True)

>>> print(t3[1]) 100.

>>> print(t3[2]) python

>>> print(t3[3]) (10, 'b', False).

>>> print(t3[3][0]) 10

>>> print(t3[3][1]) b.

>>> print(t3[3][2]) False.

>>> print(t3[0][0]) 1.

>>> print(t3[0][1]) a

>>> print(t3[0][2]) True.

>>> print(t3[0:2]) ((1, 'a', True), 100)

>>> t3[2:4] ('python', (10, 'b', False)).

>>> t3[-2:4] ('python', (10, 'b', False))

Note: We can't modify any element of the above tuples because tuples are immutable.

If the tuple contains a list as a element then we can modify the elements of the list as it a mutable object.

Eg: `>>> tup = (1, 2, [10, 12, 'd'], (100, 200, 300), 3, 'Narayana').`

(149)

`>>> print(tup)`  $(1, 2, [10, 12, 'd'], (100, 200, 300),$   
`3, 'Narayana').`

`>>> print(tup)`

" " "

`>>> type(tup)`

`< class 'tuple' >.`

`>>> tup[0]`

1

`>>> tup[1]`

2

`>>> tup[2]`

$[10, 12, 'd']$ .

`>>> tup[3]`

$(100, 200, 300)$

`>>> tup[4]`

3.

`>>> tup[5]`

'Narayana'.

`>>> tup[0] = 50` # trying to replace element 1 with  
50, interpreter throws error

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

`>>> tup[1] = 50` # trying to replace element 2 with  
50, interpreter throws error.

`>>> tup[1] = 50`. # trying to replace element 2  
with 2 with 50, interpreter throws error.

TypeError: 'tuple' object doesn't support item  
assignment.

```
>>> tup[2] = 50 # trying to replace element
[10, 12, 'd'] with 50, interpreter
throws error. (150)
```

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

```
>>> tup[2][0] = 50 # trying to replace element of list
[100] with 50; interpreter accepts.
```

```
>>> print(tup) (1, 2, [50, 12, 'd'], (100, 200, 300), 3,
'Narayana').
```

## Conversions :-

### Converting tuple to list :-

```
>>> tup = (1, 2, 4, 9, 8) # creating a tuple.
>>> print(tup) (1, 2, 4, 9, 8)
>>> type(tup) <class 'tuple'>.
>>> lst = list(tup) # converting tuple to
 list by using list().
>>> print(lst) [1, 2, 4, 9, 8].
>>> type(lst) <class 'list'>
```

### Converting list to tuples :-

```
>>> lst = [10, 20, 30, 40, 'a'] # creating a list
>>> print(lst) [10, 20, 30, 40, 'a']
>>> type(lst) <class 'list'>
```

```
>>> tup = tuple(lst)
```

Python Narayana  
# converting list to  
tuple by using tuple()

```
>>> print(tup)
```

(10, 20, 30, 40, 'a') 151

```
>>> type(tup)
```

<class 'tuple'>

### Converting tuple to string:

```
>>> tup = ('a', 'b', 'c') # creating tuple.
```

```
>>> print(tup) ('a', 'b', 'c')
```

```
>>> type(tup) <class 'tuple'>
```

```
>>> str1 = ''.join(tup) # converting tuple to
string by using join
method.
```

```
>>> print(str1) abc.
```

```
>>> type(str1) <class 'str'>
```

### Converting string to tuple:

```
>>> str1 = "python Narayana" # creating a string
```

```
>>> print(str1) python Narayana
```

```
>>> type(str1) <class 'str'>
```

```
>>> tup = tuple(str1) # converting a tuple
to string by using
tuple fun.
```

```
>>> print(tup) ('p', 'y', 't', 'h', 'o', 'n', ' ', 'N', 'a',
'r', 'a', 'y', ' ', 'n', 'a', 'r').
```

## Tuple packing :-

python Narayana.

We can create a tuple by using existing variables,  
So its called tuple packing.

(152)

```
>>> a=10
>>> b=20
>>> c='python'.
>>> d=2+5j.

>>> tup = (a,b,c,d)
>>> print(tup) (10, 20, 'python', (2+5j))
>>> type(tup) <class 'tuple'>
>>> id(tup) 621678303.
```

## Tuple Unpacking:-

1. Tuple unpacking allows to extract tuple elements automatically.
2. Tuple unpacking is the list of variables on the left has the same number of elements as the length of the tuple.

```
>>> tup = (1,2,3,4)
>>> a,b,c,d = tup # tuple unpacking.

>>> print(a) 1
>>> print(b) 2.
```

>>> print(c)

3 Python Narayan

>>> print(d)

4

(153)

## Advantages of Tuple Over list

- generally we use tuple for heterogeneous elements and list for homogeneous elements.
- Iterating through tuple is faster than with list because tuples are immutable, so there might be a slight performance boost.
- Tuples can be used as key for a dictionary with list, this is not possible because list is a mutable object.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write protected.

Python Narayan

# Difference between list & tuple

(15H)

1. If we need to add or remove the elements to object in the future then we choose list  
If we don't want add or remove the elements to the object in the future then we choose tuple.
2. List is represented by []  
Tuple is represented by ()
3. [] are compulsory for list  
() are optional for Tuple
4. We can delete specific element by using Del command in the list  
We can't delete the specific element by using Del in tuple.
5. We can clear all elements of a list by using clear()  
We can't clear all elements of a tuple by using clear()
6. When we create a tuple which with one element,  
then we should use comma ',' after the element  
Comma is not required in the list to create list with single element.

7. List is dynamic object  
Tuple is static object.
8. We can add or remove elements in the list by using append(), extend(), insert(), remove(), and pop()  
we can't add or remove elements in the tuple by using functions.
9. Range() is used to generate the list  
Range() is not used to generate the tuple
10. split() result stores in list format.  
Database data stores in tuple format when we fetched data from database to python application.

Python Narayana

By Using :: :-

python Narayana

(156)

>>> t = (10, 'Py', 30, 40, 10, 70, 'Narayana', 'python')

>>> t[1::] ('Py', 30, 40, 10, 70, 'Narayana', 'python')

>>> t[2::] (30, 40, 10, 70, 'Narayana', 'python')

>>> t[3::] (40, 10, 70, 'Narayana', 'python')

>>> t[7::] ('python',)

>>> t[8::] ()

>>> t[1::1] ('Py', 30, 40, 10, 70, 'Narayana', 'python')

>>> t[1::4] ('Py', 70).

>>> t[1::8] ('Py',)

>>> t[1::7] ('Py',)

>>> t[2::5] (30, 'python')

>>> t[2::6] (30,)

>>> t[2::7] (30,)

>>> t[3::3] (40, 'Narayana')

>>> t[3::4] (40, 'python')

>>> t[3::6] (40,)

>>> t[7::1] ('python',)

>>> t[8::1] ()

>>> t[2::-1] (30, 'Py', 10)

>>> t[-1::-5] ('python',)

## Tuple Comparison

python Notes

(157)

Eg1 In Python 2>

>>> tup1 = (10, 20, 30)

>>> tup2 = (11, 12)

>>> print cmp(tup1, tup2)

-1

Eg2 >>> tup1 = (10, 20, 30)

>>> tup2 = (1, 2, 3, 4, 5)

1

>>> print cmp(tup1, tup2)

Eg3 >>> tup1 = (10, 20, 30)

>>> tup2 = (10, 20, 30, 4, 5)

-1

>>> print cmp(tup1, tup2)

Eg4 >>> tup1 = (10, 20, 30)

>>> tup2 = (10, 20, 30)

0

>>> print cmp(tup1, tup2)

Eg5

>>> tup1 = (10, 20, 30)

>>> tup2 = (10, 20, 30)

1

>>> print cmp(tup1, tup2)

Eg6 >>> p1 tup1 = ('a', 'b', 'x')

>>> tup2 = ('a', 'b', 'y')

-1

>>> print cmp(tup1, tup2)

E97

```
>>> tup1 = (True, True)
>>> tup2 = (False, True)
>>> print cmp(tup1, tup2)
```

python Narayana

(158)

1

Python Narayana

## \* Set - Data Structure \*

Python Narayana

(159)

1. A Set is unordered collection of unique elements.
2. Set is commonly used in membership testing, removing duplicates, from a sequence and performing mathematical operations such as intersection, union difference and symmetric difference.
3. Set will not allow duplicate values.
4. Insertion order is ~~order~~ not preserved but elements can be stored.
5. The major advantages of using a Set, as opposed to a List, is that it has a highly optimized method for checking whether a specific element is contained in the set.
6. Sets do not support indexing, slicing.
7. Sets do not support concatenation and multiplication.
8. There are currently two built-in set types.
  - a. set
  - b. frozenset

Set: The set type is mutable - the contents can be changed using methods like add() and remove()

Python Narayana  
160

→ Since is mutable, it has no hash value and cannot be either a dictionary key (or) as an element of another set.

### Frozen Sets

→ The frozenset type is immutable. Its contents cannot be altered after it is created, it can be used as a dictionary key or as an element of element set.

We can create a set in different ways,

1. Creating an empty set using set() and add elements to that empty set

Eg:-

```
>>> Set1 = set()
```

# creating an empty set with set()

```
>>> Set1.add(10)
```

# adding elements to empty set.

```
>>> Set1.add(20)
```

# adding elements to empty set.

```
>>> Set1.add(10)
```

# adding duplicate value to set.

```
>>> print(Set1)
```

{ 10, 20, 30 }

## 2. Creating a Set with elements using set():

(161)

Eg:-  
    >>> Se2 = set([1, 2, 4, 'a', 2+4j, True]) # creating  
        >>> print(Se2)  
        >>> type(Se2)

{1, 2, 4, (2+4j), 'a'}.  
<class 'Set'>.

## 3. Creating a Set with curly braces:-

Eg:-  
    >>> Se3 = {1, 2, 3, 4, "Narayana", True} # creating  
        >>> print(Se3)  
        >>> type(Se3)

{1, 2, 3, 4, 'Narayana'}.  
<class 'Set'>.

### Performing membership operations:-

- we use 'in' and 'not in' to perform membership operations.
- 'in' and 'not' are used to check the specific element is a part of the existing set (or) not.

```
>>> Se1 = {10, 20, 30, True, 100, 'Narayana', 'python'
>>> 'Narayana' in Se1 True.
>>> 'Django' in Se1 False.
>>> 'Oracle' in Se1 False.
>>> 1 not in Se1 False.
>>> 100 not in Se1 False.
>>> 20 in Se1 True.
```

# Removing duplicate elements from other Python Non-sequences

(162)

```
>>> lst = [10, 20, 10, 40, 50, 10, 20].
```

```
>>> lst = list(set(lst))
```

```
>>> print(lst)
```

[40, 10, 20, 50].

```
>>> type(lst)
```

<class 'list'>

## Set functions :-

1. Add () :- this function adds new elements to existing set.

Eg:-

```
>>> set1 = {1, 2, 3, 4, 5}.
```

```
>>> print(set1)
```

{1, 2, 3, 4, 5}.

```
>>> set1.add(6)
```

# adding elements.

```
>>> set1.add(7)
```

```
>>> print(set1)
```

{1, 2, 3, 4, 5, 6, 7}

Note:- we can not add new elements to the frozenset

```
>>> fs = frozenset([10, 20, 30, 40])
```

```
>>> print(fs)
```

{10, 20, 30, 40}

```
>>> fs.add(50) # trying to add new element
```

to frozenset.

Error: AttributeError: 'frozenset' object has no attribute  
'add'

Remove() :- It will remove elements from the set, if that element is not found then it will throw error

(163)

Eg:-

```
>>> Set = {1, 2, 3, 4, 5}.\n>>> print(Set)\n{1, 2, 3, 4, 5}.\n>>> type(Set)\n<class 'set'>.\n>>> Set.remove(5)\n# removing element from set\n>>> Set.remove(4)\n# removing element from set.\n>>> Set.remove(15)\n# trying to remove element\nwhich is not there in set
```

Error : KeyError: 15.

Discard() :- It will remove elements from the set, if that element is not fund in the set then it will do nothing.

Eg:-

```
>>> Set = {1, 2, 3, 4, 5}.\n>>> print(Set)\n{1, 2, 3, 4, 5}.\n>>> Set.discard(10)\n# trying to remove element\nwhich not there in the set.\n>>> Set.discard(20)\n# trying to remove element\nwhich not there in the set.\n>>> Set.discard(5)\n# removing element which is\nthere in the set
```

>>> print(s1)

python Narayana  
S1, 2, 3, 4, 5.

(16H)

the difference b/w remove() and discard() is,  
remove() :- if we take the element which is not there  
in the set then it will throw error.

discard() :- if we take the element which is not  
there in the set then it will do nothing, means it  
will not throw error.

Copy() :- this function copies the elements of one set  
to another new set.

Eg:-

>>> s1 = {1, 2, 3, 4, 5}.

#copying s1 elements to s2

>>> s2 = s1.copy()

{1, 2, 3, 4, 5}

>>> s1.

{1, 2, 3, 4, 5}

>>> s2.

Clear() :- this function clears the existing function

>>> s1 = {1, 2, 3, 4, 5}.

{1, 2, 3, 4, 5}

>>> print(s1)

< class 'set'>

>>> s1.clear()

#clearing the s1, so s1 will become  
empty set

>>> print(s1)

set().

isdisjoint() :- This function returns python Narayana True if both are empty sets or if both sets contains nonmatching elements.

(165)

Eg:-

>>> Se1 = Set()

>>> Se2 = Set()

>>> Se1. isdisjoint(Se2)

True.

>>> Se1 = set()

>>> Se2 = {1, 2, 3}.

True.

>>> Se1. isdisjoint(Se2)

>>> Se1 = {1, 2, 3}

>>> Se2 = {1, 2, 3, 4}.

>>> Se1. isdisjoint(Se2)

False.

issubset() :-

x. issubset(y), returns True, if x is a subset of y

"<=" is an abbreviation for "subset of".

>>> Se1 = {1, 2, 3, 4, 5}.

>>> Se2 = {1, 2, 3}.

>>> Se2. issubset(Se1)

True.

>>> Se1. issubset(Se2)

False.

(or)

python Narayana

131

>>> set2 <= set1  
>>> set1 <= set2

True  
False.

166

## issuperset():

X.issuperset(y) returns True, if X is a superset of y. ">=" is an abbreviation for "issuperset of".

Eg:-

>>> set1 = {1, 2, 3, 4, 5}.

>>> set2 = {1, 2, 3}.

>>> set2.issuperset(set1)      False  
>>> set1.issuperset(set2)      True

>>> set2 >= set1      False  
>>> set1 >= set2      True

We can also check the elements whether they belong to set (or) not.

Eg:-

>>> set1 = {1, 2, 3, "python", 3+5j, 8}.

>>> 4 in set1      False

>>> 1 in set1      True

>>> "python" in set1      True

>>> 10 not in set1      True

>>> "Narayana" not in set      python Narayana  
                                        True. (16)

Union: it returns the union of two sets, that means it returns all the values from both sets except duplicate values.

→ the same result we can get by using `|` b/w two sets.

Syn: < first Set . union (< second - Set >) (Q1)  
< first Set > | < second - Set >

Eg:-

>>> Se1 = {1, 2, 3, 4, 5}

>>> Se2 = {1, 2, 3, 6, 7}.

>>> Se1.union(Se2)                    {1, 2, 3, 4, 5, 6, 7} Q1

>>> Se1 | Se2                            {1, 2, 3, 4, 5, 6, 7}.

(Q1).

>>> Se2.union(Se1)                    {1, 2, 3, 4, 5, 6, 7}

>>> Se2 | Se1                            {1, 2, 3, 4, 5, 6, 7}

Intersection: it returns an intersection elements of two sets, that means it returns only common elements from both sets.

→ that same operation we can get by sing '`&`' operator.

Syn: <First-Set>.intersection (<Second-Set>) (or)  
 <First-Set> & <Second-Set>.

(168)

Eg:-

&gt;&gt;&gt; Set1 = {1, 2, 3, 4, 5}.

&gt;&gt;&gt; Set2 = {1, 2, 3, 6, 7}.

>>> Set1.intersection (Set2)      {1, 2, 3} (or).  
 >>> Set1 & Set2                      {1, 2, 3}

>>> (or).

>>> Set2.intersection (Set1)      {1, 2, 3}  
 >>> Set2 & Set1                      {1, 2, 3}

Difference → It returns all elements from first set  
 which are not there in the second set

Syn: <first-Set>.difference (<Second-Set>) (or)  
 <first-Set> - <Second-Set>.

Eg:-

&gt;&gt;&gt; Set1 = {1, 2, 3, 4, 5}.

&gt;&gt;&gt; Set2 = {1, 2, 3, 6, 7}.

>>> Set1.difference (Set2)      {4, 5} (or)  
 >> Set1 - Set2                      {4, 5}.

(or)

>>> Set2.difference (Set1)      {6, 7}  
 >>> Set2 - Set1                      {6, 7}.

Python Narayana

Intersection - update: this function will update  
the first set with the result of intersection b/w  
first-set and second-set.

(169)

Syn: [ < First\_Set > . intersection\_update (< Second\_Set >) ]

Eg:- >>> set1 = {1, 2, 3, 4, 5}.

>>> set2 = {1, 2, 3, 6, 7}.

>>> set1.intersection\_update (set2)

>>> print (set1)

{1, 2, 3}

>>> print (set2)

{1, 2, 3, 6, 7}.

(Q1).

>>> set1 = {1, 2, 3, 4, 5}.

>>> set2 = {1, 2, 3, 6, 7}.

>>> set2.intersection\_update (set1)

>>> print (set1)

{1, 2, 3, 4, 5}

>>> print (set2)

{1, 2, 3}.

Difference update: The result of difference b/w two  
sets will in first-set.

Syn: < first\_Set > . difference\_update (< Second\_Set > )

Eg:- `>>> set1 = {1, 2, 3, 4, 5}.`  
`>>> set2 = {1, 2, 3, 6, 7}.`

`>>> set1.difference_update(set2)`

`>>> print(set1)`

{4, 5}

`>>> print(set2)`

{1, 2, 3, 6, 7}.

(or).

`>>> set1 = {1, 2, 3, 4, 5}.`

`>>> set2 = {1, 2, 3, 6, 7}.`

`>>> set2.difference_update(set1).`

`>>> print(set1)`

{1, 2, 3, 4, 5}

`>>> print(set2)`

{6, 7}.

Symmetric-difference :- It returns unmatched elements from both sets

Syn:- `<First-Set> symmetric_difference(<second_Set>)`

Eg:-

`>>> set1 = {1, 2, 3, 4, 5}.`

`>>> set2 = {1, 2, 3, 6, 7}.`

`>>> set1.symmetric_difference(set2)`

{4, 5, 6, 7}

## Symmetric-difference-update:

Python Programming

(171)

It will store the unmatched elements from both Sets into First-Set.

Syn: <First-Set>. Symmetric-difference-update (<Second-Set>)

Eg:-

>>> set1 = {1, 2, 3, 4, 5}.

>>> set2 = {1, 2, 3, 6, 7}.

>>> set1. symmetric\_difference\_update (set2)

>>> print (set1)

{4, 5, 6, 7}.

>>> print (set2)

{1, 2, 3, 6, 7}.

(Ex).

>>> set1 = {1, 2, 3, 4, 5}.

>>> set2 = {1, 2, 3, 6, 7}.

>>> set2. symmetric\_difference\_update (set1)

>>> print (set1)

{1, 2, 3, 4, 5}.

>>> print (set2)

{4, 5, 6, 7}.

Python Programming

## Set and frozen sets support the following operators:

(172)

key in s	# containment check
key not in s.	# non-containment check.
set1 == set2	# $s_1$ is equivalent to $s_2$ .
set1 != set2	# $s_1$ is <del>not</del> equivalent to $s_2$ .
set1 < set2	# $s_1$ is not <sup>subset</sup> equivalent to $s_2$ .
set1 > set2	# $s_1$ is proper subset of $s_2$ .
set1 >= set2	# $s_1$ is superset of $s_2$ .
set1 < set2	# $s_1$ is proper superset of $s_2$
set1   set2	# the union of $s_1$ and $s_2$
set1 & set2	# the intersection of $s_1$ and $s_2$
set1 - set2	# the set of elements of $s_1$ but not $s_2$
set1 ^ set2	# the set of elements in precisely one of $s_1$ or $s_2$ .

remove()

→ remove() used to remove a specific element

→ If the element is not available in the given set remove() through KeyError.

discard()

→ discard function allow an element as an argument.

→ It remove the specific element.

→ If is element is not available in the given set discard() doesn't through any error.

## Set Unpacking

python Narayana

(173)

```
>>> se = {1, 2, 3, 'x', True}
```

```
>>> a, b, c, d, e = se
```

```
>>> print(a)
```

1

```
>>> type(a)
```

<class 'int'>

```
>>> print(b)
```

2

```
>>> type(b)
```

<class 'int'>

```
>>> print(c)
```

3

```
>>> type(c)
```

<class 'int'>

```
>>> print(d)
```

x

```
>>> type(d)
```

<class 'str'>

## Set Packing

```
>>> a = 10
```

```
>>> b = 20
```

```
>>> c = 30
```

```
>>> d = 'a'
```

```
>>> se = {a, b, c, d}
```

Python Narayana

(174)

```
>>> print(se) {10, 'a', 20, 30}
>>> type(se) <class 'set'>
>>> id(se) 105865744
```

### Using Set in List

```
>>> lst = [10, 20, 30, {True, 2, False, 3}, {1, 2, {100, 200, 300}}]

>>> print(lst) [10, 20, 30, {False, True, 2, 3}, {1, 2, {100, 200, 300}}]

>>> len(lst) 5
>>> lst[0] 10
>>> lst[1] 20
>>> lst[2] 30
>>> lst[3] {False, True, 2, 3}
>>> len(lst[3]) 4
>>> lst.index({False, True, 2, 3}) 3
>>> lst[3][0] TypeError: 'Set' object does not support indexing.
>>> lst[4] [1, 2, {100, 200, 300}]
>>> len(lst[4]) 3
>>> lst[4][0] 1
>>> lst[4][1] 2
>>> lst[4][2] {200, 100, 300}
```

Python Narayana  
175

```

>>> lst[4].index(1) 0
>>> lst[4].index(2) 1
>>> lst[4].index({200, 100, 300}) 2
>>> lst[4].index({200, 300, 100}) 2
>>> lst[4].index({300, 100, 200}) 2

>>> lst[0:2] [10, 20]
>>> lst[0:3] [10, 20, 30]
>>> lst[0:4] [10, 20, 30, {False, True, 2, 3}]
>>> lst[0:5] [10, 20, 30, {False, True, 2, 3},
 [1, 2, {200, 100, 300}]].

```

Ex:

>>> lst[0 : lst[5][2]]

IndexError: list index out of range

Note: the given list has index number upto 4 only, but we are trying to access 5, which is not available.

\* Using Set in the tuple\*

e.g. >>> t = ( 11, True, {2, 3, 1}, 12, 13, (10, 20, [100, 200, 300], {25, 35, 45}))

>>> print(t) ( 11, True, {1, 2, 3}, 12, 13, (10, 20, [100, 200, 300], {25, 35, 45}))

>>> len(t) 6

>>> t[0] 11

## python Numpy array

(176)

>>> t[1] True.  
>>> t[2] {1, 2, 3}.  
>>> t[2][0] TypeError: 'set' object doesn't support indexing.  
>>> t[2][1] TypeError: 'set' object doesn't support indexing.  
>>> t[2][2] TypeError: 'set' object doesn't support indexing.  
>>> t[3] 12  
>>> t[4] 13.  
>>> t[5] (10, 20, {100, 200, 300, 225, 35, 45})  
Numpy array  
>>> t[5][0] 10  
>>> t[5][1] 20  
>>> t[5][2] 200  
>>> t[5][2][0] 300.  
>>> t[5][2][3][0] TypeError: 'set' object does not support indexing.  
>>> t[5][2][3][2] TypeError: 'set' object doesn't support indexing.  
>>> t[0:2] (11, True)  
>>> t[0:4] (11, True, {1, 2, 3}, 12)  
>> t.index(1, 3, 2) 2  
>> len(t[5]) 3  
>>> t[1:t[5][0]] (True, {1, 2, 3}, 12, 13, (10, 20, {100, 200, 300, 225, 35, 45}))

# \* Dictionary - Data Structure

(17)

## Introduction :-

1. Dictionary is an ~~order~~ ordered set of key: value pairs, here keys are unique.
2. A pair of braces creates an empty dictionary: {}
3. The main operations on a dictionary are storing a value with some key and ~~extracting~~ extracting the value given the key.
4. Dictionary keys ~~get~~ are not allowed duplicates, but dictionary values are allowed duplicate values.
5. Insertion order is not preserved.
6. we can use homogeneous and heterogeneous elements for both keys & values.
7. Dictionary w/ keys are immutable and values are mutable
8. Dictionary will not allow indexing & slicing.
9. Dictionaries are indexed by keys. which can be any immutable type, strings, and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers. (8) tuples.

→ If a tuple contains any mutable object either directly (or) indirectly, it can't be used as a key. You can't use list as keys, since lists can be modified in place using index assignments, (or) methods like append() and extend().

We can create dictionary in different ways.

1. Creating empty dictionary and adding key:

Value pairs:-

Eg:-

```
>>> dic1 = {}.
>>> print(dic1)
{}

>>> type(dic1)
<class 'dict'>

>>> dic1 ['a'] = 10 #adding key: Value pair to dictionary

>>> dic1 ['b'] = 20 #adding key: Value pair to dictionary

>>> dic1 ['c'] = 30 #adding key: value pair to dictionary

>>> dic1 ['a'] = 10 # trying to add same key:

 Value pair:

>>> dic1 ['a'] = 50 #adding new value for same

 key 'a'.

>>> print(dic1)
{'a': 50, 'b': 20, 'c': 30}.
```

## 2. Creating a dictionary with dict() function

Eg:-

(179)

```
>>> dic1 = dict([('a', 10), ('b', 20), ('c', 30), ('d', 40)])
>>> print(dic1)
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> type(dic1)
<class 'dict'>
```

## 3. Creating a dictionary with curly braces including key : value pairs;

Eg:-

```
>>> dic1 = {'a': 10, 'b': 20, 'c': 30, 'd': 40}.
>>> print(dic1)
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> type(dic1)
<class 'dict'>
```

## Accessing data from dictionaries

We can assign values to the keys and later we can fetch values by using keys.

Eg:-

```
>>> stuDetails = {'id': 100, 'Name': 'Sai', 'Age': 20, 'Marks': 90}
```

```
>>> print(stuDetails)
{'id': 100, 'Name': 'Sai', 'Age': 20,
'Marks': 90}
>>> type(stuDetails)
<class 'dict'>
```

```
>>> print(stuDetails['id'])
100
```

```
>>> print(stuDetails['Age'])
20.
```

Eg:2

```
>>> studDetails = { 'id':100, 'Name': 'sai', 'Subjects': ['SQL Server', 'Oracle', 'Python'] }.
```

```
{'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server', 'Oracle', 'Python']}
```

```
>>> studDetails {'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server', 'Oracle', 'Python']}
```

```
>>> studDetails['id'] 100
```

```
>>> studDetails['Name'] 'Sai'.
```

```
>>> studDetails['Subjects'] ['SQL Server', 'Oracle', 'Python'].
```

```
>>> studDetails['Subjects'][0] 'SQL Server'
```

```
>>> studDetails['Subjects'][1] 'Oracle'.
```

```
>>> studDetails['Age'] KeyError('age') # this key not available.
```

To prevent this type of error we can check whether the specified key existed or not by using has-key().

But this "has-key()" is available in "python 2" only not in python 3 version.

```
>>> studDetails.has-key('Age') False # in python 2 version.
```

In "python 3" we have to check by using "membership operator 'in'"

```
>>> 'Age' in studDetails False # in python 3 version.
```

## Adding new key & value pairs:

Python Narayana.

(181)

```
{'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server', 'Oracle', 'python']}.
```

If we need to add another element then.

```
>>> stuDetails['Age'] = 25 # adding new element
 to the stuDetails Dictionary
```

```
>>> print(stuDetails)
```

```
{'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server',
 'Oracle', 'python'],
 'Age': 25}.
```

## Updating dictionary values

If we need to change the age from 25 to 27 then.

```
>>> stuDetails['Age'] = 27. # changing the age from 25 to 27
```

```
>>> print(stuDetails)
```

```
{'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server', 'Oracle',
 'python'], 'Age': 27}.
```

## Deleting key: value pairs from Dictionary

If we need to delete the value 27 from the above dictionary then (by using pop).

```
>>> stuDetails.pop('Age') 27.
```

```
>>> print(stuDetails).
```

```
{'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server', 'Oracle', 'python']}
```

We can also delete by using `popitem()` Python Narayana  
B2

If this function removes last key-value pair in the dictionary.

```
>>> studDetails.popitem() ('Subject', [SQL Server, Oracle, Python])
```

If we need to delete all key-value pairs then we use `clear()`

```
>>> studDetails.clear() # deleting all pairs, then we can have an empty dict.
```

```
>>> print(studDetails) {}.
```

## Dictionary functions:

1. keys() :- It returns all keys from dict

Eg:-  
>>> dic1 = {1: 'python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}.

```
>>> dic1.keys() dict_keys([1, 2, 3, 4, 's'])
```

2. Values() :- It returns all values from the dict.

Eg:-

```
>>> dic1 = {1: 'python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}.
```

```
>>> dic1.values() dict_values(['python', (3+5j), (10, 20, 30), [100, 'a', False], 100]).
```

3) `Copy()` :- it copies the dict into new dict.

(183)

Eg:-  
`>>> dic1 = {1: 'python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'd', False], 's': 100}.`

`>>> dic2 = dic1.copy()`

`>>> dic1` {1: 'python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'd', False], 's': 100}.

`>>> dic2` {1: 'python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'd', False], 's': 100}.

4) `pop()` :- it removes Specific key value pair.

Eg:-  
`>>> dic1 = {1: 'python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'd', False], 's': 100}.`

`>>> dic1.pop(1)` 'python'.

`>>> print(dic1)` {2: (3+5j), 3: (10, 20, 30), 4: [100, 'd', False], 's': 100}.

Note: We can also remove the key:value pair by using 'del' command.

Eg:-  
`>>> del dic1[2]`

`>>> print(dic1)` {3: (10, 20, 30), 4: [100, 'd', False], 's': 100}.

5) `popitem()` :- It will not allow any argument  
but it will remove the last key value pair from the  
existing dict. 18A

```
>>> d = {`id':1, 'Name':'Teja'}
```

```
>>> d.popitem()
```

```
>>> ('Name', 'Teja')
```

```
>>> d
```

```
{`id':1}
```

6) `clear()` :- It removes all key:value pairs from dict.

Eg:-

```
>>> tup = (1,2,3,4,5)
```

# creating tuple

```
>>> dic1 = dict(tup)
```

# taking tuple elements  
as keys in the dict dic1

{1:None, 2:None, 3:None, 4:None, 5:None}

\* By default values are "None", if we need to get '0'  
as default then.

```
>>> tup = (1,2,3,4,5)
```

```
>>> dic = {3: None, 4: 0}
```

{1:0, 2:0, 3:0, 4:0, 5:0}

We can also use last elements as keys in the dict and  
the elements must be unique

```
>>> lst = [1, 2, 3, 4, 5]
```

python Narayana

# creating list

```
>>> dict2.fromkeys(lst)
```

# taking list elements  
as keys in the dict dict2

{ 1: None, 2: None, 3: None, 4: None, 5: None }.

7. get() - this function is used to get the value  
of Specified key.

Eg:

```
>>> stuDetails = { 'id': 100, 'Name': 'Sai', 'Subjects': ['SQL server',
'Oracle', 'Python'], 'id': 1000, 'name': 'Mani' }.
```

```
>>> stuDetails.get('id') 100
```

```
>>> stuDetails.get('Subjects') ['SQL Server', 'Oracle', 'Python']
```

8. Items() - this function is used to get all items.  
All key and value pairs will be displayed in tuple  
format.

Eg:

```
>>> stuDetails = { 'id': 100, 'Name': 'Sai', 'Subjects': ['SQL server',
'Oracle', 'Python'], 'id': 1000, 'name': 'Mani' }.
```

```
>>> stuDetails.items()
```

```
dict_items([(id, 100), (Name, 'Sai'), (Subjects, ['SQL server',
'Oracle', 'Python']), (id, 1000), (name, 'Mani')])
```

8) update() → the current dictionary will be updated with all key: value pairs from another dictionary. (86)

```
>>> stuDetails = { 'id': 100, 'Name': 'Sai', 'Subjects': ['SQL Server', 'Oracle', 'Python'] }
```

```
>>> stuDetails1 = { 'id': 1000, 'name': 'nani' }.
```

```
>>> stuDetails.update(stuDetails1)
```

```
>>> print(stuDetails)
```

```
{'id': 100, 'Name': 'Sai', 'Subjects': ['sqlserver', 'Oracle', 'python']}, {'id': 1000, 'name': 'nani'}
```

How to perform arithmetic operations on the values of a dictionary?

```
>>> d1 = { 'Sub1': 80, 'Sub2': 90, 'Sub3': 70, 'Sub4': 80 }
```

```
>>> print(d1) { 'Sub1': 80, 'Sub2': 90, 'Sub3': 70, 'Sub4': 80 }
```

```
>>> s = sum(d1.values())
```

```
>>> print(s) 320.
```

```
>>> mx = max(d1.values())
```

```
>>> print(mx) 90
```

```
>>> mn = min(d1.values())
```

```
>>> print(mn) 70
```

>>> cnt = len(d1.values())

python Narayana

>>> print(cnt)

4.

(187)

difference b/w pop() and popitem()

<u>pop()</u>	<u>popitem()</u>
<ul style="list-style-type: none"><li>→ pop() function allows one argument</li><li>→ It will delete the key value pair based on the given key as an argument.</li></ul>	<ul style="list-style-type: none"><li>→ popitem() doesn't allow any argument</li><li>→ It will delete the last key:value pair from existing dictionary.</li></ul>
Difference b/w append(), extend() and insert() in <u>List</u>	

append()

extend()

insert()

All datatypes - mutable & immutable

(188)

class	Description	Mutable	Immutable
int	integer value		✓
str	character string		✓
float	floating-pt number		✓
Bool	Boolean value		✓
Complex	Complex value		✓
List	Sequence of objects	✓	
Set	Ordered set of unique objects	✓	
Frozenset	immutable form set		✓
Tuple	Sequence of objects		✓
dictionary	key: value pairs	✓	

Eg:-

# Create a mapping of state to abbreviation.

states = { 'Oregon': 'OR', 'Florida': 'FL', 'California': 'CA',  
'New York': 'NY', 'Michigan': 'MI' }

# Create a basic set of states and some cities in them

cities = { 'CA': 'San Francisco', 'MI': 'Detroit', 'FL': 'Jacksonville' }.

## Python Dictionary

(189)

# add some more cities.

cities['NY'] = 'New York'.

cities['OR'] = 'Portland'

# print out some cities.

```
print ('-' * 20)
```

```
print ("NY state has:", cities['NY'])
```

```
print ("OR state has:", cities['OR'])
```

# print some states.

```
print ('-' * 20)
```

```
print ("Michigan's abbreviation is:", states['Michigan']).
```

```
print ("Florida's abbreviation is:", states['Florida']).
```

# do it by using states then cities dict

```
print ('-' * 20)
```

```
print ("Michigan has:", cities[states['Michigan']]).
```

```
print ("Florida has:", cities[states['Florida']]).
```

# print every state abbreviation

```
print ('-' * 20)
```

```
for state, abbrev in states.items():
```

Python Narayana

print ("%s has the city %s" % (abbrev, city)).

(90)

# now do both at the same time

print ('-' \* 20)

for state, abbrev in states.items():

print ("%s state is abbreviated %s and has city %s" %  
(state, abbrev, cities[abbrev])).

print ('-' \* 20)

# Safely get an abbreviation by state that might  
not be there.

state = states.get('Texas', None)

if not state:

print ("Sorry, no Texas.")

# get a city with a default value

city = cities.get('Tx', 'Does Not Exist')

print ("The city for the state 'Tx' is : %s" % city)

Output:-

NY state has : New York

OR state has : portland

Michigan's abbreviation is : MI

Florida's abbreviation is : FL

Michigan has : Detroit

Florida has : Jacksonville.

Oregon is abbreviated OR

Florida is abbreviated FL

California is abbreviated CA.

New York is abbreviated NY

Michigan is abbreviated MI

CA has the city San Francisco.

MI has the city Detroit.

FL has the city Jacksonville

NY has the city New York.

OR has the city Portland.

Oregon state is abbreviated OR and has city Portland.

Florida state is abbreviated FL and has city Jacksonville.

California state is abbreviated CA and has city San Francisco.

New York state is abbreviated NY and has city New York.

Michigan state is abbreviated MI and has city Detroit

Sorry, no Texas.

The city for the state 'Tx' is: Does not Exist

(192)

### Setdefault()

d.setdefault(k,v)

If the key is already available then this function returns the corresponding value

If the key is not available then the specified key-value will be added as new item to the dictionary

1. d = {100: "durga", 200: "ravi", 300: "shiva"}.

2. print(d.setdefault(400, "pavan"))

3. print(d)

4. print(d.setdefault(100, "saethin"))

5. print(d)

6.

7. Output

pavan

8. {100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}

9. durga

10. {100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}

11. {100: 'durga', 200: 'ravi', 300: 'shiva', 400: 'pavan'}

12. Update ()

d.update(x)

All items present in the dictionary x will be added to dictionary d

Python Narayana

Q. Write a program to take dictionary from the keyboard & print the sum of values? (193)

1. d = eval(input('Enter dictionary:'))

2. S = sum(d.values())

3. print("Sum = ", S)

4.

5. Output

6. D:\python-classes>py test.py

7. Enter dictionary : { 'A':100, 'B':200, 'C':300 }

8. Sum = 600.

Q. Write a program to find number of occurrences of each letter present in the given string?

1. word = input("Enter any word:")

2. d = {}

3. for x in word:

4.     d[x] = d.get(x, 0) + 1

5. for k, v in d.items():

6.     print(k, "occurred", v, "times")

7.

8. Output

9. D:\python-classes>py.test.py

10. Enter any word : mississippi

11. m occurred 1 time

12. i occurred 4 times

13. s occurred 4 times

14. p occurred 2 times

Q) Write a program to find number of occurrences of each vowel present in the given string? 19H

1. word = input("Enter any word:")
2. vowels = { 'a', 'e', 'i', 'o', 'u' }
3. d = {}
4. for x in word:
5.     if x in vowels:
6.         d[x] = d.get(x, 0) + 1
7.     for k, v in sorted(d.items()):
8.         print(k, "occurred", v, "times")
- 9.
10.      Output
11.      D:\python-classes>py test.py
12.      Enter any word: doganimaldoganimal
13.      a occurred 4 times.
14.      i occurred 2 times.
15.      o occurred 2 times.

Q) Write a program to accept student name and marks from the keyboard and creates a dictionary. Also display student marks by taking student name as input?

1. n = int(input("Enter the number of students:"))
2. d = {}
3. for i in range(n):

4. name = input("Enter student Name")  
5. marks = input("Enter Student Marks")  
6. d[name] = marks  
7. while True:  
8. name = input("Enter Student Name to get Marks :")  
9. marks = d.get(name, -1)  
10. if marks == -1:  
11. print("Students Not Found")  
12. else:  
13. print("The marks of", name, "are", marks)  
14. option = input("Do you want to find another Student  
marks [yes/No] ")  
15. if option == "No":  
16. break  
17. print("Thanks for using our application")  
18.  
19. Output:  
20. D:\Python-classes>py test.py  
21. Enter the number of students: 5  
22. Enter Student Name: Barthoshi  
23. Enter Student Marks: 80  
24. Enter student Name: DhanaLakshmi  
25. Enter Student Marks: 80  
26. Enter Student Name: Narayana  
27. Enter Student Marks: 70

(195)

Python Narayana

(96)

- 28) Enter student Name : pinay
- 29) Enter student Marks : 60.
- 30) Enter student Name : Vinny.
- 31) Enter student Marks : 50
- 32) Enter student Name to get marks : Narayana.
- 33) The marks of Narayana are 70.
34. Do you want to find another student marks [yes/no] yes
- 35 Enter student Name to get Marks : Srinu
- 36 Student Not found
- 37 Do you want to find another student marks [yes/no] no
38. thanks for using our application.

## Dictionary Comprehension

Comprehension concept applicable for dictionaries also

1. Squares = { $x : x * x$  for  $x$  in range(1, 6)}
2. print(Squares)
3. doubles = { $x : 2 * x$  for  $x$  in range(1, 6)}
4. print(doubles)
- 5.
6. Output
7.  $\{1: 1, 2: 4, 3: 9, 4: 16, 5: 25\}$
8.  $\{1: 2, 2: 4, 3: 6, 4: 8, 5: 10\}$

## Assignment - 2

Python Narayana

(197)

1. what are the different types of data structures in python?

A.

2. what is a string? what is the data type to represent string value?

A.

3. what is a list? what is the data type of list? Create one list?

A.

4. what is a Tuple? what is the data type? Create one tuple?

A.

5. What is a set? what is the data type? python varagious  
Create One set?

A. (198)

6. what is dictionary? what is the data type of it?  
Create one dictionary?

A.

7. what is mutable and immutable? Explain each?

A.

8. Is string mutable or immutable? Explain each?

A.

9. what are the starting index numbers for both forward  
& backward indexing?

A.

10. what is indexing and slicing? *Python Narayana*  
A. *Differences b/w both* (199)

11. What are the IndexError and TypeError errors?

A.

12. st = 'python developer'

a. Display as 'python developer'

b. Display as 'python Developer'

c. Count number of e's in that string.

d. Find index position of first 'o' in the above string

e. Find index position of second 'o' in the above string.

f. split the same string in to two elements like  
'python', 'developer'. (100)

g. Display string in reverse.

h. from the above string, display only 'on dev'

i. from above string remove 'then'

j. copy the same string into other new strings like  
str1, str2, and str3 at a time.

13. generate a list [10, 12, 14, 16, 18, 20] by using range function?

14. Lst = [10, 20, 'python', 10.5, 1, 10, True, False, 0]  
what are the results of?

a. How to display the count of elements in the  
above list?

b. How to count no. of occurrences of '10' in the above list?

c. what are the no. of occurrences '1' in the above list?

d. How to add complex number  $10+5j$  to the above list?

e. How to add both Complex number  $1+2j$  and float value 1.3 to the above list

f. How to add bool value 'True' between 10 and 20 in the above list?

g. How to remove the str value 'python' from the above list?

h. How to reverse the above list?

i. How to copy the above list list to another list? Python Lists

(102)

j. Display index position of float value 10.5 in the above list?

k. Replace float value 10.5. with 20.5.

l. what are the differences between append() & extend() in list?

m. what is the 'Del' command ? Explain with one example?

n. How to convert a list to string? Give one example?

python Narayana

18. How to convert a string to list? Give one example?  
203

19. How to convert list to tuple? Give one example?

20. How to convert tuple to list? Give one example?

21. How to convert tuple to string? Give one example?

22. How to convert string to tuple? Give one example?

23. What is tuple? Does tuple allow duplicate elements?

24. Can we add new element to the existing set?  
How can we add?

25. Can we add new elements to the frozenset? If no,  
give one example?

(04)

26. Let's take two sets.

$$set_1 = \{1, 2, 3, 4, 5\}$$

$$set_2 = \{1, 2, 3, 4, 5\}$$

1. perform Union between set<sub>1</sub> and set<sub>2</sub>, by using '+' also

2. perform intersection between set<sub>2</sub> and set<sub>1</sub>, by using '&' also.

3. Perform difference b/w set<sub>2</sub> and set<sub>1</sub> by using '-' also.

4. what intersection\_update? and perform between set<sub>1</sub>  
and set<sub>2</sub> & vice versa.

5. what is difference-update? and perform between set  
and set and vice versa.

205

6. what is symmetric-difference? perform between set and  
set and vice versa.

7. what is symmetric-difference-update? perform between  
set and set and vice versa.

27. Difference between remove() and discard()?

28. Is frozenset mutable or immutable?

29. what is the difference result of `isdisjoint()` if one set contains elements and other set is empty?

(Ans)

30. what is the difference between '`=`' and '`==`'? explain with one example?

31. How to check whether the specific element is existed or not in the given list? Example:

32. `>>> set1 = {11, 12, 13, 14, 15}.`

`>>> set2 = {11, 12, 13, 16, 17}` what is the result of

a. `set1 == set2?`

b. `set1} == set2.`

c. `set1 <= set2.`

Python Narayana.

d.  $s_1 < s_2$ c.  $s_2 > s_1$ .f.  $s_1. \text{isdisjoint}(s_2)$ 

33. In what cases isdisjoint method will be True? Give Examples?

34.  $\ggg s_1 = \{1, 2, 3\}$ . $\ggg s_2 = \{1, 2, 3, 4\}$  what is the result of.a.  $s_1 < s_2$ .b.  $s_1 \leq s_2$ c.  $s_1 > s_2$

python Narayana: 8

208

d.  $s_1 \geq s_2$ .

e.  $s_1.$  isdisjoint ( $s_2$ )

f.  $s_1.$  issubset ( $s_2$ )

g.  $s_1.$  issuperset ( $s_2$ )

h.  $s_2.$  issubset ( $s_1$ )

i.  $s_2.$  issuperset ( $s_1$ )

j.  $s_1 \downarrow = s_2$ .

k.  $4$  not in  $s_2$  ?

l.  $1$  in  $s_2$  ?

35. `>>> s1 & s2.`

python Narayana

(209)

`>>> s2 & s3` what is the result of

a.  $s_1 < s_2$

b.  $s_1 \leq s_2$

c.  $s_1 > s_2$ .

d.  $s_1 \geq s_2$

e.  $s_4 >= s_2$ .

f. `s1.isdisjoint(s2)`

g. `s1.issuperset(s2)`

h. `s2.issubset(s1)`

i. `s2.issuperset(s1)`

36. `>>> s1 = {Tree, False, 10, 20, 3.5}`. python Narayana

`>>> s2 = {1, 0}` what is the result of

(20)

a. `s2. issubset(s1)`

b. `s1. issubset(s2)`

c. `s1. issuperset(s2)`

d. `s2. issuperset(s1)`

e. `s2 > s1`

f. `s1 < s2`

g. `s1 >= s2`

h. `s2 <= s1`

i. `s2 = s1`

37. Can we use duplicate keys in dictionaries?

38. `>>> set1 = {10, 20, 30, 40, 50}` python Narayana  
`>>> set2 = {10, 20, 30, 60, 70}.`

(21)

- a. Display all distinct elements from both sets.
- b. Display only non-matching elements from  $S_1$  Set.
- c. Display only non-matching elements from both sets.
- d. store the common elements from both sets into set 1
- e. Remove matching elements from set1.
- f. store nonmatching elements from both sets into set 2.

g. How to take backup of set1 to new set set3? Python Narayana

(212)

39. Create one empty dictionary and add any 5 key-value pairs?

40. Can we list and tuple as value in dictionary? if yes then create one dictionary with tuple & list as values?

41. `StuDetails = { 'id': 100, 'Name': 'Sai', 'Subjects': ['SQLServer', 'Oracle', 'python'] }`

a. modify the value of name as 'Durga'

b. add new pair age: 25

c. display all keys from the StuDetails dictionary?

d. Display all values from the StuDetails dictionary?

e. Display all subjects which are stored in the StuDetails dictionary?

f. Remove name key from the StuDetails?

g. Create new dictionary from the tuple element?

Eg: tup = (1, 2, 3, 4, 5)

1. Store value 'Sai' for key 1
2. Store value 'Mahesh' for key 2.
3. Store value Tree for key 3.
4. Store value 3+6j for key 4.
5. Store Value 1000 for key 5.

b. remove all pairs from the above dictionary.

42. st = 'python narayana'

How to get the following from the above string

- a st[3::5]
- b st[1::1]
- c st[-1::-3]
- d st[2::-1]
- e st[1:: 15].

Python Narayana

# \* FAQ's On Data Structures \*

(25)

1. How display the given string in Lower case?
2. What is the purpose of Capitalize()
3. How to display the given string st="python'Dev" as PYTHONDev
4. Is string mutable or immutable object?
5. How to represent an empty string?
6. What is the output format of split function?
7. How to check whether the given string is lower case or not?
8. How to find the index number of second occurrence 'o' in "Oracle developer"?
9. Can we add a new string to the existing string?
10. What is a list and what is the datatype of list?
11. How to add a new element to the existing list?
12. What is the insert()?
13. What is the difference b/w append() and extend()?
14. Can we delete a specific character of a string by using del?
15. Can we remove all characters from a string by using clear()?

16. Is list mutable or immutable object?
17. How can we decide whether a specific <sup>(Q6)</sup> Data Structure is mutable or immutable?
18. Can we add multiple elements to the existing list at required place?
19. What is the difference b/w remove() and pop()?
20. How to know the length of a list?
21. Is insertion order preserved or not in list?
22. What is range() and what is required argument(s) of range()?
23. What are different ways to create a list?
24. Can we give duplicate elements in list?
25. How to convert a list into string?
26. What is a tuple?
27. Difference b/w tuple and list?
28. How to create a tuple with one element?
29. Can we create a tuple without () & tuple ()? If yes then how?
30. Can we give duplicate element in the tuple?
31. How to add a group of elements to the existing tuple?

python Narayana.

- 32 Can we remove elements from tuple?
- 33 What is the difference b/w del command & clear()?
- 34 Can we delete a specific element of list using del command? if yes then how?
- 35 Can we delete a specific element from the tuple by using del Command? if yes, then how?
- 36 what is any() and all()?
- 37 what is a set?
- 38 Is insertion order preserved or not in tuple?
39. Is insertion order preserved or not in Set?
- 40 How to add new element to the existing set?
- 41 Can we add duplicate elements to the existing set?
42. Does set allows duplicate elements?
43. What is issubset() and issuperset()?
- 44 What is the difference b/w remove() and discard()?
- 45 What is the main purpose of set?
- 46 If a list contains duplicate elements then how to remove duplicate elements from the list?
- 47 How to Convert a list into set?
- 48 How to Convert a list into string?
- 49 Can we Concatenate two sets?

- 50 Can we multiply a set 'n' number of times?  
51 Can we use a list in set? (218)
- 52 Can we concatenate a string & a set?
- 53 How to display [10, 30] from [1, 3, 10, 5, 7, 30, 15]?
- 54 How to reverse a string? Example?
- 55 Can we use a set in tuple?
- 56 How to display a string in ascending order (a-z)?
- 57 How to display a string in descending order (z-a)?
- 58 How to remove a part of strings?  
Cg: st = 'oracle dev' o/p: 'Oracle'.
- 59 How to reverse a list without using `reverse()`?
- 60 What is the difference b/w list and set?
- 61 What is the main purpose of tuple?
- 62 What is dictionary?
- 63 What is the main purpose of dict?
- 64 Can we add a new key: value pair to the existing dict? How?
- 65 How to access the value of specific key from dict?

66. How to remove a specific key: value pair from the dict?

(219)

67. How to update the existing value of a specific key in the dict?

68. How to display all keys from the dict?

69. How to get all values from the dictionary?

70. What are the different ways to create a dict?

71. How to create a dict by using an existing tuple?

72. How to update a dict by using another dict?

73. What is the copy()?

74. What is the get() in dict?

75. If two set are empty then what is the result of issubset() and issuperset()?

76. If two sets are having different elements then what is the output of issubset() and ~~issuperset~~ issubset()

77. What is the difference b/w intersection() & intersection\_update().

78. How to create an empty set?

79. What are different ways to create a string?

80. Is dict mutable or immutable?

## python Narayana

- 81 what the replace()?
- 82 can we give a duplicate key in dict? (20)
83. what is can we delete a specific key:value pair by using del command?
- 84 Can we delete a specific element of a set by using del?
- 85 what is the indexing & slicing?
- 86 Can we generate a heterogeneous list by using range()?
- 87 How to represent an empty set?
- 88 What is the list packing & list unpacking?
- 89 How to perform string packing?
- 90 Can we Concatenate two dict?
- 91 Is insertion order preserved or not in dict?
- 92 what is difference b/w title() and Capitalize()?
- 93 what is swapCase()?
- 94 what is the difference b/w pop() and popitem()
- 95 can we create a new dict by using existing string?
- 96 when we create a new dict by using an existing obj  
then what is the default value for all keys?
- 97 what is the purpose of join()?
98. Can we Count the number of occurrence of a specific

- elements in the set? python Narayana
99. what is the default delimiter in the split()?
- 100 why set is not supporting indexing?

Python  
Narayana

Python Narayana

222

Python Narayana

## \* Operators \*

→ An operator is a symbol that tells the compiler to perform certain mathematical (or) Logical manipulations.

→ Operators are used in program to manipulate data & variables.

Python Language supports the following types of operators.

1. Arithmetic operators
2. Assignment operators
3. Comparison (i.e., Relational) operators
4. Logical operators.
5. Bitwise operators
6. Membership operators
7. Identity operators.

### Arithmetic operators:-

Assume variable  $a$  holds 4 and variable  $b$  hold 2, then,

## Arithmetic operators

python Narayana

operator	Description	Example
+	Addition - Adds values on either side of the operator	$a+b=6$
-	Subtraction - Subtracts right hand operand from left hand operands	$a-b=2$
*	Multiplication - Multiplies values on either side of the operator	$a*b=8$
/	Division - Divides left hand operand by right hand operand	$a/b=2$
%	Modulus - Divides left hand operand by right hand operand and returns remainder	$a \% b=0$
**	Exponent - performs exponential (power) calculation on operators	$a**b$ (b to the power of a) 16.
//	floor division - The division of operands where the result is in quotient in which the digits after the decimal point are removed	$9//2$ is equal to 4 and $9.0//2.0$ is equal to 4.0.

## Comparison operators:-

python Narayana

225

Example

Operator	Description	Example
$= =$	checks if the value of two operands are equal (or) not if yes then Condition is True	$(a == b)$ is not True
$\neq$	checks if the value of two operands are not equal if values are not equal then Condition become true	$(a \neq b)$ is true.
$>$	checks if the value of left operand is greater than the value of right operand if yes then Condition True	$a > b$ is not True
$<$	checks if the value of left operand is less than the value of right operand if yes then Condition becomes true.	$(a < b)$ is true.
$> =$	checks if the value of left operand is less greater than (or) equal to value of right operand, if yes then Condition becomes true	$(a >= b)$ is not True

operator	description	Example.
$<=$	checks if the value of left operand is less than (or) equal to the value of right operand if yes then condition becomes true	(126) $(a \leq b)$ is true

## Assignment operators:

operator	Description	Example
$=$	Simple assignment operator, Assigns value from right side operands to left side operand	$c = a + b$ will assign value of $a + b$ into $c$ .
$+=$	Add AND assignment operator, it adds right operand to the left operand and assign the result to left operand	$c += a$ is equivalent to $c = c + a$ .
$-=$	Subtract AND assignment operator it subtracts right operand from the left operand and assign the result to left operand	$c -= a$ is equivalent to $c = c - a$ .
$*=$	Multiply AND assignment operator, it multiplies right operand with the left operand & assign the result to left operand	$c *= a$ is equivalent to $c = c * a$ ,

$/=$  Divide AND assignment operator  
It divides left operand with  
the right operand and assign  
the result to left operand

(227)  
 $C/a$  is equivalent  
to  $C = C/a$ .

$\% =$  Modulus AND assignment operator,  
it takes modulus using two  
operands and assign the result to  $C = C \% a$ .  
to left operand

$** =$  Exponent AND assignment  
operator , performs Exponential  
(power) Calculation on operators  
on operators and assign value  
to left operand

$C ** a$  is  
equivalent to  
 $C = C ** a$ .

$//=$  floor division and assigns a  
value to performs floor division  
on operators and assign value  
to the left operand

$C // a$  is  
equivalent  
 $C = C // a$ .

Python Narayana

Eg:1 $a = 20$  $b = 10$  $c = 0$ 

```
print ('a value is', a)
```

```
print ('b value is', b)
```

 $c += a$ 

```
print ("Add AND - value of c is", c)
```

 $c *= a$ 

```
print ("Multiply AND - value of c is", c)
```

 $c /= a$ 

```
print ("Divide AND - value of c is", c)
```

 $c = 2$  $c \% 2 = a$ 

```
print ("Modulus AND - value of c is", c)
```

 $c **= a$ 

```
print ("Exponent AND - value of c is", c)
```

 $c //= a$ 

```
print ("Floor Division AND - value of c is", c)
```

 $c -= a$ 

```
print ("Subtract AND - value of c is", c)
```

Output:

a value is 20.

b value is 10

Add AND - value of c is 20

Multiply AND - value of c is 400

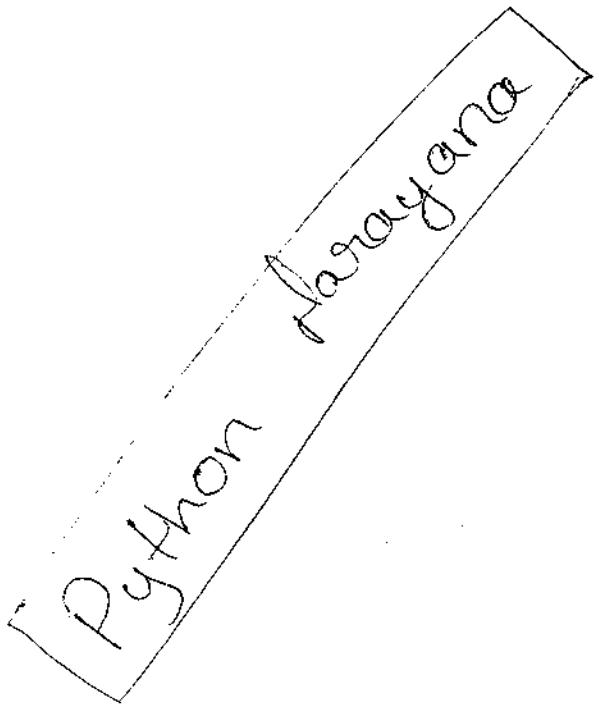
Divide AND - value of c is 20.0

Modulus AND - value of c is 2

Exponent AND - value of c is 1048576.

Floor Division AND - value of c is 52428.

Subtract AND - value of c is 52408.



## Bitwise Operators

python Narayana

(230)

Bitwise operators works on bits and performs bit by bit operation. Assume if  $a=60$ , and  $b=13$  Now in binary format they will be as follows.

$$a = 0011 \ 1100$$

$$b = 0000 \ 1101$$

$$a \& b = 0000 \ 1100$$

$$a/b = 0011 \ 1101$$

$$a \wedge b = 0011 \ 0001$$

$$\sim a = 1100 \ 0011$$

there are following bitwise operators supported by python language.

Operator	Description	Example
&	Binary AND operator copies a bit to the result if it exists in both operands	$(a \& b)$ will give 12 which is 0000 1100.
	Binary OR operator copies a bit if it exists in either operand	$(a b)$ will give 61 which is 0011 1101
^	Binary XOR operator copies bit if it is set in one operand but not both	$(a \wedge b)$ will give 49 which is 0011 0001

## Python Narayana

(23)

~ Binary ones complement operator ( $\sim a$ ) will give -61  
is unary and has the effect of 'flipping' bits  
which is 1100 0011 in 2's complement form  
due to a signed binary no.

<< Binary left shift operator. The left operand's value is moved left by the number of bits specified by the right operand  
acc << 2 will give 240 which is 1111 0000

>> Binary Right shift operator. the left operand's value is moved right by the number of bits specified by the right operand  
a >> 2 will give 15 which is 0000 1111.

## Logical Operators

These are following Logical operators supported by Python Language.

These are NOT, AND and OR

NOT

X

NOT X

True

false

false

True

# python Narayana

AND

X	y	X AND y.
True	false	false
false	True	false
True	True	True
false	false	false

(23)

OR

X	y	X OR y.
True	false	True
false	True	True
True	True	True
false	false	false

Eg:

```
>>> a=10
>>> b=20
>>> a == 10 and b == 20 True
>>> a == 20 and b == 10 False
>>> a == 10 and b == 10 False
>>> not a == 20 and b == 20 True
>>> not a == 11 and not b == 21 True
>>> a == 20 (or) b == 20 True
>>> not a == 12 and not b == 10 True
```

## Membership Operators:

Python Narayana

(233)

In addition to the operators discussed previously, Python has membership operators, which tests for membership in a sequence, such as strings, list, dict, tuples. There are two membership operators,

Operator	Description	Example
In	Evaluates to True if it finds a variable in the specified sequence and False otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to True if it does not find a variable in the specified sequence and False otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Eg:-

```
Lst = [1, 2, 3, 4, 'python', True]
>>> 4 in Lst True
>>> 10 in Lst False
>>> 3 not in Lst False
>>> 20 not in Lst True
```

## Identity Operators

Python Narayana

(234)

Identity operators compare the memory locations (of) references of two objects there are two identity operators explained below.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object & false otherwise.	x is y here <u>is</u> results in 1 if id(x) equals to id(y).
isnot	Evaluates to false if the variables on either side of the operator point to the same object & true otherwise	x is not y, here is not results in 1 if id(x) is not equal to id(y)

Eg:-

```
>>> st = "Sai"
```

```
>>> id(st)
```

57264032

```
>>> st1 = "Sai"
```

```
>>> id(st1)
```

57264032

>> st is st1                          True.  
 >> st is not st1                      false.

Eg:2:-

>> lst = [1, 2, 3, 4].  
 >> id(lst)                            57297136.

>> lst1 = [1, 2, 3, 4]  
 >> id(lst)                            57240572

>> lst is lst1                        false  
 >> lst is not lst1                    True.

## Operators Precedence

The following table lists all operators from highest precedence to lowest

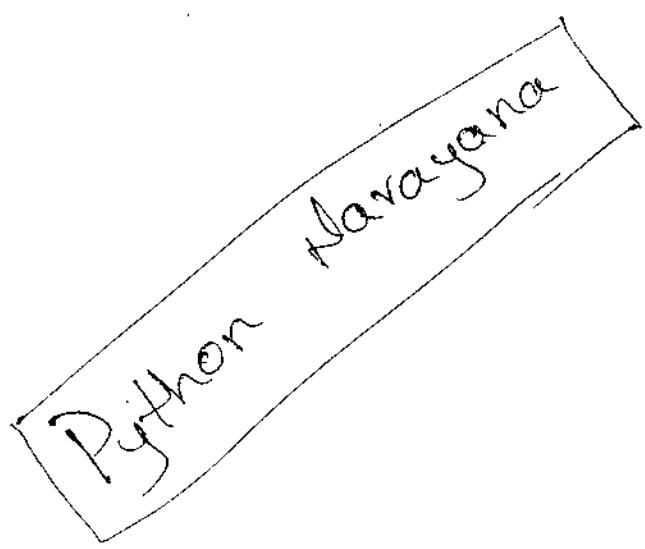
Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus, and unary minus (method names for the last two are <code>+@</code> and <code>-@</code> )
* / % //	Multiply, divide, modulo, & floor division.

$+$ -	Addition and subtraction
$>> <<$	Right and Left bitwise shift
$\&$	Bitwise 'AND'.
$\text{A} \text{I}$	Bitwise exclusive 'OR' and regular OR
$<= <> >=$	Comparison operators.
$<> == !=$	Equality operators
$= \% = / = // =$	Assignment operators
$- = + = * = ** =$	
$\text{is}, \text{is not}$	Identity operators
$\text{in}, \text{not in}$	Membership operators.
$\text{not}, \text{or}, \text{and}$	Logical operators.

### Exercise on operators:-

1. True and True.
2. False and True.
3.  $1 == 1$  and  $2 == 1$
4. "test" == "test"
5.  $5 == 1$  or  $25 == 1$
6. True and  $1 == 1$
7. False and  $0 == 0$
8. True or  $1 == 1$
9. "test" == "Testing".

10.  $1 \& 0 \text{ and } 2 == 1$
11.  $\text{"test"} \&= \text{"testing"}$
12.  $\text{"test"} == 1.$
13.  $\text{not} (\text{True} \text{ and } \text{False})$ .
14.  $\text{not} (1 == 1 \text{ and } 0 \& 1 == 1)$
15.  $\text{not} (10 == 1 \text{ or } 1000 == 1000)$
16.  $\text{not} (11 == 10 \text{ or } 3 == 4)$
17.  $\text{not} (\text{"testing"} == \text{"testing"} \text{ and } \text{"Zed"} == \text{"Cool Guy"})$
18.  $1 == 1 \text{ and } \text{not} (\text{"testing"} == 1 \text{ or } 1 == 0)$
19.  $\text{"Chunky"} == \text{"bacon"} \text{ and } \text{not} (3 == 4 \text{ or } 3 == 3)$
20.  $3 == 3 \text{ and } \text{not} (\text{"testing"} == \text{"Testing"} \text{ or } \text{"Python"} == \text{"Fun"})$



# Some Examples on Bitwise Operators - Python Narayana

(258)

1. Bitwise AND
2. Bitwise OR
3. Bitwise XOR
4. Bitwise leftshift
5. Bitwise right shift.

## Bitwise AND

① 3 & 2

$$\begin{array}{r} 0011 \\ \& 0010 \\ \hline 0010 \end{array} \rightarrow 2$$

② 6 & 9

$$\begin{array}{r} 0110 \\ \& 1001 \\ \hline 0000 \end{array} \rightarrow 0$$

## Bitwise OR

① 4 | 5

$$\begin{array}{r} 0100 \\ | 0101 \\ \hline 0101 \end{array} \rightarrow 5$$

② 4 | 7

$$\begin{array}{r} 0100 \\ | 0111 \\ \hline 0111 \end{array}$$

③ 9 | 15

$$\begin{array}{r} 0100 \\ | 1001 \\ \hline 11001 \end{array}$$

(25)

## Bitwise XOR

4 ^ 5

$$\begin{array}{r} 0100 \\ | 0101 \\ \hline 0001 \end{array} \quad ①$$

2) 15 ^ 25

$$\begin{array}{r} 01111 \\ | 11001 \\ \hline 10110 \end{array} \quad 22$$

Ques 8

python Narayana

(239)

$$\begin{array}{r} 623 \\ \times 10 \\ \hline 0011 \\ \hline 0010 \end{array}$$

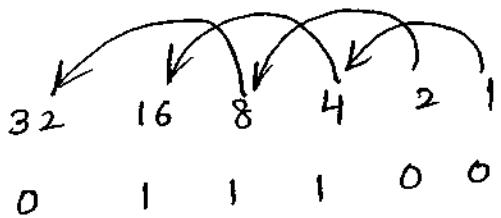
$$\begin{array}{r} 218 \\ \times 10 \\ \hline 1010 \rightarrow 10 \end{array}$$

Bitwise left shift :-

$$7 \ll 2$$

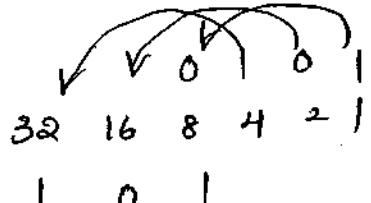
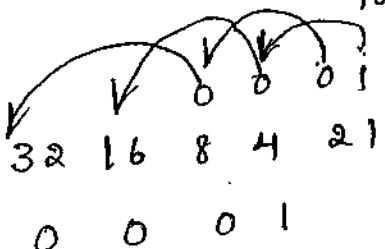
$$7 \rightarrow 0111$$

$$2 \rightarrow 0010$$



$$16+8+4=28 \quad 5 \ll 3$$

$$1 \ll 2$$



$$32+8=40$$

Bitwise right shift :-

$$8 \gg 2$$

$$16 \ 8 \ 4 \ 2 \ 1 = 2$$

$$16 \gg 2$$

$$16 \ 8 \ 4 \ 2 \ 1 = 4$$

$$10 \gg 2$$

$$16 \ 8 \ 4 \ 2 \ 1 = 2$$

$$20 \gg 3$$

$$16 \ 8 \ 4 \ 2 \ 1 = 2$$

$$17 \gg 3$$

$$2$$

$$18 \gg 2$$

$$= 4$$

## Some examples on bitwise operators - Python Narayana

(20)

>>> a = 20	
>>> b = 25	
>>> c = 15	
>>> a & b	16
>>> a & b & c	0
>>> a   b & c	29
>>> a & b   c	31
>>> a & b & c	2
>>> a & b << 2	4
>>> a & b   c << 2	60.
>>> a   b   c << 2	61
>>> a & b   c >> 2	19
>>> a   b   c >> 2	31

## Some examples on Logical operators

>>> a = 100	
>>> b = 150	
>>> c = 200	
>>> a == b and a < b	false
>>> a != c and b < c	true.
>>> a == b and b == c and a < b	false
>>> a != b and b > c or a <= b	false.
>>> not a == b and a < b and a < c	true.
>>> not a < b or a == c	false
>>> not not a < b or a == c	true.

## \* Conditional Statements \* Narayana

(241)

Conditional statements will decide the execution of a block of code based on the expression.

The conditional statements return True (or) False.

Python supports four types of conditional statements.

1. Simple If or If statement
2. If-Else statement
3. If else If (elif) statement
4. Nested If statement
5. If Statement

The python If statement is same as it is with other programming languages. It executes a set of statements conditionally, based on the value of a logical expression.

Here is the general form of one way If statement

Syntax:-

```
if expression:
 Statement-1
 Statement-2
 ...
.
```

In the above syntax, expression specifies the condition. It produces either true or false. If the expression evaluates true then the same amount of indented statement(s) following it will be executed. This group of the statement(s) is called a "block".

Eg:-1

```
marks = int(input("Enter your percentage of marks to know pass or failed"))
```

```
if marks >= 35:
 print("you are passed...")
```

Output

```
Enter your percentage of marks to know pass or failed:45
you are passed ---.
```

Eg:-2 compare values and display message

```
a = int(input("Enter First value:"))
```

```
b = int(input("Enter Second value:"))
```

```
c = int(input("Enter third Value:"))
```

```
if a > b < c:
```

```
 print("a is greater than b and also c")
```

(Q)

python Narayana

To test max value in two values Python Narayana

max = a if (a>b) else b;

(243)

or

if a>b &

x=a

else:

x=b.

Eg:3 If name is more than or one character then do all string methods.

name = input ("Enter name:")

if len(name) > 0:

print "the length of name is:", len(name)

print "the given name is:", name

print "the Capitalization of name is:", name.capitalize()

print "The title of the name is:", name.title()

print "The lower case form of given String:", name.lower()

print "The uppercase form of given String:", name.upper()

print "The reverse of given strings:", ".join(reversed(name))

print "The asc order of given string is:", ".join(sorted(name)).

print "The desc order of given string is:", ".join(reversed(sorted(name)))

Python Narayana

Eg:4

people = 20

cats = 30

dogs = 15

if people &lt; cats:

print ("Too many cats! The world is doomed!")

if people &gt; cats:

print ("Not many cats! The world is saved!")

if people &lt; dogs:

print ("The world is drooled on!").

if people &gt; dogs:

print ("The world is dry!")

dogs += 5

if people &gt;= dogs:

print ("people are greater than or equal to dogs")

if people &lt;= dogs:

print ("people are less than or equal to dogs")

if people == dogs:

print ("people are dogs.").

O/P:-Too many cats! The world is doomed!  
The world is dry!

Python Narayana

people are greater than or equal to dogs.

People are less than or equal to dogs. else

people are dogs.

## If-ELSE Statement

In python if-else statements has two blocks, first block follows the expression and other block follows the else clause. Here is the syntax.

### Syntax:

```
if expression:
 statement_1
 statement_2
 - - -
 - - -
else:
 statement_3
 statement_4
 - - -
```

In the above case, if the expression evaluates to true then the same amount of indented statements follow the expression and if the expression evaluates to false the same amount of indented statements follow else block.

Eg:1 write python script to check whether the given number is even or odd number.

```
a = int(input('Enter your number:'))
```

```
if a%2 == 0:
```

```
 print(a, "is a even number")
```

```
else
```

```
 print(a, "is a odd number").
```

### Output

```
Enter your number: 20
```

```
20 is a even number.
```

```
(8)
```

```
Enter your number: 21
```

```
21 is a odd number.
```

Eg:2 write a python script to know passed or failed.

```
marks = int(input("Enter your percentage of marks to know pass or failed")).
```

```
if marks >= 35:
```

```
 print("you are passed")
```

```
else:
```

```
 print("you are failed").
```

Output:-

Enter your percentage of marks to know pass or failed : 20  
you are failed.

Enter your percentage of marks to know pass or failed : 90  
you are passed.

Ex:3 Check whether he entered proper name (or) not,  
if it is more than or equal to one character then do  
all string operations else display "please enter valid name".

```
name = input ("Enter name:")
```

```
if len(name)<0:
```

```
 print ("please enter a valid name")
```

```
else:
```

```
 print ("The length of name is:", len(name))
```

```
 print ("The given name is:", name)
```

```
 print ("The Capitalization of name is:", name.capitalize())
```

```
 print ("The title of the name is:", name.title())
```

```
 print ("The Lowercase form of given string is:", name.lower())
```

```
 print ("The Uppercase form of given string is:", name.upper())
```

```
 print ("The reverse of given string is:", "join(reversed(name)))
```

```
 print ("The asc order of given string is:", "join(sorted(name)))
```

Python Narayana

```
print("the desc order of given string is:", "join(reversed
 (sorted(name))))).
```

Output:-

Enter name: python dev.

the length of name is: 10

the given name is: python dev.

the capitalization of name is: Python dev.

the title of the name is: Python Dev

the lower case form of given string: python dev.

the uppercase form of given string: PYTHON DEV.

the reverse of given string: ved nohtyp.

the asc order of given string is: edhnoptry.

the desc order of given string is: yvtponhed.

Output:-

Enter name:

please enter a valid name.

Eg:4 enter two values and find out bigger value

```
num1 = int(input("Enter first Number:")).
```

```
num2 = int(input("Enter second Number:"))
```

if num1 > num2:

```
 print(num1, "greater than", num2)
```

```
 print
```

python Navayana

240

```
O
O else:
O print (num1, " is Smaller than ", num2).
```

Output:

Enter first Number : 10.

Enter second Number : 5

10 is greater than 5

Enter first Number : 5

Enter second Number : 7

5 is smaller than 7.



elif statement

- It will check the condition 1 first, if the condition 1 is true then it will execute the block of statements which are following the condition, if the condition 1 is false then it will check the condition 2.
- If the condition 2 is true then it will execute the block of statements which follow the condition 2, if the condition 2 is false then it will execute the condition 3, like this it will check all conditions. If all conditions are false then it will execute the else block.

Syntax:

```

if condition 1:
 Statement-1
 Statement-2.

elif Condition 2:
 Statement-3
 Statement-4

elif Condition 3:
 Statement-5
 Statement-6.

else:
 Statement-7

```

## Ex-1 Marks example

Python Narayana

251

```
marks = int(input("Enter your percentage of marks:"))
```

```
If marks >= 0 and marks < 35:
```

```
 print("you are failed")
```

```
elif marks >= 35 and marks < 50:
```

```
 print("you got 3rd class")
```

```
elif marks >= 50 and marks < 60:
```

```
 print("you got 2nd class")
```

```
elif marks >= 60 and marks < 75:
```

```
 print("you got 1st class")
```

```
elif marks >= 75 and marks <= 100:
```

```
 print("you got distinction")
```

```
else:
```

```
 print("invalid marks")
```

## Output:

Enter your percentage of marks : 50

you got 2nd class.

Enter your percentage of marks : 90

you got distinction.

Enter your percentage of marks : 20

you are failed.

Eg2: food timings example

```

time = int(input("Enter your time:"))
if time > 7 and time < 10:
 print("It's time to have Breakfast...")
elif time >= 10 and time < 12:
 print("It's time to have Brunch...")
elif time >= 12 and time < 15:
 print("It's time to have Lunch...")
elif time >= 15 and time < 18:
 print("It's time to have Snacks")
elif time >= 18 and time < 20:
 print("It's time to have Dinner")
elif time >= 20 and time <= 24:
 print("It's sleeping time")
elif time >= 1 and time <= 7:
 print("It's sleeping time")
else:
 print('you entered invalid time')

```

Output:

Enter your time: 8  
It's time to have breakfast

Enter your time: 13

Its time to have lunch.

Enter your time: 2

Your entered invalid time

Enter your time: 3.

Its sleeping time.

Eg3: Find bigger value of two given values.

```
a = int(input("Enter first value:"))
b = int(input("Enter Second value:"))

if a > b:
 print(a, 'is greater than', b)
elif b > a:
 print(b, "is greater than", a)
else:
 print(a, 'and', b, 'are same values').
```

Output:

Enter first value: 10

Enter Second value: 15

15 is greater than 10.

Output:-2

Enter first value : 10

Enter Second value : 10

10 and 10 are same values.

Output:-3

Enter first value : 15

Enter Second value : 10

15 is greater than 10.

Ex:-4 find biggest value of three given values.

```
a = int(input("Enter first value:"))
```

```
b = int(input("Enter second value:"))
```

```
c = int(input("Enter third value:"))
```

if  $a > b$  and  $a > c$ :

    print(a, 'is greater than', 'b', 'and', c)

elif  $b > c$ :

    print(b, 'is greater than', 'a', 'and', c)

else:

    print(c, 'is greater than', 'a', 'and', b)

Output:-4

Enter first value : 10

Enter Second value : 20

Enter Third value : 15.

20 is greater than 10 and 15.

Output: 2 :

Enter first value: 10

Enter Second value: 5

Enter third value: 30.

30 is greater than 10 and 5.

Output: 3:

Enter first value: 10

Enter Second value: 3

Enter third value: 5

10 is greater than 3 and 5.

Output: 4

Enter first value: 10

Enter second value: 10

Enter third value: 10.

three are same values.

Eg: 5:

people = 30

Cars = 40

buses = 15

if cars > people :

print("We should take the cars").

Python  
Narayana

Python Narayana  
256

```

if cars < people:
 print("we should not take the cars").
else:
 print("we can't decide").
if buses > cars:
 print("that's too many buses").
elif buses < cars:
 print("Maybe we could take the buses")
else:
 print("we still can't decide")
if people == buses:
 print("Alright, let's just take the buses")
else:
 print("Fine, let's stay home then").

```

### Output:-

we should take the cars.  
 Maybe we could take the buses.  
 Alright, let's just take the buses.

### Eg:6:-

```

Print("you enter a dark room with two doors. Do
you go through door #1 or door #2?")
door = input(">")

```

## Python Narrator

(257)

if door == "1":

print ("There's a giant bear here eating a cheese  
cake. What do you do?

print ("1. Take the cake")

print ("2. Scream at the bear").

bear = input (">")

if bear == "1":

print ("The bear eats your face off. Good job!")

elif bear == "2":

print ("The bear eats your legs off. Good job!")

else:

print ("Well, doing %s is probably better. Bear  
runs away." % bear)

elif door == "2":

print ("You stare into the endless abyss at  
Cthulhu's retina.")

print ("1. Blueberries")

print ("2. Yellow jacket clothespins")

print ("3. Understanding revolvers yelling melodies")

insanity = input (">")

if insanity == "1" or insanity == "2":

print ("Your body survives powered by  
a mind of jello. Good job!")

```
else:
 print ("the intensity rots your eyes into a pool
 of muck. good job!")

else:
 print ("you stumble around and fall on a knife
 and die. good job!")
```

Output:-

you enter a dark room with two doors. Do you go  
through door #1 or door #2?

&gt;1

there's a giant bear here eating a cheese cake. what do  
you do?

- 1. take the cake.
- 2. Scream at the bear.

&gt;1

The bear eats your face off. Good job!

Python Narayana

## Nested if statements

python Narayana

(259)

python supports using "if statements in another if statement".

If the upper if condition is true then the inner if will be evaluated, if the upper if is false then it will not check the inner if statement.

### Syntax:-

if condition 1:

    if condition 2:

        if condition 3:

            Statement 1

            Statement 2

-----

Elif condition 3:

    Statement 1

    Statement 2

-----

Else:

    Statement 1

    Statement 2

-----

Elif condition 2:

    Statement 1

    Statement 2

-----

Elif condition 2:

    Statement 1

    Statement 2

-----

Else:

Statement 1

Statement 2

$$\begin{array}{r} 17 \\ \times 3 \\ \hline 51 \end{array}$$

Elif Condition 1:

Statement 1

Statement 2

Else:

Statement 1

Statement 2.

Eg 81

Database contains only male records, if user enters about female then display 'female records are not available' and if user enters about male then check employees name, if male employee name existed then display his details, if that name is not available then display 'no body is there with that name'; if enters any wrong gender then display 'you entered wrong gender'. Finally display 'thank you' at the end of result.

male records displaying:-

```
gender = input("Enter gender: ")
```

```
name = input("Enter name")
```

```
if gender == "female" or gender == "Male":
```

```
 if gender == 'Female':
```

python Narayana

print("female records are not available")

(26)

else:

if name == 'satya' and gender == "Male":

    print("Satya is from hyd and working as SE")

elif name == 'Narayana' and gender == "Male":

    print("Narayana is from nagpur and working as ASE").

elif name == 'Nani' and gender == "Male":

    print("Nani is from hyd and working as JL")

else:

    print("you nobody is there with that name")

else:

    print("you entered wrong genders")

    print("thank you")

Output:-

Enter gender : Male

Enter name Satya.

Satya is from hyd and working as SE

thank you

(8)

Male records display example

python Narayana

(162)

```
gender = input("Enter your gender:")
name = input("Enter your name:")

if gender == "Male" or gender == "female":
 if gender != "female":
 if name == "Satya":
 print("Satya is from hyd and having 10 years
 exp")
 elif name == "Sai":
 print("Sai is from mumbai and having 20
 years exp")
 elif name == "Narayana":
 print("Narayana is having 6 years exp f
 from hyd")
 else:
 print("Sorry, your name is not available in
 database")
 else:
 print("Sorry female records are not available
 in the database---")
else:
 print("Sorry, you entered invalid gender")

print("Thank you - - -")
```

Output:-

Python Narayana.

(263)

Enter your gender : Male

Enter your name : Satya

Sorry, you entered Male its invalid gender.  
please check Once

thank you....

Enter your gender : Male

Enter your name : Sai

Sai is from mumbai and having 20 years exp.

thank you....

Enter your gender : Female

Enter your name : Renu

Sorry female records are not available in the database..

thank you....

Python Narayana

## Eg:- Checking the eligibility of interview Python Narayana

(261)

```
name = input('Enter your name: ')
qual = input('Enter your qualifications: ')
year = int(input('Enter passed out year: '))
per = eval(input('Enter your percentage: '))

qual = qual.lower()
name = name.capitalize()

if qual == 'batch' or qual == 'be':
 if year == 2016 or year == 2017:
 if per >= 0 and per < 35:
 print('Hello', name, 'don't come to interview')
 print('because you got', per, 'percentage only')
 elif per >= 35 and per < 50:
 print('Hello', name, 'please come to interview')
 print('with work exp')
 elif per >= 50 and per < 60:
 print('Hello', name, 'please come to interview')
 print('after two months')
 elif per >= 60 and per < 75:
 print('Hello', name, 'please come to interview')
 print('tomorrow').
```

elif per >= 75 and per <= 100:

print('Hello', 'name', 'please come to interview today  
because you got', 'per', 'percentage')

else:

print('Hello', 'name', 'you entered', 'per', 'Its invalid  
percentage')

if year < 2016:

print('Hello', 'name', 'you entered', 'year', 'So you are  
not fresher')

else:

print('Hello', 'name', 'you entered', 'year', 'Its invalid  
passedout year')

else:

print('Hello', 'name', 'you entered', 'qual', 'So you are not  
eligible')

print('thank you for your interest in our Company')

Output :-

Enter your name : Nani

Enter your qualification : Btech

Enter your passed out year : 2017.

Enter your percentage : 88.

Hello Nani, please come to interview today because  
you got 88 percentage.

thankyou for your interest in our company.  
Python Marayana  
266

### Output 2:

Enter your name: Nani

Enter your qualification & Degree.

Enter passed out year: 2016.

Enter your percentage: 75

Hello Nani, you entered degree. So you are not eligible.

thankyou for your interest in our Company.

### Output 3:

Enter your name: Krishna.

Enter your qualification & be.

Enter passed out year: 2018

Enter your percentage: 90.

Hello Krishna you entered 2018. Its invalid passed out year.

thankyou for your interest in our Company.

### Output 4:

Enter your name: Madhu

Enter your qualification & btech.

Enter passed out year: 2014.

Enter your percentage: 77.

Hello Madhu you entered 2014. So you are not fresher.

thankyou for your interest in our Company.

Output 5:-

Python Narayana.

(267)

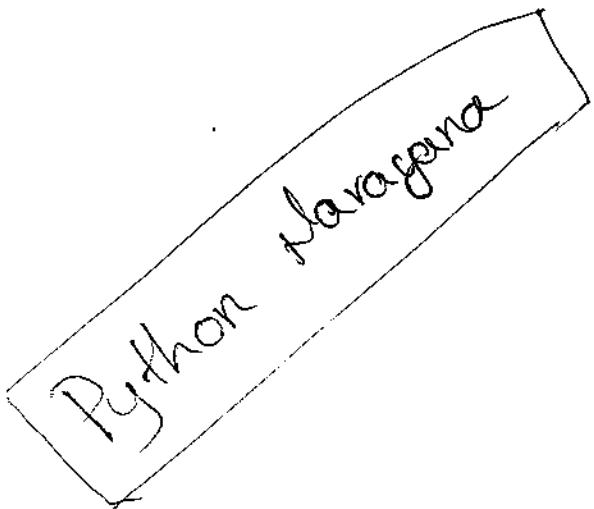
Enter your name : Venu.

Enter your qualifications : btech.

Enter passed out year : 2017.

Enter your percentage : 10.

Hello Venu, don't come to interview because you got  
10 percentage only.  
thankyou for your interest in our company.



# Iterative Statements

(268)

## for Loops

for loop allows a code block to be repeated a certain number of times.

### Eg:-1 for Loop on string

```
st = "python Developer"
```

```
for i in st:
 print(i)
```

### Output

P  
y  
t  
h  
o  
n  
  
D  
e  
v  
e  
l  
o  
p  
e  
r  
Y

Python Narayana

## Eg:2 for loop on list object

Python Narayana

269

```
Lst = [1, 2, 3.5, "python", 4+5j, True]
```

```
for i in Lst:
```

```
 print(i)
```

Output:-

1

2

3.5

python

(4+5j)

True.

## Eg:3 for loop on tuple object

```
tup = (1, 2, 3, True, False, "Narayana")
```

```
for i in tup:
```

```
 print(i)
```

Output:-

1

2

3

True.

False

Narayana.

Eg:4 for loop on set object

```
Se = {2, 3, 'python', 0, 0, True, 2+6j, 'Narayana'}
```

```
for i in se:
```

```
 print(i)
```

Output:-

0

python

2

3

True

Narayana

(2+6j)

Eg:5 for loop on dict object:-

```
dic = {1:'a', 2:'b', 'c':453}
```

```
for i in dic:
```

```
 print(i)
```

Output:-

1

2

c

Eg:6 displaying 10<sup>th</sup> table by using range function

```
for i in range(1, 11):
```

$ri = 10 * i$

```
print('10','*', i, '=', ri)
```

Output

10 \* 1 = 10

10 \* 2 = 20

10 \* 3 = 30

10 \* 4 = 40.

10 \* 5 = 50

10 \* 6 = 60

10 \* 7 = 70

10 \* 8 = 80.

10 \* 9 = 90.

10 \* 10 = 100.

for loop with else:-

A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.

break statement can be used to stop a for loop. In such case, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

Here is an example to illustrate this.

Eg:

MyList = ["Narayana", 100, "python", True].

for i in MyList:

print(i)

else:

# here else is optional, without else  
also possible.

print("Specified", len(MyList), "Items over")

(272)

### Output

Narayana.

100

python

True.

Specified 4 items over.

### While Loop:

In python programming language, A while loop statement repeatedly executes a target statement as long as a given condition is true.

while expressions

Statement 1

Statement 2

Statement 3

-----

Eg: Display first 5 numbers.

num = 1

while (num <= 5)

    print ("the count is : ", num)

    num += 1

    print ("Thank you")

### Output

the count is : 1

the count is : 2

the count is : 3

the count is : 4

the count is : 5

thank you.

Eg: Display squares of first 10 numbers.

MyNum = 11

myVar = 0

while myVar < myNum:

    print 'Square of ' + str(myVar) + ' is ' + str(myVar \*\* 2)

    myVar += 1

### Output:

Square of 0 is 0

Square of 1 is 1

Square of 2 is 4

Square of 3 is 9

Square of 4 is 16

Square of 5 is 25

Square of 6 is 36

Square of 7 is 49

Square of 8 is 64

Square of 9 is 81

Square of 10 is 100.

Python Reference

Python By Example

Eg: Display sum of all natural number of given number.

(274)

num = 5

Sum = 0

i = 1

while i &lt;= num:

Sum = Sum + i

i = i + 1

print ("the sum is", Sum)

Output: the sum is 15.

Eg: Display all even number to before given number.

Target\_Num = 10

Var = 1

while Var &lt; Target\_Num:

if Var % 2 == 0:

Var += 1

Continue.

print ('This number = ' + str(Var))

Var += 1

print ("Displayed all even number before", Var)

Python Narayana

Output

This number = 2

This number = 4

This number = 6

This number = 8

Displayed all even number before 10.

Eg: Checking username and password.

while True:

Name = input ("Enter Name")

If name != 'Narayana':

    Continue.

    print ("Hello", name, ", please enter your password")

    pwd = input ("Enter password")

    If pwd == 'DurgaSoft':

        print ("you entered correct details")

        print ("Congratulations", name)

        break

    Print ("Thankyou", name).

Output

Enter Name : Narayana.

Hello Narayana, Please enter your password -

Enter password : DurgaSoft

You entered correct details.

Congratulations Narayana  
thank you Narayana.

Python Narayana  
(276)

## while loop with else:

We can have an optional 'else' block with while loop as well.

The else part is executed if the condition in the while loop evaluates to false. The while loop can be terminated with a "break statement"

In such case, the else part is ignored. Hence a while loop's else part runs if no break occurs and the condition is false.

cnt = 0

while cnt < 3:

    print ("Now we are in inside loop")

    cnt = cnt + 1

else:

    print ("Now we are in else block")

## Output:-

Now we are in inside loop

Now we are in inside loop

Now we are in inside loop

Now we are in else block.

Eg:1

```

i=0
numbers = []
while i<6:
 print("At the top i is %d" % i)
 numbers.append(i)

 i=i+1
 print("Numbers now: ", numbers)
 print("At the bottom i is %d" % i)

print("The numbers:")
for num in numbers:
 print(num).

```

Output:

At the top i is 0

Number now : [0]

At the bottom i is 1

At the top i is 1

Number now : [0, 1].

At the bottom i is 2

At the top i is 2

Number now : [0, 1, 2]

At the bottom i is 3.

Python Narayana

# Python Narayana

(278)

At the top i is 3.

Numbers now : [0, 1, 2, 3].

At the bottom i is 4.

At the top i is 4

Numbers now : [0, 1, 2, 3, 4]

At the bottom i is 5

~~the numbers~~

At the top i is 5

Numbers now : [0, 1, 2, 3, 4, 5].

At the bottom i is 6

~~the numbers:~~

0

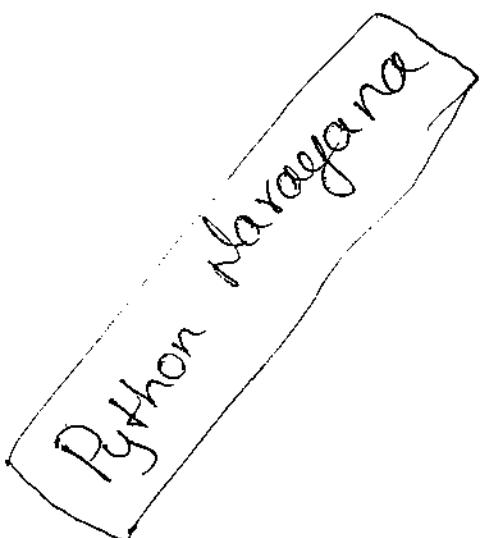
1

2

3

4

5



# Python Transfer Statements

(279)

## 1. break:

We can use break statement inside loops to break loop execution based on some condition.

Eg:-

1. for i in range(10):

2. If  $i == 7$ :

3. print("processing is enough... plz break")

4. break.

5. print(i)

6.

7. D:\python classes>py test.py..

8 0

9 1

10 2

11 3

12 4

13 5

14 6

15 processing is enough... plz break.

Eg:-

1. Cart = [10, 20, 600, 60, 70].

2. for item in Cart:

3. if item > 500:

4. print("To place this order insurance must be required")

(B80)

5. break.

6. print(item)

7.

8. D:\python-classes>py test.py.

9.. 10

10. 20

11 To place this order insurance must be required.

2. Continue: we can use continue statement to skip current iteration and continue next iteration.

Eg:- To print odd numbers in the range 0 to 9.

1. for i in range(10):

2. if i%2 == 0:

3. continue.

4. print(i)

5.

6. D:\python-classes>py test.py.

7 1  
8 3

9 5

10 7

11 9

Python Narayana

## Python Nested Loops.

(281)

- Eg: 2. 1. Cart = [10, 20, 500, 700, 50, 60].
2. for item in Cart:
3. if item >= 500:  
4. print("we cannot process this item:", item)
5. Continue
6. print(item)
- 7.
8. Output.
9. D8\Python-classes>py test.py
10. 10
11. 20
12. we cannot process this item: 500.
13. we cannot process this item: 700
14. 50
15. 60.

Eg: 3

1. numbers = [10, 20, 0, 5, 0, 30].

2. for n in numbers:

3. if n == 0:

4. print("Hey how we can divide with zero--  
just skipping")

5. Continue.

6. print("100/23 =?", format(1, 100/n))

- 7.

8. Output

- 9.

10. 100/10 = 10.0

11.  $100/20 = 5.0$ 

12. Hey how we can divide with zero... just skipping.

13.  $100/5 = 20.0$ 

14. Hey how we can divide with zero... just skipping.

15.  $100/30 = 3.333333333335$ loops with else block

Inside loop execution, if break statement not executed  
 then only else part will be executed.

else means loop without break.

Eg: 1. `cart = [10, 20, 30, 40, 50]`

2. `for item in cart:`

3. `if item >= 500:`

4.  `print ("we cannot process this order")`

5. `break`

6. `print(item)`

7. `else:`

8.  `print ("Congrats... all items processed successfully")`

9.

10. Output

11. 10

12. 20

13. 30

14. 40

15. 50

16. Congrats - all items processed successfully.

Expt 1. cart = [10, 20, 600, 30, 40, 50] python Narayan

(283)

```
2. for item in cart:
3. if item >= 500:
4. print ("We cannot process this order")
5. break
6. print(item)
7. else:
8. print ("Congrats... all items processed successfully")
9.
10. Output
11. D:\python-classes>py test.py
12. 10
13. 20
14. We cannot process this order.
```

Q. what is difference b/w for loop and while loop in Python?

We can use loops to repeat code execution.

Repeat code for every item in Sequence  $\Rightarrow$  for loop.

Repeat code as long as condition is true  $\Rightarrow$  while loop.

Q. How to exit from the loop?

by using break statement.

Q. How to skip some iterations inside loop?

by using continue statement.

Q. When else part will be executed wrt loops? Narayana  
If loop executed without break. 284

### 3. pass Statement?

pass is a keyword in python.

In our programming syntactically if block is required which won't do anything then we can define that empty block with pass keyword.

pass

1- It is an empty statement.

1- It is null statement.

1- It won't do anything

Eg:-

if True:

SyntaxError: unexpected EOF while parsing.

if True: pass

==> Valid.

def m():

SyntaxError: unexpected EOF while parsing.

def m(): pass.

Use Case of pass- Sometimes in the parent class we have to declare a function with empty body and child

class responsible to provide proper implementation.

Such type of empty body we can define by using

pass keyword.

python Narayana.

Eg:- def my(): pass

285

Eg:- 1. for i in range(100):

2. if i%9 == 0

3. print(i)

4. else: pass

5.

6. D:\python\_classes>py test.py

7. 0

8. 9

9. 18

10. 27

11. 36

12. 45

13. 54

14. 63

15. 72

16. 81

17. 90

18. 99,

Del Statement :- del is a keyword in Python.

After using a variable, it is highly recommended to delete that variable if it is no longer required, so that the corresponding object is eligible for Garbage Collection.

We can delete Variable by using del keyword.

Eg:- 1. x=10

2. print(x)

3. del x

After deleting a variable we cannot access that variable Python throws an  
Otherwise we will get NameError

(286)

- Eg:-
1.  $x = 10$ .
  2.  $\text{del } x$ .
  3.  $\text{print}(x)$

NameError: name 'x' is not defined.

Note: we can delete variables which are pointing to  
immutable objects. But we cannot delete the elements  
present inside immutable object.

- Eg:-
1.  $s = "durga"$
  2.  $\text{print}(s)$
  3.  $\text{del } s \Rightarrow$  valid.
  4.  $\text{del } s[0] \Rightarrow$  TypeError: 'str' object doesn't support  
item deletion.

Difference b/w del & None

In the case of del, the variable will be removed & we  
cannot access that variable.

1.  $s = "durga"$ .
2.  $\text{del } s$
3.  $\text{print}(s) \Rightarrow$  NameError: name 's' is not defined.

But in the case of None assignment the variable won't  
be removed but the corresponding object is eligible for  
Garbage Collection. Hence after assigning with None value, we can  
access that variable.

1.  $s = "durga"$
2.  $s = \text{None}$
3.  $\text{print}(s) \# \text{None}$

## python functions

python Narayana

187

functions are first class objects in python what it means. that they can be treated as just like any other variables and you can pass them as arguments to another function or return them as return statement.

- They are known in most programming languages,  
sometimes also called ~~Subroutines~~ or procedures.
- Functions are used to utilize code in more than one place in a program. The only way without functions to reuse code ~~consists~~ in copying the code.

A function in python is defined by a def statement. the general syntax look like this:

### Syntax

```
def function-name (parameter list): #function definition
 statements, in function body
```

```
function-name (actual parameters list) #function call
```

Python Narayana  
1. Write a python functions to add two numbers.

(288)

```
>>> def add(x, y):
```

```
 return x+y
```

```
>>> add(10, 20) 30.
```

2. write a python functions to find maximum value of two given values.

```
>>> def max(x, y):
```

```
 if x > y:
```

```
 return x
```

```
 else:
```

```
 return y.
```

```
>>> max(1, 2) 2
```

```
>>> max(10, 2) 10.
```

3. write a python function to find max value of three given values

```
>>> def max of three(x, y, z):
```

```
 if x > y and x > z:
```

```
 return x
```

```
 elif y > z:
```

```
 return y
```

```
 else
```

```
 return z
```

```
>>> max of three (100, 3, 20)
>>> max of three (10, 3, 20)
>>> max of three (10, 30, 20)
```

python Narayana

100

20

30-

(289)

## Default arguments:

Default parameters assume a default value if a value is not provided by the actual parameters in the function call.

```
def display_message (times, message):
 for i in range (times):
 print (message)
```

```
display_message (4, 'python Narayana')
```

## Output:-

python Narayana.

python Narayana.

python Narayana.

python Narayana.

So we can get some default values to the formal parameters in the function definition. Those are called default arguments. So that if we don't

Python Narayana

Python Narayana

Specify actual parameters in the function call  
then interpreter takes formal parameters values &  
Continue the operation.

```
def display_message(time=5, message = "This is python time"):
 for i in range(times):
 print(message)
display_message()
```

### Output:

This is python time.

This is python time.

This is python time

This is python time.

This is python time.

In the above function we didn't pass the actual parameters in the function call so interpreter has taken the default values and continued the operation.

If we pass the actual values when we have default values already in the function definition then interpreter takes actual values & continue the operation.

Eg:-

```
def display-message (time=5, message = "This is
python time"):
 for i in range (times):
 print (message)
display-message (2, 'python Narayana')
```

Output:-

python Narayana

python Narayana.

Generally the first actual parameter will map to the first formal parameters and Second actual parameters will map to the second formal parameters and so on....

If we give those mappings in the reverse way then it will throw error like,

```
def display-message (time=5, message = "This is python
time"):
 for i in range (times):
 print (message)
```

```
>>> display-message ('Narayana', 3).
```

Output:-

TypeError: 'str' object cannot be interpreted as an integer.

Keyword arguments:

- If we use any "variable name" or formal arguments in function call for the purpose of assigning actual arguments
- Then those are called "keyword arguments"
- ~~If we~~

Syntax: def fun-def (positional / formal args, keyword args):

function-call (actual arguments, keyword args)

```
def display (a, n, msg):
```

```
 for i in range(n):
```

```
 print (msg)
```

```
display (10, msg='Py', n=3)
```

Non key  
word args.

key word arguments

\* If we don't use any variable name or formal arguments names in the function call then there are called "Non-keyword arguments"

```
def display_message(times=5, message="This is " +
 "Python time!"): ②23
 for i in range(times):
 print(message)
display_message(message='python Narayana', times=2)
```

Output:

python Narayana.

python Narayana.

### Scope of the Variables

All variables in a program may not be accessible at all locations in that program, they are accessible depends on where you have declared the variables.

→ If the "lengths" of both actual args & formal args is "Same" and function call has all keyword args then it is known as "fixed length keyword arguments"

```
def display(n,msg):
```

```
 for i in range(n):
```

```
 print(msg)
```

```
display(msg='py', n=3).
```

Python  
Narayana

Python Narayana  
294

→ If the lengths of both formal & actual args is same and any variable (or) formal argument is not used in the function call then those are called "fixed length non-keyword arguments"

~~eg:~~ def display (n, msg):  
 for i in range(n):  
 print (msg)  
display (2, 'py')

\*args and \*\* keyword args:

\* args - Variable length non-keyword arguments.  
\*\* kwargs → Variable  
\* args is used to handle the variable lengths args  
and with non-keyword args in the funcall.

Syntax:

def fun-name (non-default args, default args, variable length  
non keyword args):

def fun-name (non-default args, default args, \*args):

funcall (actual args)

~~etc~~

Python Narayana

```

def display (b, a=10, *c):
 print (b, a, c)

```

variable length  
non-keyword)

non-keyword

display (3, 4, 5, 6) actual args.

O/p: 3 4 (5, 6).

Note: the output format of variable-length nonkeyword args is Tuple.

```

def display (a, b=10, *c):
 print (a, b, c)

```

display (3, 2, 6)

O/p: 3 2 (6,)

```

def display (a, b=10, *c):
 print (a, b, c)

```

display (3, 2)

O/p: 3 2 ()

```

def display (a, b=10, *c):
 print (a, b, c)

```

display (3)

O/p: 3, 10 ()

Python Narayana

## Python Narayana

→ In the above example, the default argument:

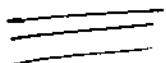
'b' is not required a formal argument from the function call because it is assigned with '10' as a default value.

(29)

→ The variable length non-keyword args '\*c' is also not required value because a variable length means '0' or more.

\*\*kwargs: It is used to handle variable length keyword args.

def fun-name (non-default args, default args, Variable length keywords (or) \*\*args)



fun-call (actual args)

def display (a, b=10, \*\*c):

    print (a, b, c)

display (1, 5, x=10; y=20; z = 'Santhoshi')

Output: 1 5 8 10, 20, z: 'Santhoshi'

```
def display (a, b=10, **c, **d):
 print (a, b, c, d)
```

(27)

```
display (1, 5, y=10, z=20, m='py', n='dex')
```

Op: 1 5 (10, 20, 30) { m:'py', n:'dex' }.

```
def display (a, b=10, *c, **d):
```

```
 print (a, b, c, d)
```

```
display (1)
```

Output :-

```
1 10 () 23
```

```
display (2, 5)
```

```
2 5 () 23
```

```
display (2, 3, 10)
```

```
2 3 (10,) 23.
```

Python Narayana

Syntax of both `*args` and `**args`:

```
def fun-def (non-default args, default args, *args, **kwargs)
```

```
=====
```

```
fun-call (actual args)
```

→ the values are optional for default args, `*args`,  
`**kwargs`.

Python Narayana

def display (a, b = 10\*c, \*\*d):  
 print (a, b, c, d)

(29B)

display ()

O/p 1 10 () 23.

### Scope of the Variables:

All Variables in a program may not be accessible at all locations in that program, they are accessible depends on where you have declared the variables.

python supports two types of variables.

1. Local Variables.

2. Global Variables.

### Local Variables:

Local variables are variables which are declared outside of a function.

Let's an example with local and global variables scopes,

Here Loc-var is a "local variable" and "glo-var" is a "global variable"

glo-var = 0.

def cal\_product\_of\_two\_values (num1, num2):

Loc-var = num1 \* num2.

Python Narayana  
209

```
print("the result of LOCAL variable is:", loc-var)
return loc-var.
```

```
print("the result of GLOBAL variable before calling
 the function is:", glo-var)
```

```
glo-var = cal-product-of-two-values(3, 5)
```

```
print("the result of GLOBAL variable after calling
 the function is:", glo-var)
```

### Output:-

```
the result of GLOBAL Variable before calling the function is: 0
```

```
the result of LOCAL variable is: 15
```

```
the result of GLOBAL variable after calling the
 function is : 15.
```

We can also use global variable in the functions like

```
glo-var = 0
```

```
def cal-product-of-two-values(num1, num2):
```

```
 Loc-Var = num1 * num2
```

```
 print("the result of LOCAL variable is:" Loc-Var)
```

```
 print("the result of GLOBAL Variable in the function
 is :" glo-var).
```

```
return Loc-Var.
```

```
print("the result of GLOBAL variable before Calling
 the function is :" glo-var).
```

glo-var = cal-product-of-two-values(3,5) Python Noragami.  
print("The val result of GLOBAL Variable after calling  
the function is:", glo-var) (300)

### Output

The result of GLOBAL variable before calling the function  
is: 0

The result of LOCAL variable is: 15

The result of GLOBAL Variable in the function is: 0

The result of GLOBAL variable after calling the function 15\*15

We can't use Local variable outside the function where  
it is declared, if we use the interpreter will throw error.

glo-var = 0

def cal-product-of-two-values(num1, num2):

Loc-var = num1 \* num2

print("The result of LOCAL Variables is:", Loc-var)

return Loc-var.

print("The result of GLOBAL Variable before calling  
the function is:", glo-var).

glo-var = cal-product-of-two-values(3,5)

```
print ("the result of GLOBAL Variable after calling
the function is:", glo-var)
```

(301)

```
Print ("the result of LOCAL variable after function
call is : " Loc-var).
```

Output:

The result of GLOBAL variable before calling the  
function is : 0

the result of LOCAL Variable is : 15

the result of GLOBAL Variable after calling the function is : 15

NameError: name 'Loc-var' not defined.

Arguments packing & Unpacking

When we need to call a function definition with function  
call which is having a List with size 3. So if we  
pass simply then it will not work. Let's check once.

```
def pack-var (arg1, arg2, arg3):
```

```
 print(arg1, arg2, arg3)
```

```
my-list = [100, 200, 300] # Creating list with size 3
elements
```

```
pack-var(my-list) # calling a func with my list
```

Output:-

`TypeError: pack-Var() missing 2 required positional  
arguments: 'arg2' and 'arg3'.` 302

In this case we should use unpack the my-list.  
This is called "arguments unpacking"

```
def packVar (arg1, arg2, arg3):
```

```
 print (arg1, arg2, arg3)
```

```
my-list = [100, 200, 300].
```

```
pack-Var (*my-list)
```

Output:- 100 200 300

We can also pack the arguments like,

when we don't know how many arguments need to be passed to a python function, we can pack all arguments in a tuple. This is called "argument packing"

```
def sum-of-args (*args):
```

```
 sum=0
```

```
 for i in range (0, len(args)):
```

```
 sum= sum+ args[i].
```

```
 return sum
```

```
print (sum-of-args(2, 20, 30))
```

(9)

Python Narayana

print (sum-of-args (20, 20, 30))

303

OpT = 70.

Eg:-

```
def cheese_and_crackers(cheese_count, boxes_of_crackers):
 print ("you have %d cheeses!" % cheese_count)
 print ("you have %d boxes of crackers!" % boxes_of_crackers)
 print ("Man that's enough for a party!")
 print ("Get a blanket\n")
```

print ("we can just give the function number directly: ")

cheese\_and\_crackers (20, 30)

Output:

We can just give the function numbers directly:

you have 20 cheeses!

you have 30 boxes of crackers!

Man that's enough for a party.

Get a blanket.

Python Narayana

# Python Programs

python Narayana

1.

Write a python function to check the given number is even or odd.

(30H)

```
num = int(input("Enter any value:"))
```

```
def even & odd(r):
```

```
 if n%2 == 0:
```

```
 print(num, 'is even number')
```

```
 else:
```

```
 print(num, 'is odd number')
```

```
even & odd(num)
```

Output:

Enter any value: 12

12 is even number.

Enter any value: 13

13 is odd number.

2. Write a python function to check the given number is positive or negative.

```
num = int(input("Enter any value:"))
```

```
def posorneg or zero (n):
```

```
 if n>0:
```

```
 print('The given num, is positive number')
```

```
 elif n==0:
```

```
 print('The given number is', num)
```

python Narayana.

else :

    print (num, 'is negative number')

(305)

pos or neg or zero (num)

Output:-

Enter any value : 8

8 is positive number

Enter any value : -9

-9 is a negative number

Enter any value : 0

0 is the given number is 0.

- Q 3. Write a python function to check given number is divisible by 10 (or) not?

```
num = int(input ("Enter any number:"))
```

```
def div by 10(n):
```

```
 if n%10 == 0:
```

```
 print (num, 'is divisible by 10')
```

```
 else:
```

```
 print (num, 'is not divisible by 10')
```

```
div by 10(num)
```

Output:-

Enter any number : 12

12 is not divisible by 10

Enter any number : 50  
50 is divisible by 10.

python Narayana

(306)

4. Write a python function to print first n even numbers.

```
num = int(input("Enter any number:"))
```

```
lst = []
```

```
def nevennums(n):
 for i in range(10000):
 if i%2==0:
 lst.append(i)
 if len(lst)==n:
 break
 print(lst)
nevennums(num).
```

Output:

Enter any number : 10

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Enter any number : 5

[0, 2, 4, 6, 8].

5. Write a python function to print even numbers upto n:

```
num = int(input("Enter any number:"))
```

```
def firstnnevennums(n):
```

```
 for i in range(n+1):
 if i%2==0:
```

Narayana  
Python

```
print(i)
```

```
firstnevennumbs (num)
```

Output:-

Enter any number : 10

0  
2  
4  
6  
8  
10

Enter any number : 11

0  
2  
4  
6  
8  
10

6. Write a python function to print sum of digits of number n.

```
num = int(input('Enter any number :'))
```

```
def Sum of digits (n):
```

```
S=0
```

```
for i in range (n+1):
```

```
S= S+i
```

```
print (S)
```

```
Sum of digits (num)
```

Python Narayan

## Output:

Enter any number: 15

120

Enter any number: 4

10

# write a python function to print the largest number of three given values.

```
num1 = int(input('Enter first number:'))
```

```
num2 = int(input('Enter Second number:'))
```

```
num3 = int(input('Enter third number:'))
```

```
def largest_number(a, b, c):
```

```
 if a > b and a > c:
```

```
 print(a, 'is the largest number of', b, 'and', c)
```

```
 elif b > c:
```

```
 print(b, 'is the largest number of', a, 'and', c)
```

```
 else:
```

```
 print(c, 'is the largest number of', a, 'and', b)
```

```
Largest_number(num1, num2, num3)
```

## Output:

Enter first number: 10

Enter second number: 11

Enter third number: 14

14 is the largest number of 10 and 11

Enter first number: 6

Enter Second number: 3

Enter third number: 2

6 is the largest number of 3 and 2.

8. write a python function to reverse the number or string.

```
st = input('Enter any string or number: ')
def reverseString(s):
 str1 = s[::-1]
 print(str1)
 reverseString(st)
```

Output:

Enter any string: python Narayana

anayaran nohtyp.

Enter any string: 1234

4321.

(Q)

```
st = input('Enter any string or number: ')
```

def reverseString(s):

```
str1 = ''.join(reversed(s))
```

print(str1)

reverseString(st)

Python Narayana

Output:

python Nareyana

Enter any string or number : Django.

(310)

OganjD

Enter any string or number : 7382

2837

#9 Write a python function to print the factors of given number

```
num = int(input('Enter any value:'))
```

```
def factors(n):
 for i in range(1, n+1):
 if n % i == 0:
 print(i)
```

```
factors(num)
```

Output:

Enter any value : 10.

1

2

5

10

Enter any value : 20

1

2

4

5

10

20

Python Nareyana

10. Write a python function to print factorial of given number.

(31)

```
num = int(input('Enter any number'))
```

```
def factorial(n):
```

```
 s=1
```

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

```
 s = s*i
```

```
 print(s)
```

```
factorial(num)
```

O/P: Enter any number 5

120

Enter any number 3

6.

11. Write a python fn to swap the given two numbers.

```
num1 = int(input("Enter first number"))
```

```
num2 = int(input("Enter second number"))
```

```
def swap(x,y):
```

```
 temp = x
```

```
 x = y
```

```
 y = temp
```

```
 print('first number is', num1, ', after swapping', x)
```

```
 print('second number is', num2, ', after swapping', y)
```

```
swap(num1, num2)
```

Python Narayana

python Naayana

312

O/p:-  
Enter first number : 13

Enter second number : 12

first number is 13, after Swapping 12

Second number is 12, after Swapping 13.

12 Write a python function to check the number is prime or not.

```
num = int(input("Enter any number:"))
```

```
def prime ¬ (n):
```

```
 for i in range (2, n):
```

```
 if n % i == 0:
```

```
 print (num, 'is not a prime')
```

```
 break
```

```
 else:
```

```
 print (num, 'is prime')
```

```
prime ¬ (8num)
```

O/p

Enter any number : 12

12 is not a prime.

Enter any number : 7

7 is prime.

Python Naayana

13. Write a python function to check the given no  
is palindrome or not. Python Narayana (313)

```
n = int(input("Enter any Number"))
def palindrome or not (num):
 if num == str(num)[::-1]:
 print(num, 'palindrome number')
 else:
 print(num, 'not a palindrome number')
palindrome or not (n)
```

Op's

Enter any number 12321

12321 palindrome number

Enter any number: 123432

123432 not a palindrome number

14. Check given number is armstrong or not

```
num = int(input('Enter any three digit number:'))
```

```
def armstrong or not (n):
```

S=0

while n>0:

digit = n%10

S+= digit\*\*3

n // 10

If num == 8:

    print(f'{num} is a armstrong number') (314)

else:

    print(f'{num} is not a armstrong number')

armstrong or not (num).

### Output

Enter any three digit number? 151

151 is not a armstrong number.

Enter any three digit numbers 153.

153 is a armstrong number.

15 Print fibonacci Series for below n number

```
num = int(input("Enter any number:"))
```

```
def fibonacci(n):
```

t<sub>1</sub> = 0

t<sub>2</sub> = 1

for i in range(10000):

    temp = t<sub>1</sub> + t<sub>2</sub>

    t<sub>1</sub> = t<sub>2</sub>

    t<sub>2</sub> = temp

    if t<sub>1</sub> >= n:

        break

    print(t<sub>1</sub>)

fibonacci(num)

Output:-

Python Narrator

315

Enter any number: 100

1

1

2

3

5

8

13

21

34

55

89

Enter any number ≥ 10

0

1

1

2

3

5

8

- 16 Write a python function to print first 10 Fibonacci Series numbers.

```
num = int(input('Enter any number:'))
```

```
def fibonacci(n):
```

```
 t1 = 0
```

```
 t2 = 1
```

```
 lst = [0]
```

```
 temp = 0
```

(316)

```

while temp < 10000:
 temp = t1 + t2
 t1 = t2
 t2 = temp
 lst.append(t1)
 if len(lst) == n:
 break
for i in lst:
 print(i)
fibonacci(10)

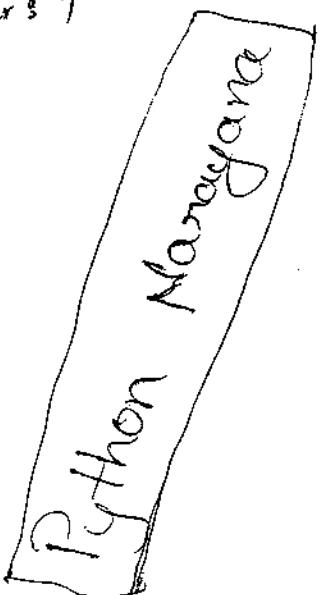
```

O/p: Enter any number: 5

0  
1  
1  
2  
3

Enter any number: 7

0  
1  
1  
2  
3  
5  
8



17 Write a python function to print the given three values in asc and desc order. python Note 317

```
a = eval(input("Enter first value:"))
b = eval(input("Enter second value:"))
c = eval(input("Enter third value:"))

def order_of_values(x, y, z):
 if x > y and x > z:
 if y > z:
 print('The desc order is', x, y, z)
 print('The asc order is', z, y, x)
 else:
 print('The desc order is', y, z, x)
 print('The asc order is', x, z, y)
 elif z > x and z > y:
 if x > y:
 print('The desc order is', z, x, y)
 print('The asc order is', y, x, z)
 else:
 print('The desc order is', z, y, x)
 print('The asc order is', x, y, z)
 elif y > x and y > z:
 if x > z:
 print('The desc order is', y, x, z)
 print('The asc order is', z, x, y)
 else:
 print('The desc order is', y, z, x)
 print('The asc order is', x, z, y)
```

else:

Python Narayana

    print('The desc order is', y, z, x)

(36)

    print('The asc order is', x, z, y)

elif z>x and z>y:

    if x>y:

        print('The desc order is', z, x, y)

        print('The asc order is', y, x, z)

    else:

        print('The desc order is', z, y, x)

        print('The asc order is', x, y, z)

Order of values a, b, c)

Output:

Enter first value: 12

Enter Second value: 11

Enter third value: 15

The desc order is 15 12 11

The asc order is 11 12 15

Enter first value: 3

Enter Second value: 2

Enter third value: 1

The desc order is 3 2 1

The asc order is 1 2 3.

Python  
Narayana

## Python Narayane

18. Write a python program to check the marital status, gender, age

(319)

```
marsta = input("Enter marital status (married & Single):").lower()
gen = input("Enter your gender (male or female):").lower()
age = int(input("Enter your age:"))
```

```
if marsta == "Married":
 print("you are not allowed to marry again")
elif marsta == "Single":
 if gen == "male":
 if age >= 21:
 print("congrats, you are eligible to marry")
 else:
 print("Sorry, you are not eligible to marry")
 elif gen == "female":
 if age >= 18:
 print("Congrats, you are eligible to marry")
 else:
 print("Sorry, you are not eligible to marry")
 else:
 print("you entered invalid gender")
else:
 print("you entered invalid marital status")
```

Output

Enter marital status (married or single) : married

Enter your gender (male or female) : male.

Enter your age : 24.

You are not allowed to marry again.

Enter marital status (married or single) : married

Enter your gender (male or female) : female.

Enter your age : 17

You are not allowed to marry again.

Enter marital status (married or single) : single

Enter your gender (male or female) : male

Enter your age : 22

Congrats, you are eligible to marry.

Q9. Write a python function to print  $n^{\text{th}}$  table.

```
num = int(input("Enter any number"))
```

```
def tabledisplay(n):
```

```
 for i in range(1, 11):
```

```
 print(n, "*", i, "=", n*i)
```

```
tabledisplay(num).
```

O/p:

Python Narayana

(321)

Enter any number 10

$$10 * 1 = 10$$

$$10 * 2 = 20$$

$$10 * 3 = 30$$

$$10 * 4 = 40$$

$$10 * 5 = 50$$

$$10 * 6 = 60$$

$$10 * 7 = 70$$

$$10 * 8 = 80$$

$$10 * 9 = 90$$

$$10 * 10 = 100$$

20. Write a python fn to remove vowels from given string.

```
st = input('Enter any string:')
```

```
v = 'aeiouAEIOU'
```

```
st = list(st)
```

```
for i in st:
```

```
 if i in v:
```

```
 st = st.replace(i, '')
```

```
st = ''.join(st)
```

```
print(st)
```

O/p:

Enter any string : python Narayana

pytn Ngn.

Enter any string : Django framework

Ding frmk.

Python  
Narayana

21 write a python function to copy a given string into new variable and count how many characters are copied.

(322)

```
st = input('Enter any string:')
```

```
def copystring(st):
```

```
c=0
```

```
str1 = ''
```

```
for i in st:
```

```
 str1 += i
```

```
c=c+1
```

```
print(str1)
```

```
print(c)
```

```
copystring(st)
```

O/p Enter any string: python Narayana.

python Narayana.

15

22 write a python function to count no. of digits in a given no.

```
num = input('Enter any number:')
```

```
def countdigits(n):
```

```
n1 = len(n)
```

```
print(n1)
```

```
countdigits(num)
```

python Narayana

Output:-

Enter any number : 12345

5

Enter any number : 788

3.

- Q3. Write a python -function to print power values base on two given base and exponent values.

```
base = int(input('Enter base value:'))
expo = int(input('Enter exponent value:'))

def power(m, n):
 x = base ** expo
 print(x)

power(base, expo)
```

O/p:-

Enter base value: 3

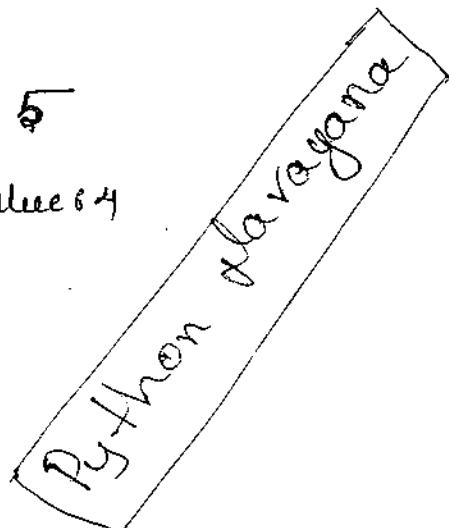
Enter exponent value: 2

9

Enter base value: 5

Enter exponent value: 4

625



Q4) Write a python function to check whether  
we're given string or number.

python Narayana

(324)

```
x = input('Enter either string or number:')
```

```
def strornum(m):
```

```
 if m.isalpha() == True:
```

```
 print(x, 'is a string value')
```

```
 elif m.isnumeric() == True:
```

```
 print(x, 'is a number value')
```

```
 elif m.isalnum() == True:
```

```
 print(x, 'is alpha-numeric value')
```

```
 else:
```

```
 print(x, 'is not a complete string or number
 or alpha-numeric value')
```

```
strornum(x)
```

Output :-

Enter either string or number : python  
python is a string value

Enter either string or number : 1038

1038 is a number value.

Enter either string or number : CS 1035

CS 1035 is alpha-numeric value.

Enter either string or number : Sai@gmail.com.

Sai@gmail.com is not a complete string or number or

alpha-numeric value

25. Write a python function to check whether the given character is vowel or consonant.

```
char = input ('Enter any character')
```

```
def vowelOrConso(ch):
```

```
vowel = 'aeiouAEIOU'
```

```
if ch in vowel:
```

```
 print (char, 'is a vowel')
```

```
else:
```

```
 print (char, 'is a consonant')
```

```
VowelOrConso(char)
```

### O Output

Enter any character : a

a is a vowel.

Enter any character : E

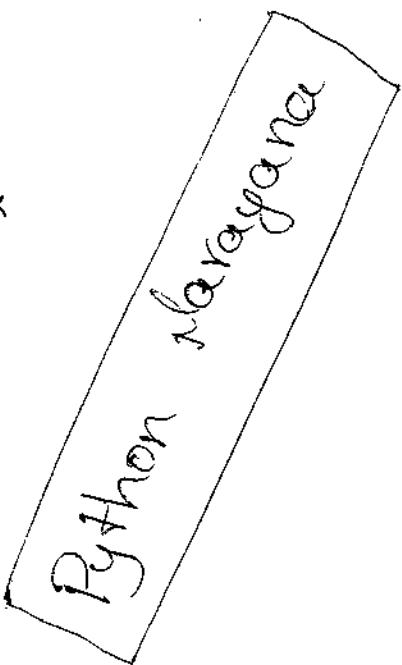
E is a vowel.

Enter any character : x

x is a consonant.

Enter any character : y

y is a consonant.



26 write a python function to print following pattern.

Python Narayana

(326)

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

n=4

```
def pattern1(n):
 for i in range(n):
 for j in range(n):
 for k in range(n):
 print('*', end=' ')
 print('\n')
```

pattern1(n)

27 write a python function to print the following pattern.

\*

\*\*

\*\*\*

\*\*\*\*

n=4

```
def pattern2(n):
 for i in range(n+1):
 for j in range(n+1):
 for k in range(i):
 print('*', end=' ')
 print('\n')
```

print('\*' , end = '')      Python Narayana :-  
print('\n')

(327)

pattern 2(n)

28. Write a python function to print the following  
pattern

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

n = 5

O      def pattern 3(n):

D

for i in range(1, n+1):  
    for j in range(1, (n-i)+1):

Python Narayana

Python Norayana

(328)

Python  
Norayana

## Python Format function

(329)

Some basics about formatting:-

```
print ("This is {} and working as {}".format('Narayana',
 "python Developer"))
```

O/p: This is Narayana and working as python Developer

```
print ("This is {}0{} and working as {}1{}".format('Narayana',
 " python ", " Narayana ", " Developer"))
```

O/p: This is Narayana and working as python Developer.

```
print ("This is {}1{} and working as {}0{}".format('Narayana',
 " python ", " Narayana ", " Developer"))
```

O/p: This is python Developer and working as Narayana.

```
print ("This is {}1{} and working as {}1{}".format('Narayana',
 " python ", " Narayana ", " Developer"))
```

O/p: This is python Developer and working as python developer.

```
print ("we eat {} time per {}".format(3, "day"))
```

O/p we eat 3 time per day. (33)

Display squares and cubes for given numbers

Traditional way.

```
for i in range(1, 11):
```

```
 print(i, " ", i*i, " ", i*i*i)
```

O/p

1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

By using format function:

```
>>> for i in range(1, 11):
```

```
 print ("{}:{}d {}:{}d {}:{}d".format(i, i*i, i*i*i))
```

Output :-

1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

format function with dictionary :-

```
persons = { 'name': 'Sai', "Age": 28 }
```

# traditional way of using key in the sentence.

```
print("My name is "+ persons['name'] + " and iam" + str(persons
[Age]) + " years old")
```

```
print("My name is ", persons['name'], " and iam", str(persons
[Age]), " years old").
```

# by using format function also we can use value with  
Corresponding keys in the Sentences.

python Narayana

Print ("My name is {} and i am {} years old".format  
(persons['name'], persons['Age'])).  
33)

print("My name is {} and i am {} years old".format  
(persons['name'], persons['Age'])).

print("my name is {} and i am {} year  
old".format(persons, persons)).

print("My name is {} and i am {} years  
old".format(persons)).

O/p

my name is Sai and i am 28 years old

My name is Sai and i am 28 years old

my name is Sai and i am 28 years old

My name is Sai and i am 28 years old

My name is Sai and i am 28 years old.

My name is Sai and i am 28 years old.

My name is Sai and i am 28 years old.

format function with list

lst = ["Narayana", "python", 3]

print("My Name is {} and i have been giving  
training on {} for {} years".format(lst))

Output: My name is Narayana and i have been giving training on python for 3 years.

Python Narayana

(33)

### Adding formatting to the place holder:

-for i in range (1,5):

```
 msg = "The current value is {0:02f} ".format(i)
 print (msg)
```

Output:

The current value is 01

The current value is 02

The current value is 03

The current value is 04.

### format function with tuples:

```
lst = ("Sai", "python", 2)
```

```
print ("My name is {0[0]} and i have been training
on {0[1]} for {0[2]} years".format(lst)).
```

Op: My name is Sai and i have been giving training on python for 2 years.

### format dates:

#about dates we will discuss in  
modules concept

Eg:1

```
import datetime
```

```
my_date = datetime.datetime(2017, 10, 16, 23, 20, 44)
```

```
Sentence = "%Y-%B-%d %H.%M.%S".format(my_date)
```

```
print(Sentence)
```

(334)

O/p: October 16, 2017

Eg:2

```
import datetime
```

```
my_date = datetime.datetime(2017, 10, 16, 23, 20, 44)
```

```
print("O: %Y.%B.%d,%Y.%y fell on %A and %W")
```

day of this week and %d day of this month

and %j day of the year".format(my\_date)

O/p:

October 16, 2017 fell on Monday and 1 day of this week  
and 16 day of this month and 289 day of the year.

Some other examplesEg:1

```
age = input("How old are you?")
```

```
height = input("How tall are you?")
```

```
Weight = input("How much do you weigh?")
```

## python Narayana

print ("So, you're {} years old, {} feet tall and {} kgs weight".format (age, height, weight)). (335)

### Output:

How old are you? 27

How tall are you? 5.9

How much do you weigh? 75

So you're 27 years old, 5.9 feet tall and 75kgs weight.

### Ex 2

```
def add(a,b):
```

```
 print ("ADDING {} + {}".format (a,b))
```

```
 return a+b
```

```
def subtract(a,b):
```

```
 print ("SUBTRACTING {} - {}".format (a,b))
```

```
 return a-b
```

```
def multiply(a,b):
```

```
 print ("MULTIPLYING {} * {}".format (a,b))
```

```
 return a*b
```

```
def divide(a,b):
```

```
 print ("DIVIDING {} / {}".format (a,b))
```

```
 return a/b
```

python Narayana  
print ("Let's do some math with just functions!")

age = add (30, 5)

(336)

height = subtract (78, 4)

weight = multiply (90, 2)

iq = divide (100, 2)

print ("Age: 35, Height: 74, weight: 180, IQ: 50")

format (age, height, weight, iq))

### Output:

Let's do some math with just functions!

ADDING 30+5

SUBTRACTING 78-4

MULTIPLYING 90\*2

DIVIDING 100/2

Age: 35 height: 74 weight: 180 IQ: 50.0.

Python Narayana

# Lambda functions

python Narayana

337

1. Lambda function is a small anonymous functions,  
i.e., functions without a name.

2. These functions are throw-away functions, i.e.,  
they are just needed where they have been created.

3. We use a Lambda function when we require a  
nameless function for short time

general syntax of a Lambda function is quite simple

Lambda argument list : expression.

The argument list consists of a comma separated  
list of arguments and the expression is an arithmetic  
expression using these arguments.

Eg:-

a = Lambda x, y : x if x>y else y

print (a(20,5))

20.

In the above script Lambda x, y : x if x>y else  
y is lambda fn whereas x, y are the arguments  
and x if x>y else y is expression.

the above function has no name and it returns  
a function object which is assigned to a identifier 'a'.

→ generally we use def keyword to create a function definition but in this case we "don't use def keyword" instead we used "Lambda keyword"

Q. Can we either def a Lambda methods to create a function.

Q. Write a function to add two numbers?

By using def:

```
def add_nums(x,y):
```

```
 return x+y
```

```
print(add_nums(1,2))
```

By using lambda:

```
a = lambda x, y: x+y
```

```
print(a(1,2))
```

Q. Write a function to find maximum value of given two numbers.

By using def:

```
def max_nums(x,y):
```

```
 if x > y:
```

```
 return x
```

(339)

else :

return y

O/p print (max\_nums(20,5))

20

By using Lambda

a = Lambda x, y: x if x &gt; y else y

O/p print (a(20,5))

20

Q. Write a function to calculate square for given number

By using def

def square\_val(x):

return x\*x

O/p print (square\_val(10))

100

By using Lambda

a = Lambda x: x\*x

O/p : print(a(10))

100.

Lambda functions are mainly used in combination with the functions filter(), map() and reduce().

Filter

filter function mainly takes two arguments, first one is the a function and second one is list of

## Python ~~Numpy~~

arguments.

`r = filter(func, seq)`

(340)

-this function calls all the items from the existing sequence and new sequence is returned which contains elements that are evaluated to true.

Q. Write a function to return only odd numbers from the existing list which contains values from 1 to 20?

`Lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]`

`odd_nums = list(filter(lambda x: (x % 2 == 1), Lst))`

`print(odd_nums)`

O/p [1, 3, 5, 7, 9, 11, 13, 15, 17, 19].

Q. Write a python code to separate positive values from the given list which contains both positive and negative numbers.

`Lst = range(-10, 10)`

`print(Lst)` [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

`a = list(filter(lambda x: x > 0, Lst))`

`print(a).`

## Python Narayana

341

O/p: [1, 2, 3, 4, 5, 6, 7, 8, 9].

Q. write a python script to get intersection elements from both lists?

```
a = [1, 2, 3, 4, 5, 6]
```

```
b = [4, 10, 11, 1]
```

```
lst = list(filter(lambda x: x in a, b))
```

```
print(lst) [4, 1].
```

Map: The map function contains two arguments like,

```
r = map(func, seq)
```

this first argument func is the name of a function and the second a sequence (eg. a list) seq. map() applies the function func to all the elements of the sequence seq. It returns a new list with the elements changed by func.

Eg:

write a function to add elements from different sets.

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

```
b = [11, 12, 11, 10]
```

```
c = [-1, -4, 5, 9]
```

python Narayana

List(map(lambda x, y: x+y, a, b)) [18, 14, 14, 14].  
List(map(lambda x, y, z: x+y+z, a, b, c)) [11, 10, 19, 23].

(312)

Eg:

Write a python script to find squares for all elements in the given list

a = [2, 4, 5, 6]

a = map(lambda x: x\*x, a)

print(a)

[4, 16, 25, 36].

Since it's a built-in, map is always available and always works the same way.

from operator import add

a = [1, 2, 3, 4]

b = (1, 2, 3, 4)

x = map(add, a, b)

print(x) [2, 4, 6, 8].

→ If function is None, the identity function is assumed,  
if there are multiple arguments, map() returns a list  
consisting of tuples containing the corresponding  
item from all iterables (a kind of transpose  
operation)

$m = [1, 2, 3]$

Python Narayana

$n = [1, 4, 9]$

(343)

new-tuple = map (None, m, n)

print (new-tuple)

O/P:  $[(1, 1), (2, 4), (3, 9)]$

We can still use lambda as a function and list of functions are sequence, like.

```
def square_values(x):
 return (x ** 2)
```

```
def cube_values(x):
 return (x ** 3)
```

func = [square\_values, cube\_values]

for r in range(5):

value = map (lambda x: x(r), func)

print value.

O/P:

[0, 0]

[1, 1]

[4, 8]

[9, 27]

[16, 64].

Python Narayana

Reduce:

Reduce function contains two arguments like

$x = \text{reduce}(\text{func}, \text{seq})$

the function continually applies the function func()  
to the sequence seq. It returns a single value.

Syntax

If  $\text{Seq} = [s_1, s_2, s_3, \dots, s_n]$ , calling  $\text{reduce}(\text{func}, \text{seq})$   
works like this,

\* Initially the first two elements of seq will be applied  
to func, i.e.,  $\text{func}(s_1, s_2)$ , the list on which  
 $\text{reduce}()$  works looks now like this:  $[\text{func}(s_1, s_2), s_3, \dots, s_n]$

\* In the next step func will be applied on the previous  
result and the third element of the list, i.e.,  
 $\text{func}(\text{func}(s_1, s_2), s_3)$

The list looks like this now:  $[\text{func}(\text{func}(s_1, s_2), s_3), \dots, s_n]$

\* Continue like this until just one element is left &  
return this element as the result of  $\text{reduce}()$ .

Q. Write a function to find maximum number  
from given list by using Lambda.

345

a = [1, 2, 3, 4]

d = reduce (Lambda x, y: x if x > y else y, a))  
print(d)

4.

Q. write a python script to add all the elements in  
the given list.

a = reduce (Lambda x, y: x+y, [10, 11, 12, 13, 14, 15])  
print(a):

75.

Q. write a python script to add all even no  
before 100.

a = reduce (Lambda x, y: x+y, range(0, 101, 2))  
print(a)

2550.

Nagayana

Python

(346)

Python Narayana

# \* Object Oriented Procedure Programming \*

(30)

1. Introduction to OOP
2. class and object
3. Accessing variables & methods from class
4. Self keyword.
5. Docstring
6. class or static Variables  
*NameSpace*
7. Constructor
8. Instance *Python* non-static variables
9. Data hiding.
10. Data abstraction.
11. Encapsulation
12. Tightly Encapsulation.
13. Inheritance
  - a. Single Level inheritance
  - b. Multi Level inheritance
  - c. Hierarchical inheritance
  - d. Multiple inheritance.
14. polymorphism
  - a. Method Overloading
  - b. Method Overriding.

## Introduction

In all the programs, we have designed our program around functions i.e., blocks of statements which manipulate the data. This is called "procedure-oriented" way of programming. There is another way of organizing your program which is to combine data & functionality & wrap it inside, something called an object. This is called "Object-oriented programming" paradigm.

classes and objects are the two main aspects of object oriented programming. A class creates a new type where objects are instances of the class.

### classes & objects

1. Object is simply a collection of data (variables) & method (functions)
2. Objects are an encapsulation of variables & function into a single entity.
3. objects get their variables & functions from classes.
4. Classes are essentially a template to create

your objects.

5. class is a blueprint for the object. 3/9
  6. A class is the blueprint from the individual objects are created. class is composed of three things, a name, attributes & operations.
- class → data members and member functions or  
 data and functions or  
 data and member or  
 variables & functions or  
 variables & method or  
 states & behaviors.

Eg:- If we think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these description we build the house. House is the object.

We can make many house from the description. So we can create many objects from a class. A object is also called "instance of a class." and the process of creating this object is called "instantiation"

Eg: 2

In an apartment, we may have multiple houses are there and almost all are having same structure we define a class by using class keyword.

### Accessing Variables

```
class ClsName:
```

```
 VarName = "Python"
```

```
 def function (self):
```

```
 print ("python is very simple and easy
language")
```

```
objName = ClsName()
```

```
print (objName.VarName)
```

Output: python.

We can create multiple different objects with the same class (which is having variables & functions)

Each object contains its own copy of the variables.

```
class ClsName:
```

```
 VarName = "Python"
```

```
 def function (self):
```

```
 print ("python is very simple and easy
language") .
```

## Python NameSpace

(351)

```
objNameA = clsName()
objNameB = clsName()
ObjNameB.VarName = "Developer"
Print(ObjNameA.VarName)
Print(ObjNameB.VarName)
```

O/P :- python  
Developer

### Accessing function :-

We can also access the function to the object seperately  
(like a variable)

```
class clsName:
 VarName = "python"
 def function(self):
 print("python is very simple and powerful Language")
objName = clsName()
objName.function()
print(clsName.__doc__)
```

### Output:-

python is very simple & powerful language.

Self:-

Class methods have only one specific difference from ordinary functions - they must have an extra first name that has to be added to the beginning of the parameter list, but you don't give a value for this parameter when you call the method, python will provide it. This particular variable refers to the object itself, and by convention, it is given the name Self.

Although, you can give name for this parameter it is strongly recommended that you use the name Self. Any other name is definitely frowned upon. There are many advantages to using a standard name - any reader of your program will immediately recognize it and even specialized IDE's (Integrated Development Environments) can help you if you use Self.

Note for C++/Java/C# programmers:-

The "Self" in python is equivalent to the "this" pointer in C++ and "this" reference in Java & C#.

- you must be wondering how python gives the value for self and why you don't need to give a value for it.
- An example will make this clear. Say you have a class called 'MyClass' and an instance of this class is called My "myobject"
- When you call a method of this object as myobject.method(args, args), this is automatically converted by python into Myclass.method(myobject, args, args). This is all the special self is about.

- This also means that if you have a method which takes no arguments, then you still have to have one argument - the self.

### DocString:

- The first string is called docstring and it has brief description about the class.
- Docstring is not mandatory but it is recommended.
- When we define a class then class creates a local namespace where all its attributes are defined, attributes are may be data (by func.)

→ There are special attributes also in it that begins with double underscore (-\_) , for eg \_\_doc\_\_ which displays the docstring in that class.

Let's see some examples on class and objects.

```
class ClsName:
```

"This ClsName class contains VarName and function attributes"

VarName = "python".

def function (self):

print ("python is very simple and easy language")

In the above eg just one class is created with one variable and one function.

```
class ClsName:
```

"This ClsName class contains VarName VarName and function attributes"

VarName = "python".

def function (self):

print ("python is very simple & easy language")

objName = ClsName()

Now the object ObjName holds the object of class  
clsName which contains the variable & function that  
are defined in clsName class.

(355)

Now we can access Variables & functions separately.

### Static Variables (or) Class Variables:-

1. The variables which are declared inside the class and outside all the methods are known as class variables or static variables.
2. Class or static variables are shared by all objects.
3. The data which common for all the objects is recommended to represent as static variable or class variable.
4. Memory will be allocated only once for all class or static variables.
5. We can also modify the values of static or class variables.
6. We can access the static or class variables within the class or outside the class also by using class name.

(356)

```

class MyClass:
 i = "python" ## static or class variable
 j = "Dev" #### static or class variable

 def DisMethod(self):
 print(MyClass.i) ## accessing static variables
 print(MyClass.j) with class name

x1 = MyClass()
print(x1.i) ## accessing static variable outside
print(x1.j) the class
x1.DisMethod() ## it works like DisMethod(x1)

```

Output :-

python

Dev

python

Dev.

- In C++ and Java, we can use static keyword to make a variable as class variable. The variables which don't have preceding static keyword are instance variables.
- Python doesn't require a static keyword. All variables which are declared in class declaration are class variables or static variables. And variables which

Python Narayana

are declared inside class methods are instance Variables or non-static variables.

(357)

## Constructors

1. Constructor is a special function which we call automatically when we create object for respective class.
2. It can be defined by using `--init--()` in python
3. It will not return anything.
4. It is mainly used for initializing the variable.

`--init--` is a kind of constructor, when a instance of a class is created.

→ python calls `--init--()` during the instantiation to declare additional behavior that should occur when a class is instantiated, basically setting up some beginning values for that object or running a routine required on instantiation.

`init` is an abbreviation for initialization.

## Non-Static Variables (or) Instance Variables

1. The variables which are declared inside the method and the 'self' keyword is known as instance Variable (or) non-static Variable.

2. Instance Variables are owned by the specific instances of a class. This means the instance of non-static variables are different for different objects (every object has a copy of it). 358
3. Instance Variables are always introduced with the word self. They are typically introduced & initialized in a constructor method name `__init__`.
4. We can define instance variables in a constructor or in a method.
5. The data which is separate for every object is recommended to represent as a instance variable or non-static variable.
6. Memory will be allocated for all instance variables whenever we create a object.
7. Instance variables of a class can be accessed within the same class by using 'self' keyword. And same variables also can be accessed outside the class by using reference variable name.

Eg:1

```

class employee:
 Loc = "Hyderabad" # class variable

 def __init__(self, Empno, Ename, Sal):
 self.Empno = Empno # instance variable
 self.Ename = Ename # instance variable
 self.Sal = Sal # instance variable

 def display(self):
 print("Emp number is ", self.Empno)
 print("Emp name is ", self.Ename)
 print("Emp salary is ", self.Sal)
 print("Emp Location is ", employee.Loc)

emp1 = employee(101, "Sai", 10000)
emp1.display()

emp2 = employee(102, "Nani", 20000)
emp2.display()

emp3 = employee(103, "Renu", 30000)
emp3.display()

```

Output:-

Emp number is 101

Emp name is Sai

\* Emp Salary is 10000

Emp Location is Hyderabad.

Emp number is 102

Emp name is Nani

Emp salary is 20000

Emp location is Hyderabad.

Emp number is 103

Emp name is Renu

Emp salary is 30000

Emp location is Hyderabad.

Eg:2

```
class MyClass:
```

```
 def method1(self):
```

```
 self.i = "python"
```

```
 def display(self):
```

```
 print(self.i)
```

```
obj1 = MyClass()
```

```
obj1.method1()
```

```
obj1.display()
```

# returns python.

obj.i = "It is very easy" Python Narayana  
361

obj1.display() #returns It is very easy

Obj2 = MyClass()

Obj2.method1()

Obj2.display() #returns Python.

Obj2 = "It is very powerful"

Obj2.display() #returns It is very powerful.

Obj1.display() #returns It is very easy

Obj2.display() #returns It is very powerful.

### Output

python

It is very easy

python

It is very powerful

It is very easy

It is very powerful.

Python Narayana

Data hiding:- Outside person can't access our internal data directly or our internal data should not go out directly this OOP feature is nothing but data hiding.

After validation or authentication only, outside person can access our internal data.

Eg:1 after providing proper username and password we can able to access our gmail inbox information.

Eg:2 Even though we are valid customer of the bank, we can able to access our account information & we can't access others account information.

By declaring data member (variable) as private (--) only we can achieve data hiding.

Eg:-

`--a = 100`

`def getBal():`

`#Validation`

`return --balance`

- the biggest advantage of data hiding is security
- It is highly recommended to declare a data member

as private (with underscore)

python Narayana..

(363)

## Data abstraction :-

→ Hiding internal information and just highlight the set of Services what we are offering is the concept of abstraction.

→ Through bank atm GUI Screen, bank people are highlighting the set of Services what they are offering without highlighting internal implementation, like the Server they used to work, the database they used to store the customer details, the Language they used to implement communication b/w application & database.

1. Outside person doesn't know how it is implemented internally, so "Security" is provided.

2. For Example a bank ATM GUI is developed by python now, and python program execution speed is not upto mark, so the new language is there "Sython" now we can change internal implementation from python to Sython without effecting the GUI display and end-user. We can do any modifications to internal implementation without effecting to end-user

Python Narayana

that's why "enhancement" became very easy.

(364)

3. for example we need to know how the ATM Card is implemented and everything about ATM Card before use, then nobody can use ATM Cards. Because it's not possible to know everything. that's why without knowing anything about internal implementation we are using ATM cards, and also whatsapp so here it's improving "easiness" to end-user
4. So we are able to do any internal implementation changes without effecting end user so here "Maintainability" of application becomes very easy.

Python Narayana

## Encapsulation:

1. The process of binding the data members and corresponding methods into a single unit is called encapsulation.
2. Every python class is example of encapsulation.
3. Class student has data members plus data methods is encapsulation.
4. Encapsulation is nothing but "combination of data hiding" and data abstraction".
5. If any component follows data hiding and data abstraction then that component is called "encapsulated Component".
6. Encapsulation is about ensuring the safe storage of data as attributes in an instance.
7. Encapsulation tells us that:
  - a. Data should only be accessed through instance methods.
  - b. Data should always be valid based on the validation requirements set in the class methods.

C And Data should be safe from changes  
by external processes.

(366)

Encapsulation = data hiding + abstraction.

Eg: Capsule with medicine inside

class account:

--balance = 100000

def getBal():  
 #validation

print (--balance)

def setBal(self, balance):  
 #validation  
 self.balance = balance

Let's take bank ATM GUI Screen and imagine there are only two buttons are there, like BalanceEnquiry and update Balance.

The above class contains data members which are declared as private for hiding from the others and also class contains methods like getBal() and setBal().

→ When the end user clicks on BalanceEnquiry button then automatically it will call

→ getBal() method, when user clicks on update Balance button then automatically it will call ~~get~~ SetBal() method.

(367)

→ The complete and confidential data is hidden inside the class and just abstraction is given in the GUI.

So finally the above class contains data members and methods.

The main advantages of encapsulation are.

1. We can achieve security.
2. Enhancement will become very easy.
3. It improves maintainability of application.

→ The main advantage of encapsulation is security but the main disadvantage of encapsulation is it increases length of the code and slows down execution.

→ A class is said to be tightly encapsulated if and only if each and every variable declared as private.

→ Whether class contains corresponding getter or Setter or not and whether these methods are declared as public or not these things we are not required to check.

class classA:

--a = 10

--x = 100

class classB(classA):

b = 20

class classC(classA):

--c = 30.

Python Narayana

(368)

Here classA and classC are tightly encapsulated classes because each and every variable under these two classes are private so nobody can access from outside but everyone can access variable under classB.

If parent class is not tightly encapsulated then automatically all corresponding its child classes are not tightly encapsulated.

class classA:

a = 10

x = 100

class classB(classA):

--b = 20

--y = 40

class classC(class A):

Python Narayana 367

- - ~~a~~ = 30

- - z = 50

(369)

Here, even though classB and class c are having all private variables but these two classes are the child classes of classA which is not tightly encapsulated class.

Here, all non-private variables from classA can be accessed from classB and classc that's why classB and classc are not tightly encapsulated classes.

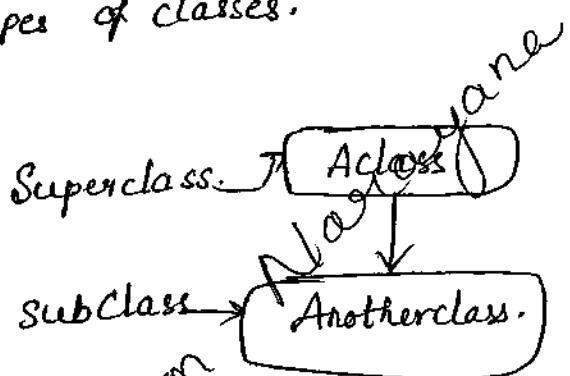
## Inheritance

Python Narayana

(370)

## Inheritance:

- Classes can be derived from other classes. The derived class (the class that is derived from another class) is called a Subclass.
- The class from which it's derived is called the superclass. The following figure illustrates these two types of classes.



- The subclass inherits state and behavior in the form of variables and methods from its superclass. The subclass can use just the items inherited from its superclass as is or the subclass can modify or override it. So as you step in the hierarchy the classes become more and more specialized.
- We use inheritance for code reusability.

1. Single Level Inheritance.
2. Multi-Level Inheritance.

3. Multiple inheritance.

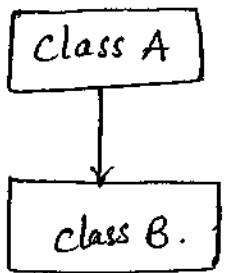
Python Notes

4. Hierarchical inheritance

(37)

### Single Level inheritance

The process of inheriting all attributes from one class to the another class is known as Single Level inheritance.



Eg:-

class A:

a<sub>1</sub> = 100

def method A1(self):

print('This is method A1 belongs to class A')

def method A2(self):

print('This is method A2 belongs to class A')

class B(A):

b<sub>1</sub> = 200

def method B1(self):

print('This is method B1 belongs to class B')

def method B2(self):

print('This is method B2 belongs to class B')

```

ObjB = B()
print(ObjB.b1)
ObjB.methodB1()
ObjB.methodB2()

print(ObjB.a1)
ObjB.methodA1()
ObjB.methodA2()

```

Output:-

200

this is method B1 belongs to class B.

this is method B2 belongs to class B.

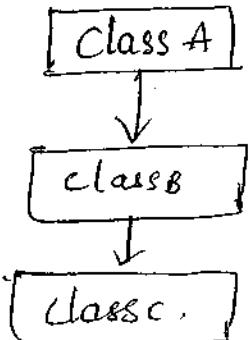
100

this is method A1 belongs to class A

this is method A2 belongs to class A.

Multi-Level inheritance:-

The process of inheriting all attributes from one base class to derived class, again from this derived class to another derived class & so on.



(313)

class A:

a1 = 100

def methodA1(self):

print('This is method A1 belongs to class A')

def methodA2(self):

print('This is method A2 belongs to class A')

class B(A):

b1 = 200

def methodB1(self):

print('This is methodB1 belongs to class B')

def methodB2(self):

print('This is method B2 belongs to class B')

class C(B):

c1 = 300

def methodC1(self):

print('This is method C1 belongs to class C')

def methodC2(~~this is method C2 belongs to class C~~  
              self):

print('This is method C2 belongs to class C')

class D(C):

d1 = 400

def methodD1(self):

print('This is method D1 belongs to class D')

```
def methodD2(self):
```

    print ('this is methodD2 belongs to class D')

(374)

```
objD = D()
```

```
print (objD.d1)
```

```
objD.methodD1()
```

```
objD.methodD2()
```

```
print (objD.a1)
```

```
objD.methodC1()
```

```
objD.methodC2()
```

```
print (objD.b1)
```

```
objD.methodB1()
```

```
objD.methodB2()
```

```
print (objD.a1)
```

```
objD.methodA1()
```

```
objD.methodA2()
```

### Output

400

this is method D1 belongs to class D

this is method D2 belongs to class D

300

this is method C1 belongs to class C

this is method C2 belongs to class C

200

this is method  $B_1$  belongs to class B. Python Narayana  
this is method  $B_2$  belongs to class B.

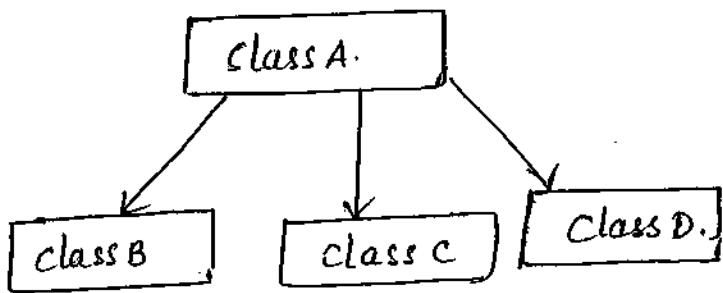
(375)

100

this is method  $A_1$  belongs to class A  
this is method  $A_2$  belongs to class B.

## Hierarchical Inheritance :-

the process of inheriting all attributes from one base class into multiple derived ~~called~~ classes is called



Class A :-

$a_1 = 100$

def method  $A_1$  (self):

print ('This is method  $A_1$  belongs to class A')

def method  $A_2$  (self):

print ('This is method  $A_2$  belongs to class A')

Class B(A) :-

$b_1 = 200$

def method  $B_1$  (self):

Python Narayana

Python Narayana

```
print('this is method B1 belongs to class B')
```

```
def method B2(self):
```

(376)

```
print('this is method B2 belongs to class B')
```

```
class C(A):
```

```
c1 = 300.
```

```
def method C1(self):
```

```
print('this is method C1 belongs to class C')
```

```
def method C2(self):
```

```
print('this is method C2 belongs to class C')
```

```
class D(A):
```

```
d1 = 400
```

```
def method D1(self):
```

```
print('this is method D1 belongs to class D')
```

```
def method D2(self):
```

```
print('this is method D2 belongs to class D')
```

```
ObjD = D()
```

```
print(ObjD.d1)
```

```
ObjD.method D1()
```

```
ObjD.method D2()
```

```
print(ObjD.a1)
```

```
ObjD.method A1()
```

```
ObjD.method A2()
```

```
objc = C()
print (objc.a1)
objc.methodA1()
objc.methodA2()

objB = B()
print (objB.b1)
objB.methodB1()
objB.methodB2()
print (objB.a1)
objB.methodA1()
objB.methodA2()
```

Python Narayana

(377)

Python Narayana

### Output

400

this is method D<sub>1</sub> belongs to class D

this is method D<sub>2</sub> belongs to class D

100

this is method A<sub>1</sub> belongs to class A

this is method A<sub>2</sub> belongs to class A

300

this is method C<sub>1</sub> belongs to class C

this is method C<sub>2</sub> belongs to class C

100

this is method A<sub>1</sub> belongs to class A.  
 this is method A<sub>2</sub> belongs to class A.

200

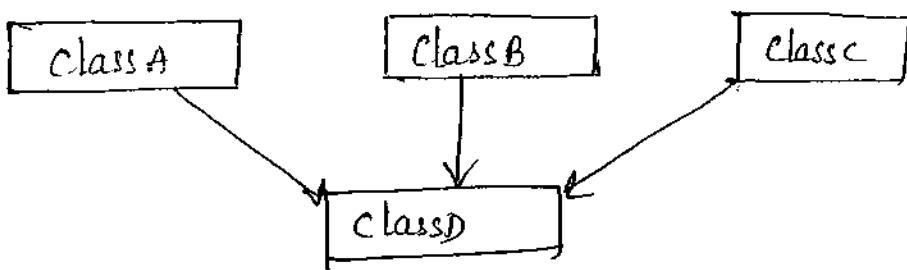
this is method B<sub>1</sub> belongs to class B.  
 this is method B<sub>2</sub> belongs to class A

100

this is method A<sub>1</sub> belongs to class A  
 this is method A<sub>2</sub> belongs to class A.

### Multiple Inheritance

The process of inheriting all attributes from multiple base classes into single derived class this process is known as multiple inheritance.

Eg:

class A :

a<sub>1</sub> = 100def method A<sub>1</sub> (self) :    print('this is method A<sub>1</sub> belongs to class A')

Python Narayana

```
def method A2(self):
 print ('this is method A2 belongs to class A')
```

(379)

class B:

d1 = 200

```
def method B1(self):
 print ('This is method B1 belongs to class B')
```

```
def method B2(self):
```

```
 print ('This is method B2 belongs to class B')
```

class C:

c1 = 300

```
def method C1(self):
```

```
 print ('This is method C1 belongs to class C')
```

```
def method C2(self):
```

```
 print ('This is method C2 belongs to class C')
```

class D (C, B, A):

d1 = 100

```
def method D1(self):
```

```
 print ('This is method D1 belongs to class D')
```

```
def method D2(self):
```

```
 print ('This is method D2 belongs to class D')
```

objD = D()

```
print (objD.d1)
```

Python Narayana

objD. method D1()

objD. method D2()

print (objD. c1)

ObjD. method C1()

objD. method C2()

print (objD. b1)

ObjD. method B1()

ObjD. method B2()

print (objD. a1)

ObjD. method A1()

objD. method A2()

Python Narayans:

(380)

### Output:-

400

this is method D1 belongs to class D

this is method D2 belongs to class D

300

this is method c1 belongs to class C

this is method c2 belongs to class C.

200

this is method B1 belongs to class B.

this is method B2 belongs to class B.

100

this is method A1 belongs to class A

this is method A2 belongs to class A.

## Polymorphism

- This is also a "built in" Python eg of polymorphism.
- the same operation results in different behaviors depending on the type of data is given.
- The same idea (same operation - different behavior) can be applied to our own class and objects.
- Polymorphism is the ability to leverage the same interface for different underlying forms such as data types or classes. This permits functions to use entities of different types at different times.
- For object oriented programming in Python, this means that a particular object belonging to a particular class can be used in the same way as if it were a different object belonging to a different class.
- Polymorphism allows for flexibility and loose coupling so that code can be extended and easily maintained over time.
- Generally polymorphism supports method overloading & method overriding but Python will not support

method overriding.

python Narayana

(38)

## Method Overloading:

The concept of defining multiple methods with same name but different number of parameters is called method Overloading.

class ClassA:

def method1(self):

print('This method belongs to class ClassA')

class ClassB:

def method1(self, a):

Print('This method belongs to class ClassB')

objB = ClassB()

objB.method1(10)

objB.method1() # it will return error so  
overloading will not support.

Output:

this method belongs to class ClassB.

## Method Overriding:

The concept of defining multiple methods with the same name with the same number of parameters is called method overriding.

```
class classA:
```

```
 def method1(self):
```

```
 print('this method belongs to class classA')
```

```
class classB:
```

```
 def method1(self):
```

```
 print('this method belongs to class classB')
```

```
Obj B = classB()
```

```
ObjB. method1()
```

### Output:-

this method belongs to class classB.

Real time example for method Overloading:

```
class BankAccount:
```

```
 def __init__(self, balance = 10000):
```

```
 self.balance = balance
```

```
 def deposit(self, value):
```

```
 self.balance = self.balance + value.
```

```
 print('The current balance is:', self.balance)
```

```
 def withdraw(self, value):
```

```
 self.balance = self.balance - value.
```

```
 print('The current balance is:', self.balance)
```

```
class SavingAccount(BankAccount):
```

```
 def __init__(self, balance = 10000):
```

python Narayana

self.balance = balance.

self.balance = self.balance + (value \* 1.03) (384)

print ('The current balance in savings account  
is:', self.balance)

Class CurrentAccount(BankAccount):

def \_\_init\_\_(self, balance = 10000):

self.balance = balance.

def withdraw(self, value):

if value > 1000:

print ('you can withdraw less than 1000 only')

else:

self.balance = self.balance - value.

print ('your current amount in current  
account is:', self.balance)

SA = SavingsAccount()

CA = CurrentAccount()

while True:

print ('1. Savings Account')

print ('2. Current Account')

MOption = int(input('please select the account  
type:'))

If MOption == 1:

Print ('1. withdraw')

Print ('2. Deposite')

SOption = int(input('please select any operation type:'))

if SOption == 1:

385

value = int(input('please enter amount to withdraw from savings account:'))

SA.withdraw(value)

elif SOption == 2:

value = int(input('please enter amount to deposite in savings account:'))

SA.deposite(value)

else:

print('you entered', SOption, 'it is invalid operation')

elif MOption == 2:

print('1. withdraw')

print('2. Deposete')

SOption = int(input('please select any operation type:'))

if SOption == 1:

value = int(input('please enter amount to withdraw from current account:'))

CA.withdraw(value)

elif SOption == 2:

value = int(input('please enter amount to deposite in current account:'))

CA.deposite(value)

```
else:
```

```
 print('you entered', soption, 'its invalid operation')
```

```
else:
```

```
 print('you entered', noption, 'its invalid Account Type')
```

```
break
```

 # this example is done without break statement

### Output:-

1. Saving Account

2. Current Account

please select the account type:1

1. withdraw

2. Deposite

please select any operation type:1

please enter amount to withdraw from savings account:2000

the current balance is : 8000

### Output:-

1. Saving Account

2. Current Account

Please select the account type:1

1. withdraw

2. Deposite

please select any operation type:2

please enter amount to deposite in savings account: 3000

the current balance in savings account is: 11000.6

387

### Output3:

1. Savings Account
2. Current Account

Please select the account type: 2

1. withdraw
2. Deposite

Please select any operation type: 1

Please enter amount to withdraw from current account: 1500

You can withdraw less than 1000 only.

### Output4:

1. Savings Account
2. Current Account

Please select the account type: 2

1. withdraw
2. Deposite

Please select any operation type: 1

Please enter amount to withdraw from current account: 500

Your current amount in current account is 9500

Python Norayane

(388)

Output 5 :-

1. Savings Account
2. Current Account

Please select the account type :-

1. withdraw.

2. Deposite

Please select any operation type :-

Please enter amount to deposit in current account : 2000  
The current balance is : 11500.

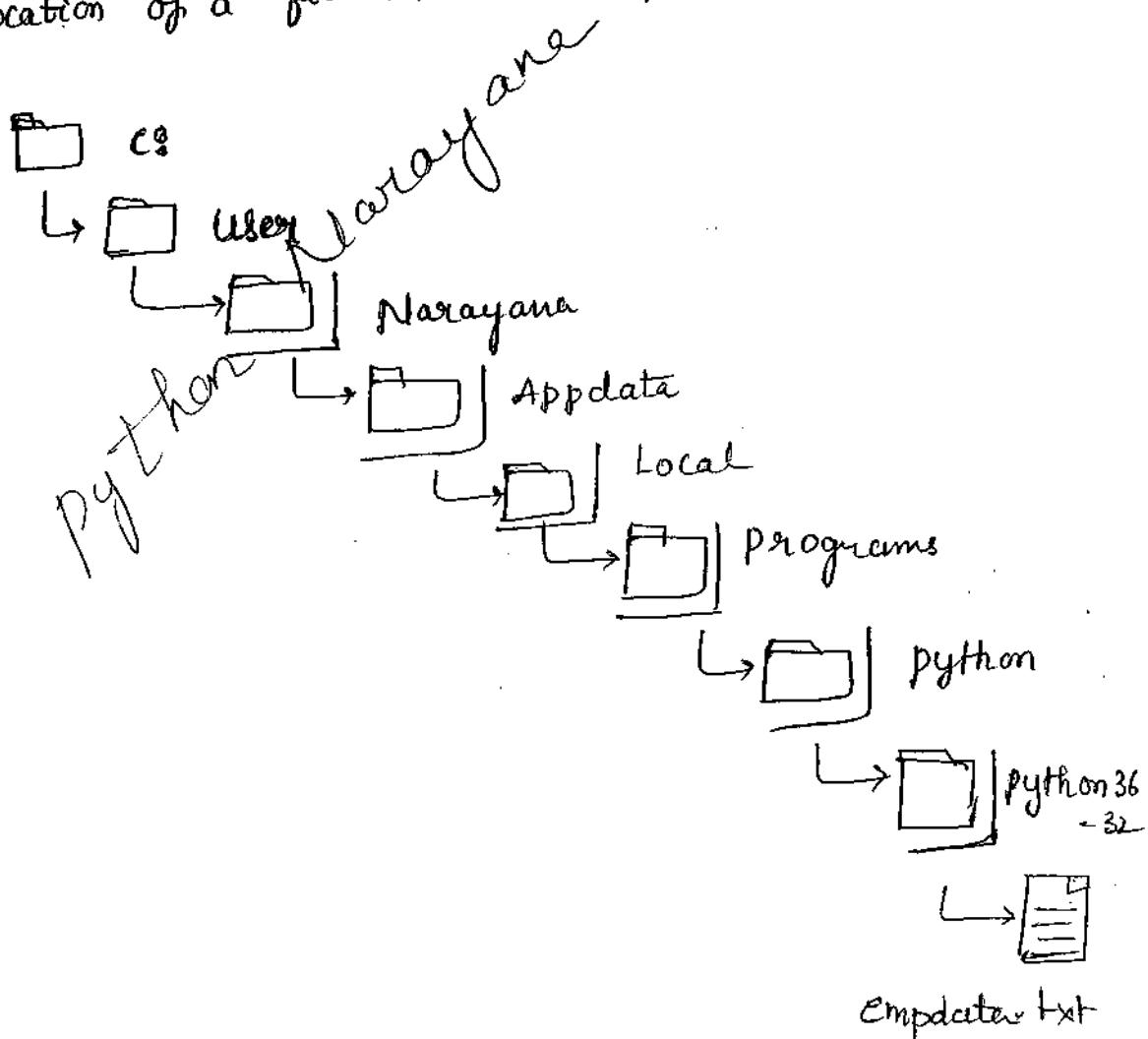
*Python Norayane*

## \* Reading & Writing files\*

(389)

Variables are a fine way to store data while your program is running, but if you want your data to persist even after your program has finished, you need to save it to a file.

A file has two key properties: a filename (usually written as one word) and a path. The path specifies the location of a file on the computer.



there is a file on my windows 10 laptop with the  
filename empdata.txt in the path.

(39)

C:\users\Narayana\AppData\local\programs\python\python36-32'  
these all refer to folders or directories. file folders  
can contain files and other folders.

for eg, empdata does is in the python 36-32 folder  
which is inside the python folder, which is inside the  
program folder, which is inside the Local folder,  
which is inside the Appdata folder, which is inside  
the Narayana folder, which is inside the user, which  
is inside C drive.

C:\ is called root folder which contains all other  
folders.

### Working with Slashes (\ or /) :-

On windows, paths are written using backslashes (\)  
as the separator b/w folder names. Osx and Linux,  
however, use the forward slash (/) as their path  
separator.

→ If we want our programs to work on all operating systems we will have to write our python scripts to handle both cases.

(31)

→ Fortunately, this is simple to do with the os.path.join() function.

→ If we pass it the string values of individual file and folder names in our path, os.path.join() will return a string with a file path using the correct path separator.

```
>>> import os
>>> os.path.join('a', 'b', 'c')
'a\\b\\c'
```

```
>>> os.path.join('users', 'Narayana', 'AppData', 'Local', 'programs'
'python', 'python 36-32')
'users\\Narayana\\Appdata\\Local\\programs\\python\\
python 36-32'
```

→ The os.path.join() function is helpful if we need to create strings for filenames.

python Narayana

for eg, the following example joins names from a list of filenames to the end of the folder's name: (392)

```
>>> files = ["empdetails.txt", 'cutedata.docx', 'productsdata.csv', 'salesdetails.txt'].
```

```
>>> for name in files:
```

```
 print(os.path.join("C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32", name)).
```

## The Current Working Directory

Every program that runs on our computer has a current working directory or cwd. Any filenames or paths that do not begin with the root folder are assumed to be under the current working directory. We can get the current working directory as a string value with the os.getcwd() function & change it with os.chdir()

```
>>> os.getcwd()
```

```
=> C:\\users\\Narayana\\AppData\\Local\\programs
```

```
\\python\\python 36-32'.
```

```
>>> os.chdir("E:\\FileFolder") python Narayana
```

```
>>> os.getcwd()
```

```
'E:\\ FileFolder'
```

(393)

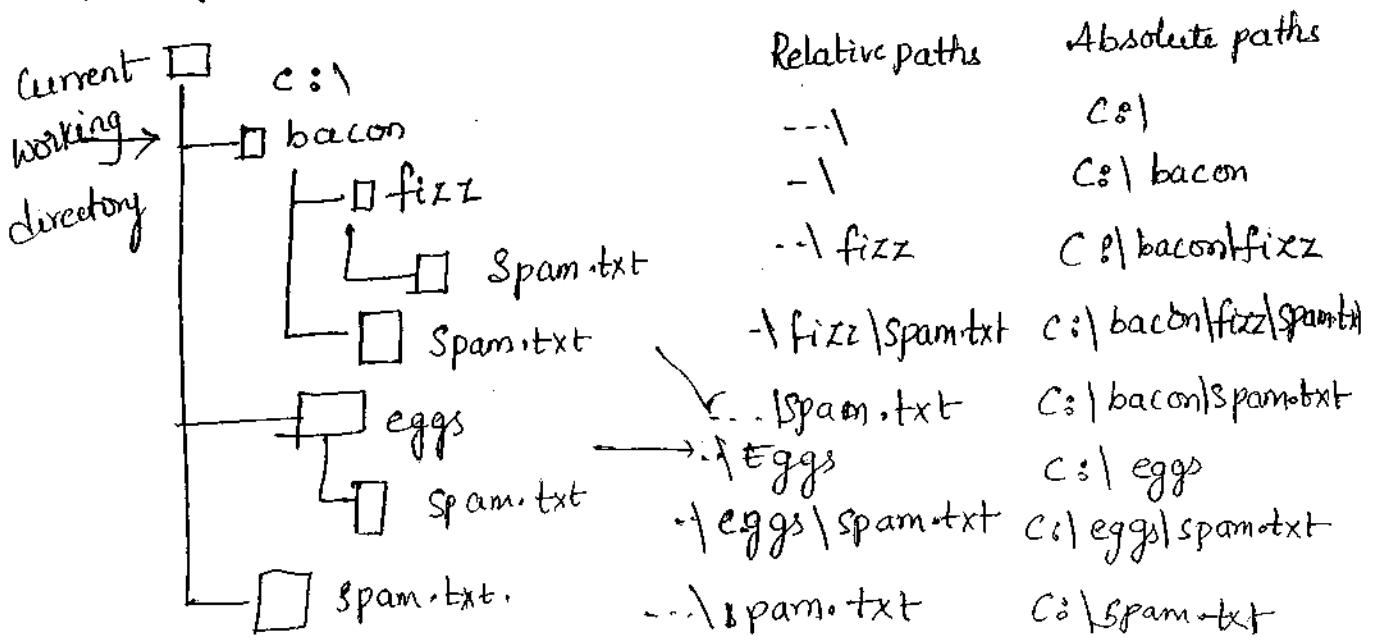
## Absolute Vs Relative paths

There are two types to specify a file path.

- \* An absolute path, which always begins with the root folder.
- \* A relative path, which is relative to the program's current working directory.

There are also the dot(.) and dot-dot(..) folders. These are not real folder but special names that can be used in a path. A single period ("dot") for a folder name is shorthand for "this directory".

Two periods ("dot-dot") means "the parent folder".



## File Sizes & Folder Contents

python Narayana

(31)

Once we have ways of handling file paths, we can then start gathering information about specific files and folders. The os.path module provides functions for finding the size of a file in bytes and the files and folders inside a given folder.

- Calling os.path.getsize(path) will return the size in bytes of the file in the path argument.
- Calling os.listdir(path) will return a list of filename strings for each file in the path argument. (Note that this function is in the OS module not os.path)

```
>>> os.path.getsize('C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32\\empdata.txt')
```

94

Here, empdata.txt file size is 94 bytes

```
>>> os.listdir("C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32").
```

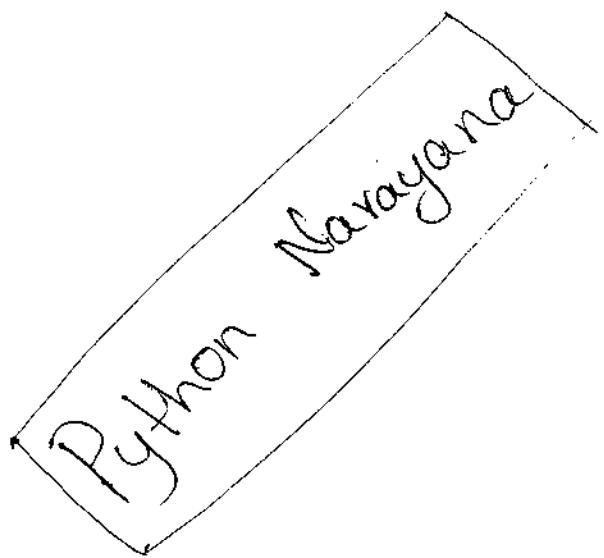
```
['allfunctions.py', 'bankexample.py', 'browser.py', 'browserify.py',
 'compare.py', 'data.pdf', 'decoratorexample.py', 'details.docx']
```

Python Narayana  
395

```
'details.docx', 'empdata.txt', 'multithreading.py', 'mydata.pdf',
'myfile.py', 'ourdata.csv', 'python3.dll', 'snakegame.py', 'tcl',
'test.py', 'testing.py', 'testing12.py', 'tools', 'unittest1.py',
'urdata.docx', 'userexp.py', 'ut.py', 'ut1.py', 'vcruntime140.
dll', 'webpage.py', 'while.py', 'zipp.py', '--pycache--']
```

Here, it is displaying all files which are in python  
36-32 folder.

→ If we want find the total size of all the files in  
this directory , then we use os.path.getsize() and  
os.listdir() together.



## \* File Handling \*

python Narayana

(396)

1. In python we can open any text file with open()
2. open() function has two arguments like open('filename', 'mode')
3. the argument 'filename' is a file name or path of the file.
4. the second argument 'mode' is the type of operation that will be performed on the file.
5. the different types of modes are w (write), r (read), a (append), r+ (both read and write)
6. the default argument is r (read)  
Create a ~~empty~~ text file in the folder where python is installed.

C:\users\Narayana\AppData\Local\Programs\Python

Python 36-32\Employees.txt.

### Writing the data to files

```
a = open('Employees.txt', 'w')
```

```
a.write('Python is easy language')
a.close()
```

Python Narayana  
>Thispc > LocalDisk(C:) > user > Narayana > AppData > local > programs

python > python 36-32

Employees - Notepad.

397

File Edit Format View Help

Python is easy language.

Appending new data to the existing file:

```
a = open('Employees.txt', 'a')
```

```
a.write('\npython is also simple language\npython is
powerful language too')
```

```
a.close()
```

Thispc > LocalDisk(C:) > user > Narayana > AppData > local > Python > py > Python  
36-32

Employees - Notepad.

File Edit Format View Help.

Python is easy language.

python is also simple language.

python is powerful language too.

Notes Now lets take the following data in the file  
to perform read operations.

this pc > local disk(C:) > user > Narayana > Appdata > local > programs > Python > Python

36-32

Employees - Notepad.

(393)

file Edit Format View Help

tcs, narayana, Venu, balu

wipro, swresh, naresh, Ramesh.

info, venu, amala, kavya, Ramu.

cts, veni, Satya, durga, ramya.

Displaying all the data from the file

```
a = open('Employees.txt', 'r')
```

```
x = a.read()
```

```
print(x)
```

or

```
print(open('Employees.txt', 'r').read())
```

Output

tcs, narayana, Venu, balu.

wipro, swresh, naresh, Ramesh

info, venu, amala, kavya, Ramu.

cts, veni, Satya, durga, ramya.

Python Narayana

## Displaying first line from the file in Python Narayana

(399)

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
print(v)
```

(Q1)

```
print(open('Employees.txt', 'r').readline())
```

(Q1).

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines.
```

```
print(v[0]).
```

(Q1)

```
print(open('Employees.txt', 'r').readlines()[0])
```

O/p:-

tcs, narayana, venu, bala.

## Displaying second line from the file.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
print(v[1])
```

(Q1)

```
print(open('Employees.txt', 'r').readlines()[1])
```

O/p:-

wipro, swresh, Naresh, ramesh

Python Narayana

Displaying 3<sup>rd</sup> line from the file. Python Narayana

(400)

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
print(v[2])
```

(8U)

```
print(open('Employees.txt').readlines()[2])
```

O/p:-

info, venu, amala, kavya, ramya.

Displaying last line in the file.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
Print(v[-1])
```

(8U)

```
print(open('Employees.txt', 'r').readlines()[-1])
```

O/p:-

cts, veni, satya, durga, ramya.

Counting number of lines in the file.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
Print(len(v))
```

O/p :- 4

Python Narayana

Displaying first word from each line. Python Narrator

(Q1)

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
for i in v:
```

```
 k = i.split(',')
```

```
 print(k[0])
```

```
(Q1)
```

```
for i in open('Employees.txt', 'r').readlines():
```

```
 print(i.split(',') [0])
```

O/p:-

tcs

wipro

info

cts.

Displaying second word from each line.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
for i in v:
```

```
 k = i.split(',')
```

```
 print(k[1])
```

```
(Q1)
```

```
for i in open('Employees.txt', 'r').readlines():
```

```
 print(i.split(',') [1])
```

Python Narrator

## Output:-

Python Narayana.

(402)

Narayana

Suresh

rene

Veni

Displaying last word from the each line

```
a = open ('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
for i in v:
```

```
 k = i.split ()
```

```
 print (k [-1])
```

(Q4)

```
for i in open ('Employees.txt', 'r').readlines ():
```

```
 print (i.split () [-1])
```

## Output:-

balu

Ramesh

rameu

ramya

Displaying first word in the first line.

```
a = open ('Employees.txt', 'r')
```

```
v = a.readlines ()
```

```
x = v [0],
```

## Python Narayana

(Q3)

Print(y[0])

(a)

Print(open('Employees.txt', 'r').readlines()[0].split(',')[0])

O/p

tcs.

Displaying second word in the first line

a = open('Employee.txt', 'r')

v = a.readlines()

x = v[0]

y = x.split(',')

print(y[1])

(b)

print(open('Employees.txt', 'r').readlines()[0].split(',')[1])

O/p

Narayana.

Displaying third word in the Second line.

a = open('Employees.txt', 'r')

v = a.readlines()

x = v[1].

y = x.split(',')

Print(y[2])

(Python) Narayana

```
print(open('Employees.txt', 'r').readlines()[1].split(','))
```

(404)

O/p:

Naresh.

\* Displaying last word in the last line.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
y = v[-1].split(',')[-1]
```

```
Print y
```

(OR)

```
print(open('Employees.txt', 'r').readlines()[-1].split(',')[-1])
```

O/p:

Ramya.

\* Displaying first character from each line.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
for i in v:
```

```
 print(i[0])
```

(OR)

```
y = open('Employees.txt', 'r').readlines()
```

```
for i in v:
```

```
 print(i[0])
```

Output:

t

w

i

c

Python Narayana..

(405)

Displaying second character from each line.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
for i in v:
```

```
 print(i[1])
```

(ii)

```
v = open('Employees.txt', 'r').readlines()
```

```
for i in v:
```

```
 print(i[1])
```

Output:

C

i

n

t

Displaying last character from each line.

```
a = open('Employee.txt', 'r')
```

```
v = a.readlines()
```

for i in v:

python Narayana...

print(i[-2])

(Q6)

(Q8)

v = open('Employees.txt', 'r').readlines()

for i in v:

print(i[-2])

Output:-

u

h

u

y

# last but one character from last-line.

Display first character from the file,

a = open('Employees.txt')

v = list(a.read())

print(v[0])

(Q9)

print(list(open('Employees.txt').read())[0])

Q9 t

Display last character from the file

a = open('Employees.txt')

v = list(a.read())

print(v[-1])

(Q9)

Python Narayana

```
print(list(open('Employees.txt').readl())[-1])
```

Output

(Q7)

a.

Displaying first character of second word in the first line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[0].
```

```
y = x.split(',')
```

```
z = y[1]
```

```
print(z[0])
```

(Q8)

```
print(open('Employees.txt').readlines()[0].split(',')[1][0])
```

O/p n

Displaying third character of third word in the third line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[2].
```

```
y = x.split(',')
```

```
z = y[2]
```

```
print(z[2])
```

(Q9)

python Narayana

```
print(open('Employees.txt').readlines()[2].split(',')[2][2])
```

O/p:-

(Ans)

a.

Displaying second character of third word in the fourth line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[3]
```

```
y = x.split(',')
```

```
z = y[2]
```

```
print(z[1])
```

(Q1)

```
print(open('Employees.txt').readlines()[3].split(',')[2][1])
```

O/p:-

a.

Displaying fourth character in the first line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[0]
```

```
print(x[4])
```

(Q1)

```
print(open('Employees.txt').readlines()[0][4])
```

O/p:-

n.

Displaying 10<sup>th</sup> character in the third line Python ~~Answer~~

```
a = open('Employees.txt')
v = a.readlines()
x = v[2]
print(x[10])
(O/P)
print(open('Employees.txt').readlines()[2][10])
```

O/P:-

a.

Displaying number of characters in the file (including commas)

```
a = open('Employees.txt', 'r')
v = a.read()
print(len(v))
```

O/P:-

103.

Displaying number of characters in the file (excluding commas)

```
a = open('Employees.txt', 'r')
v = a.readlines()
c = 0
for i in v:
 x = i.split(',')
 c = c + len(j)
```

Print(c)

O/P: 89

Python Narayana  
13/3

Displaying number of characters in the first line (including commas).

(410)

```
a = open('Employees.txt', 'r')
```

```
v = a.readline()
```

```
print(len(v))
```

(a)

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
print(len(v[0]))
```

O/p: 23.

Displaying number of characters in the first line (excluding commas)

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
x = v[0].split(',')
```

```
k = 0
```

```
for i in x:
```

```
 k = k + len(i)
```

```
print(k)
```

Python Narayana

O/p:

20

Displaying number of characters in the second line (including commas)

Python Narayana

HII

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
print(len(v[1]))
```

O/p: 27

Displaying number of characters in the last line (excluding commas)

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[-1].split(',')
k = 0
```

```
for i in x:
 k = k + len(i)
print(k)
```

Python Narayana

O/p: 22

Displaying number of characters in the last line (including commas)

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
print(len(v[-1]))
```

O/p: 26

Displaying number of characters in the last (python) N characters  
Line (excluding commas)

(412)

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[-1].split(',')
```

k=0

for i in x:

```
 k=k+len(i)
```

```
print(k)
```

O/p:- 22

Counting no. of characters in first word.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
a = v[0].split(',')
```

```
print(len(a[0]))
```

O/p:- 3.

Counting no. of characters in the last word of the  
first line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[0].split(',')
```

```
print(len(x[-1]))
```

O/p:- 5

Counting number of characters in the last word  
of the last line

Python Narayana

(H13)

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[-1].split(' ')
```

```
print(len(x[-1]))
```

O/p: 5

Counting number of characters in the file.

```
a = open('Employees.txt')
```

```
v = len(a.read())
```

```
print(v)
```

O/p: 103.

Counting number of words in the first line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
print(len(v[0].split(',')))
```

O/p: 4.

Counting number of words in the last line.

```
a = open('Employees.txt')
```

```
v = a.readlines()
```

```
x = v[-1].split(',')
```

`print(len(x))`

Python Narragana

(414)

O/p: 5

Counting number of words in the second line.

`a = open('Employees.txt')`

`v = a.readlines()`

`x = v[1].split(',')`

`print(len(x))`

Counting number of words in the file.

~~`a = open('Employees.txt')`~~

`v = a.readlines()`

`x = 0`

`for i in v:`

`k = i.split(',')`

`x = x + len(k)`

`print(x)`

O/p: 18

Counting number of lines in the file

`a = open('Employees.txt')`

`v = a.readlines()`

`print(len(v))`

O/p: 4

Counting number of vowels in the file

```
a = open('Employees.txt')
```

(H15)

```
b = a.read()
```

```
v = "aeiouAEIOU"
```

```
d = dict.fromkeys(v, 0)
```

```
for i in b:
```

```
 if i in d:
```

```
 d[i] = d[i] + 1
```

```
print(d)
```

Output { 'a': 18, 'A': 0, 'E': 6, 'I': 3, 'O': 2, 'U': 0, 'Y': 0, 'K': 6, 'D': 0  
 'E': 0, 'U': 0 }

Make the first word as header and remaining words  
 as members.

```
a = open('Employees.txt', 'r')
```

```
v = a.readlines()
```

```
for i in v:
```

```
x = i.split(',')
```

```
print(x[0], '\n-----'
```

```
for j in range(1, len(x)):
```

```
 print(x[j])
```

O/Ps

tcs

(u16)

Narayana

Venu

bala.

wipro

Suresh

Naresh

Ramesh.

info

rene

amala

Kanya

ramee

cts

Veni

Satya

dwiga

Ramya.

Python Narayana

## Extracting data from pdf files -

python Narayana

417

- python supports extracting data from PDF files also
- we use "pypdf2" module to extract data from pdf file.
- We use 'rb' mode "readbinary" to read data from PDF file

```
import pypdf as pp
a= open('PythonNotes.pdf', 'rb')
b= pp.pdfFileReader(a)
print(b.numpages) #it displays no. of pages in PDF.
c= b.getpage(a) #it gets the page from the list
d= c.extractText() # it extract the text/data
print(d).
from the specific page which
is fetched.
```

- 'a' is a pdf file it reads the file individual pages & saves in list
- "numpages" is a predefined variable to display no of pages in pdf.

→ "pdffile.read()" will read all pages from a PDF file into list format

(UI8)

→ The above eg 'b' is a list which contains all pages of PDF file  $b = [ \boxed{1}, \boxed{2}, \boxed{3} \dots \boxed{199} ]$  pages.

0 1 2 199  
→ "getpage(c)" allows index of specific page & fetches that page. Here in the above eg 'c' contain page no. 10.

In the above eg 'd' contains textual data of page no 10

### Extracting data from CSV file :- (comma separated values)

→ We use csv module to read data from CSV file

### Creating CSV file:-

→ go to / open text file and write the following data employee data

Nani ,	20000 ,	Hyd
Satya ,	10000 ,	bang
renu ,	30000 ,	chennai

- Python Narayana
- Save the file with 'csv' extension on my desktop
  - Copy the file from desktop & paste in python directory.

(H19)

```
import csv
a = open('employee data.csv')
b = csv.reader(a)
c = list(b)
print(c)
```

Output:

```
[['nani', '20000', 'Hyd'], ['satya', '10000', 'bang'], ['renu',
 '30000', 'chennai']]
```

Python Narayana

450

Python Narayana

# Assignment - 3

Python Narayana

(H21)

1. what are the identity operators? And explain each one?
2. what are the membership operators? And explain each one?
3. what is the difference b/w division and floor division? Explain with egs?
4. what are bitwise operators?
5. if  $a=5$  and  $b=7$  then what are the results of  
a    $a \& b =$   
b    $a | b =$   
c    $a \sim b =$

Python Narayana

d     $a >> 2 =$   
e     $a << 2 =$   
f     $b >> 3 =$   
g     $b << 3 =$

Python Narayana

(412)

6. How to check whether specific element is there or not in the given list?

7. How to compare the addresses of two variables?

8. What is for loop? Give one example?

9. What is while loop? Give one example?

Python Narayana

10. Display 10th table by using for Loop? Python Narayana

(H2)

Python Narayana

— 11. How to add first 10 numbers?

12. What is def keyword? And give one example by using def keyword?

DO

13. What is a function definition & function call?

14. What are actual parameters & formal parameters?

15. what are \*args and \*\*kwargs? purpose of each  
Python Narayana  
one?

(424)

16. what is Lambda function? what are we lambda  
function?

17. what are the other functions that we use along  
with lambda functions?

18. what is filter function? Give one example?

19. what is map function? Give one eg?

20. what is reduce function? Give eg?  
python Natayane;;  
425

21. A list has both even and odd numbers, then how to access only even numbers? Explain with eg?

22. A list has both negative and positive numbers, then how to access only positive numbers? Explain with eg?

23. we have two lists, each one has some elements but some elements are matching in two lists, then how to check what are the matching elements? Explain with one eg?

24. A List has so many int type values, how to access greatest element among all? Explain with one eg?

(426)

25. How to find square of all elements in the list?

26. How add all elements from both sets as per their index positions?

27. Write a python program for the following  
Scenario One MNC is soon conducting interview,  
they asked Candidates to fill the online form.

a. Enter your name (this student's name should appear in every display statement) U27

b. qualification must be B.tech / BE

i. if he enters other qualification then display "you are not eligible".

ii. if he enters B.tech or B.E then check passedout year.

c. passedout year must be either 2016 or 2017

i. if he enters 2015 or below then display "you are not fresher".

ii. If he enters 2018 or above then display "you entered invalid year"

Q 28 you need to ask user to enter any three values and display him the maximum value out of those three values?

If the user values like a, b and c then result must like.

a is greater than b and c or

b is greater than a and c or

c is greater than a and b

Q9. you need to ask user to enter his required product name, if the product name length is more than 5 character then perform the following operations on the entered product name else display "you entered invalid product name". the operations are.

- a. find the length of given string
- b. Display first character in upper case.
- c. Check whether that product name contains "x" or not.
- d. Display given product name in reverse order
- e. Display given string in ascending order like from a to z
- f. Display given string in descending order like from z to a.

30. the given string "NaRaYaNa" then how to display "NANAYANA"

Python Narayana

Q) Given string "NARAYANA" then how to separate as 'N', 'A', 'Y', 'N'.

(129)

32. What is the output for the following.

```
def a1(a,*b):
```

```
 print(a,b)
```

```
a1(1,2,3)
```

Output

O  
D

33. What is the output for the following

```
def av(a,b=10,*c,**d):
```

```
 print(a,b,c,d)
```

```
av(1,2,a1=3, c='f', d=4, r=5)
```

O/p:

Python Narayana

34. What is the o/p of the following

```
def av(a,b=10,*c,**d):
```

```
 print(a,b,c,d)
```

```
av(1,2,'a1', 3, 'f', 4, 5)
```

output:-

python Narayana

(43)

35. What is the difference b/w readline() & readlines()?

file name is "CustomerData.txt".

- 1. John, Software, 30000, Hyd, Python.
- 2. Viyaan, Data Analyst, 40000 Bang, Oracle
- 3. Renu, Manager, 50000, mumbai, Python.

36 Fetch Last word in the file?

37 Display the salary of viyaan?

Retrieve the Second line data from the file?

38

39 Count number a's & e's from the file?

Python Narayana

## python Navayana

40 How to count number rows in the file?

(431)

41 How to count number of words in the files

42 How to retrieve all employee names from the  
file?

Python Navayana

## Modules:-

python Narayana.

(432)

If we need to reuse the code then we can define functions, but if we need to reuse number of functions then we have to go for Modules.

- A module is nothing but a file with extension .py which may contains methods, Variables & also classes.
- Modules are python.py files that consist of Python code. Any python file can be referenced as a module if a python file called "mymodule.py" has the module name of mymodule that can be imported into other python files offered on the python command line interpreter.

Modules are three types:-

1. Standard Modules
2. User defined Modules
3. 3<sup>rd</sup> party Modules.

Standard Modules: these modules are ~~really~~ already defined and kept in python software. So when we install python then automatically these standard modules will install in our Machine.

(H33)  
Like math, calendar, os, sys, logging, threading modules... .

User defined Modules: these modules are defined by users as per their requirement. So here user defined module is nothing but the .py file which contains methods, variables and also classes.

3rd party Modules: these modules are already defined by some other people and kept in internet so we can download and install in our machines by using "PIP" (python package installer).

→ PIP is a package management system used to install and manage software packages written in python.

like pymysql, cx-oracle

→ In python, modules are accessed by using import statement.

→ When our current file is needed to use the code which is already existed in other files we can import that file (module).

→ When python imports a module called "Mymodule" for example, the interpreter will first search for a built-in module called Mymodule. If a built-in module is not found, the python interpreter will then search for the a file named Mymodule.py in a list of directories that it receives from the sys.path variables.

We can import module in three different ways.

1. import < module\_name >
  2. from < module\_name > import < method\_name >
  3. from < module\_name > import \*
1. import < module\_name > - this way of importing module will import all methods which are in that specified module.

Eg: import math.

Python Notes

Here this import statement will import all methods which are available in math module. We may use all methods or may use required methods as per business requirements

(135)

## 2. From < module-name > import < method-name >

Here, this import statement will import a particular method from that module which is specified in the import statement.

We can't use other methods which are available in that module as we specified particular method name in the import statement.

Note: We need to avoid this statement type of modules as they may cause name clashes in the current python file.

## 3. from < module-name > import <\*> ;

Here, this import statement will import all methods from the specified module and also it will import all other modules which are imported into that specific module-name

Python Narayana

## Standard Modules :-

Python Norayane  
436

1. sys module
2. calendar module
3. Datetime module.
4. Math module
5. Os module & so many.

## Math modules:-

```
>>> import math
>>> math.ceil(10.9) Norayane
>>> math.ceil(10.1) 11
>>> math.floor(10.9) 10
>>> math.floor(10.1) 10
>>> math.fabs(10) 10.0
>>> math.fabs(-10) 10.0
>>> math.fabs(-9) 9.0

>>> math.factorial(5) 120
>>> math.factorial(10) 3628800
>>> math.factorial(3) 6.

>>> math.fmod(10, 2) 0.0
>>> math.fmod(10, 3) 1.0
>>> math.fmod(50, 3) 2.0
```

>>> math.fmod(19,3)	1.0	python Narayana
>>> math.fmod(17,3)	2.0	
		(H37)
>>> math.fsum([1,2,4])	7.0	
>>> math.fsum([1,2.5,4.5])	8.0	
>>> math.modf(10)	(0.0, 10.0)	
>>> math.modf(7)	(0.0, 7.0)	
>>> math.modf(3)	(0.0, 3.0)	
>>> math.trunc(10.89)	10	
>>> math.trunc(-7.8734)	7	
>>> math.trunc(0.34)	0	
>>> math.trunc(-3.34)	-3.	
>>> math.exp(2)	7.38905609893065	
>>> math.exp(3)	20.08553923187688	
>>> math.pow(2,3)	8.0	
>>> math.pow(2,-3)	0.125	
>>> math.pow(-2,-3)	-0.125	
>>> math.pow(-2,3)	-8.0	
>>> math.pow(3,3)	27.0	
>>> math.sqrt(100)	10.0	
>>> math.sqrt(36)	6.0	
>>> math.sqrt(49)	7.0	

Python Narayana

```

>>> math.sin(1) Python Narayana
0.8414709848018965
>>> math.sin(100) 938
-0.5063656411091588
>>> math.sin(0.5) 0.479425538604208

>>> math.cos(1) 0.5403023058681398
>>> math.cos(100)
0.8623188722876839
>>> math.cos(0.5) 0.8775825618903728

>>> math.tan(1) 1.5574077846549023
>>> math.tan(100)
-0.5872139151569291
>>> math.tan(0.5) 0.5463024898437905

>>> math.pi 3.141592653589793
>>> math.e 2.718281828459045

```

## Calender Module

Calender.day\_name :- An array that represents the days of the week in the current Locale

### 1. Displaying all week names One by One

```

import calendar as c
for i in c.day_name
 print(i)

```

Python Narayana

Output:

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Sunday

- Q 2. Display week name character by character as per the week index number given by user:

```
import calendar as c
a = input('enter number from 0 to 6: ')
for i in c.day_name[a]:
 print(i)
```

Output:

enter number from 0 to 6:

T  
u  
e  
s  
.d  
a  
y

Python Narayana

3. Display first character of each day.

```
import calendar as c
for i in c.day_name:
 print(i[0])
```

(40)

Output:

M  
T  
W  
T  
F  
S  
S.

4. Display all days except 'day' part in each name?

Import calendar as c.

```
for i in c.day_name:
 print(i[:-3])
```

# or

```
for i in c.day_name:
 print(i.replace('day', ''))
```

Output:

Mon  
Tues  
Wednes

Thurs

Fri

Satur

Sun.

5. Display only 'day' part from each day.

```
import calendar as c
```

```
for i in c.day_name:
 print(i[-3:])
```

Output:

day

day

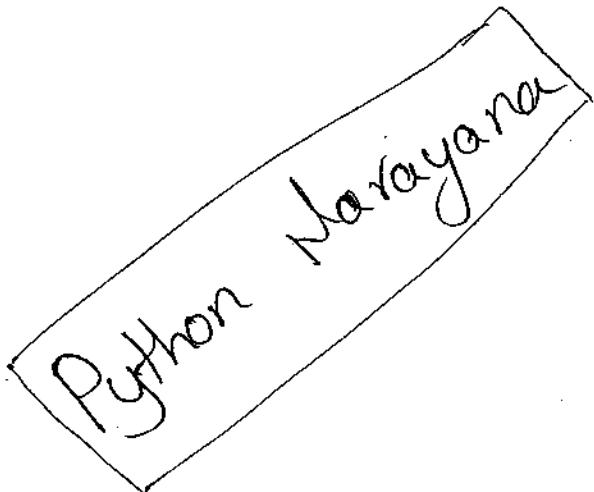
day

day

day

day

day



6. Display any day as per user requirement, if he enters invalid index number then display 'invalid' index number?

```
import calendar as c
```

```
num = int(input('Enter index number:'))
```

```
a = list(c.day_name)
```

if num >= 0 and num <= 6 :

Python Narayana

    print(a[num])

(442)

else:

    print('invalid index number')

Output: 1

Enter index number : 2

Wednesday

Output: 2

Enter index number : -1

invalid index number

Output: 3

Enter index number : 10

invalid index number.

Python  
Narayana

Output: 4

Enter index number : 6

Sunday.

Calendar.day\_abbr: Any array that represents the abbreviated days of the week in the current locale.

Display all week abbreviations:

```
import calendar as c
for i in c.day_abbrs:
 print(i)
```

python Narayana

(443)

### Output:

Mon

Tue

Wed

Thu

Fri

Sat

Sun.

Calendar.month\_name: An array that represents the months of the year in the current locale. This follows normal convention of january being month number 1, so it has a length 13 and month\_name[0] is the empty string.

```
import calendar as c
for i in c.month_name:
 print(i)
```

### Output:

January

February October

March November

April December

May

June

July

August

September

Python Narayana

Calendar.month\_abbr: An array that represents the abbreviated months of the year in the current locale. python Narayana  
This follows normal conventions of January being month number 1, so it has a length of 13 and month abbr[0] is the empty string.

```
import Calendar as c.
```

```
for i in c.month_abbr:
 print(i).
```

### Output

Jan  
Feb  
Mar  
Apr  
May  
Jun  
Jul  
Aug  
Sep  
Oct  
Nov  
Dec.

Python Narayana

Calendar.monthrange(year, month): Returns weekday of first day of the month and number of days in month, for the specified year & month.

```
import calendar as c
```

Python Narayana

(445)

```
print(c.monthrange(2018, 4))
```

```
print(c.monthrange(2017, 2))
```

```
print(c.monthrange(2011, 9))
```

Output:

(6, 30)

(2, 28)

(3, 30)

Calendar.isleap(year): Returns True if year is a

leap year, otherwise False

```
import calendar as c
```

```
print(c.isleap(2019))
```

```
print(c.isleap(2010))
```

```
print(c.isleap(2020))
```

Output:

false

false

true.

Calendar.leapdays(year1, year2): returns the number

of leap years in the range from  $y_1$  to  $y_2$

where  $y_1$  and  $y_2$  are years.

```
import calendar as c
```

```
print(c.leapdays(2010, 2020))
```

```
print(c.leapdays(2000, 2017))
print(c.leapdays(1988, 2018))
```

Python Narayana

446

Output

2

5

8

Calendar.weekday(year, month, day): Returns the day of the week (0 is Monday) for year (1970---) month (1-12), day (1-31)

```
import calendar as c
```

```
print(c.weekday(2018, 4, 18))
```

```
print(c.weekday(1988, 3, 10))
```

```
print(c.weekday(1989, 8, 15))
```

Output

2

3

1

Calendar.weekheader(n): Returns a header containing abbreviated weekday names. n specifies the width in characters for one weekday.

```
import calendar as c
```

```
print(c.weekheader(1))
```

```
print(c.weekheader(2))
```

```
print(c.weekheader(3))
```

Output:

M T W T F S S

Mo Tu We Th Fr Sa Su

Mon Tue Wed Thu Fri Sat Sun.

Calendar.monthcalendar(year, month) Returns a

matrix representing a month's calendar. Each row  
represents a week; days outside of the month are  
represented by zeros. Each week begins with Monday  
unless set by `setfirstweekday()`

```
import calendar as c
```

```
print(c.monthcalendar(2018, 4))
```

```
print(c.monthcalendar(2015, 6))
```

Output:

```
[[0, 0, 0, 0, 0, 1], [2, 3, 4, 5, 6, 7, 8], [9, 10, 11, 12, 13, 14, 15],
[16, 17, 18, 19, 20, 21, 22], [23, 24, 25, 26, 27, 28, 29], [30, 0, 0,
0, 0, 0]].
```

```
[[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14], [15, 16, 17, 18, 19, 20, 21],
[22, 23, 24, 25, 26, 27, 28], [29, 30, 0, 0, 0, 0, 0]]
```

## Python Narayana

### Calendar.Calendar(year[, w[, l[c]]]):— Returns

(498)

a 3-column Calendar for an entire year as a multi-line string using the `formatyear()` of the `TextCalendar` class.

```
import calendar as cl
```

```
print(cl.Calendar(2018)).
```

#### Output:

2018

January

Mo	Tue	We	Th	Fri	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

February

Mo	Tu	We	Th	Fri	Sa	Su
1	2	3	4			
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

March

Mo	Tu	We	Th	Fri	Sa	Su
1	2	3	4			
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

April

Mo	Tu	We	Th	Fri	Sa	Su
1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

May

Mo	Tu	We	Th	Fri	Sa	Su
1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

June

Mo	Tu	We	Th	Fri	Sa	Su
1	2	3				
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

# python Narayana

July

Mo	Tu	We	Th	Fr	Sa	Su
				1		
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

August

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

September  
30

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

October

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

November

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

December

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

## sys Module

### Eg:

```
import sys
print ("the command line arguments are:")
for i in sys.argv:
 print(i).
```

### Output:

The Command line arguments are:

C:/users/Narayana/AppData/Local/programs/python/  
python 36-32/Sysmod.py.

Eg.2

python Narayana

(450)

import sys.

print ('\\n\\n the PYTHONPATH is', sys.path, '\\n').

Output

The PYTHONPATH is [ 'C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32',

'C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32\\Lib\\idlelib',

'C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32\\Python 36.zip',

'C:\\users\\Narayana\\AppData\\Local\\programs\\python\\Python 36-32\\DLLs',

'C:\\users\\Narayana\\AppData\\Local\\programs\\Python\\python 36-32\\lib',

'C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32',

'C:\\users\\Narayana\\AppData\\Local\\programs\\python\\python 36-32\\Lib\\site-packages'].

# Python Narayana

(es)

Eg:3

```
import sys
user_input = sys.stdin.readline()
print ("input:" + user_input)
```

Eg:4

```
import sys
print (sys.copyright).
```

Output:

Copyright(c) 2001-2018 python software foundation

All Rights Reserved.

Copyright(c) 2000 BeOpen.com.

All Rights Reserved.

Copyright(c) 1995-2001 Corporation for National Research Initiatives.

All Rights Reserved.

Copyright(c) 1993-1995 Stichting Mathematisch Centrum, Amsterdam.

All Rights Reserved.

Eg:5

```
import sys.
print ("Python Narayana")
```

sys.exit(1)  
print("Hello")

python Narayana

452

Output:-

python Narayana

Eg:6:-

```
import sys
variable = "python Narayana"
print(sys.getrefcount(0))
print(sys.getrefcount(variable))
print(sys.getrefcount(None))
```

Output:-

955

3

6874

Python

Narayana

Datetime module:-

Eg:1 import datetime  
dt = datetime.date(2018, 8, 26)  
print(dt)  
print(dt.year)  
print(dt.month)  
print(dt.day)

Output:-

2018-08-26

2018

8

26

Eq:2

Python Narayana

```

import datetime
dt1 = datetime.time(10, 11, 12, 236587)
print(dt1)
print(dt1.hour)
print(dt1.minute)
print(dt1.second)
print(dt1.microsecond)

```

(H63)

Output:

10:11:12. 236587

10.  
11  
12  
236587

Eq:3

```

import datetime
dt2 = datetime.datetime(2018, 5, 28, 12, 23, 34, 567893)
print(dt2)
print(dt2.year)
print(dt2.month)
print(dt2.day)
print(dt2.hour)
print(dt2.minute)
print(dt2.second)
print(dt2.microsecond)

```

Python Narayana

Output:-

python Narayana

2018-05-28 12:23:34.567893.

(454)

2018

5

28

12

23

34

567893.

## Working with random module-

This module defines several functions to generate random numbers. We can these functions while developing games in cryptography and to generate random numbers on fly for authentication.

### 1. random() function:-

This function always generate some float value b/w 0 and 1 (not exclusive)  $0 < x < 1$

Eg:-

1. from random import \*

2. for i in range(10):

3. print(random())

4.

5. Output.

Python Narayana

6. 0.45786856093202056
7. 0.6584325233197768
8. 0.15444034016553587
9. 0.18357427005232201
10. 0.1330257265904884
11. 0.9291139798071045
12. 0.658741191891783
13. 0.8901649834619002
14. 0.25540891083913053
15. 0.7290504335962871

## 2. randint() function :-

To generate random integer b/w two given numbers  
(inclusive)

Eg:-

1. from random import \*
2. for i in range(10):
3. print(randint(1,100)) # generate random int value b/w 1 and 100 (inclusive)
- 4.

## 5. Output

6. 51
7. 44
8. 39
9. 70
10. 49
11. 74

Python Narayana

12. 52

13. 10

14. 40

15. 8.

### 3. Uniform()

It returns random float values b/w 2 given numbers  
(not inclusive)

Eg:- 1. from random import \*

2. for i in range(10):

3. print(uniform(1,10))

4.

5. Output

6. 9.787695398230332

7. 6.81102218793548

8. 8.068672144877329

9. 8.5649916351239834

10. 6.363511674803802

11. 2.176137584071641

12. 4.822067939432386

13. 6.0801725149678445

14. 7.508457735544763

15. 1.9982221862917555

random()  $\Rightarrow$  in between 0 and 1 (not inclusive)

randomint(x,y)  $\Rightarrow$  in between x and y (inclusive)

$\text{Uniform}(x,y) \Rightarrow$  in between X and Y (not inclusive)

#### 4. $\text{randrange}([\text{start}], \text{stop}, [\text{step}])$ :-

(457)

returns a random number from range

$\text{start} \leq x < \text{stop}$

$\text{start}$  argument is optional & default value is 0

$\text{step}$  argument is optional & default value is 1

$\text{randrange}(10) \rightarrow$  generates a number from 0 to 9.

$\text{randrange}(1, 11) \rightarrow$  generates a number from 1 to 10.

$\text{randrange}(1, 11, 2) \rightarrow$  generates a number from 1, 3, 5, 7, 9.

Eg:-

```
1. from random import *
2. for i in range(10):
3. print(randrange(10))
```

4.

5. output

6. 9

7. 4

8. 0

9. 2

10. 9

11. 4

12. 8

13. 9

14. 5

15. 9

Python Narayana

python Narayana

# python Narayana

(458)

Eg:1 from random import \*  
for i in range(10):  
    print (randrange (1,11))  
4  
5 output  
6. 2  
7. 2  
8. 8  
9. 10  
10. 3  
11. 5  
12. 9  
13. 1  
14. 6  
15. 3

Python Narayana

## Eg:3

1. from random import \*  
2. for i in range(10):  
    print (range (1,11,2))  
4.  
5. output  
6. 1  
7. 3  
8. 9  
9. 5  
10. 7

# Python Narayana

(459)

- 11. 1
- 12. 1
- 13. 1
- 14. 7
- 15. 3

## 5) choice() function

- It won't return random number.
- It will return a random object from the given list or tuple.

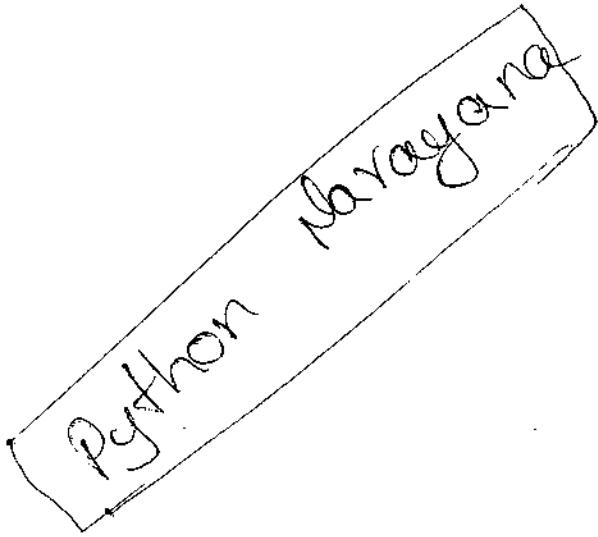
Ex:

```
1. from random import *
2. List = ["Narayana", "Santhosh", "Phanalekshmi", "Shiva"].
3. for i in range(10):
4. print(choice(List))
```

5

6. Output:

- 7. Narayana
- 8. Santhosh
- 9. Narayana
- 10. Phanalekshmi
- 11. Shiva
- 12. Narayana



Python Narayana

(460)

13. Shiva
14. Dharalakshmi
15. Shiva

Python Narayana

## \* Exception Handling \*

(NGI)

- An Exception is a run time error that happens during the execution of program.
- An Exception is an error that happens during the execution of a program.
- Python raises an exception whenever it tries to execute invalid code.
- Error handling is generally resolved by saving the state of execution at the moment the error occurred and interrupting the normal flow of the program to execute a special function or piece of code, which is known as the exception handler. Depending on the kind of error ("division by zero", "file open error" & so on) which has occurred the error handler can "fix" the problem and the program can be continued afterwards with the previously saved data.
- Exceptions handling in python is very similar to java. The code which harbours the risks of an exception is embedded in a ~~try block~~ try block. But whereas in java exceptions are caught by catch clauses.

We have statements introduced by an except keyword in python. It's possible to "create custom made" exceptions: with the raise statement it's possible to force a specified exception to occur.

generally any programming language supports two types of errors.

1. Syntax errors.

2. Runtime errors.

Syntax Errors: These errors will raise at development time and generally these errors will raise because of invalid syntax, so these are called Syntax errors.

As these errors will occur at development time, the developer is responsible for the errors, developer needs to

Eg:-

a=10

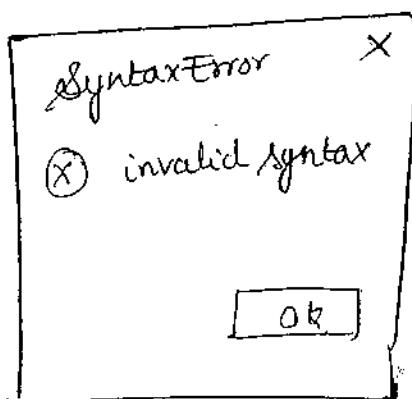
b=5

if a>b:

    print(a)

else:

    print(b)

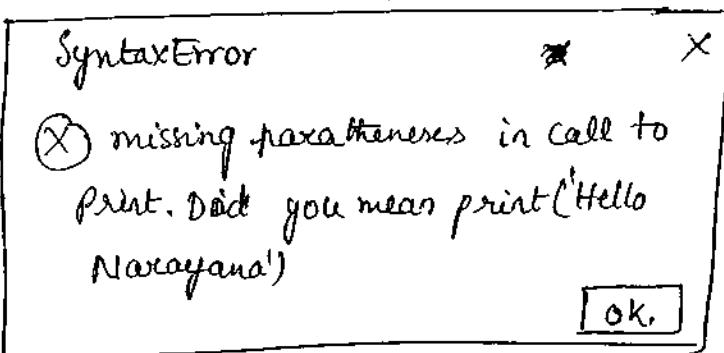


Eg:2

python . Narayana :-)

(HG3)

print "Hello Narayana"



Eg:3

a=10

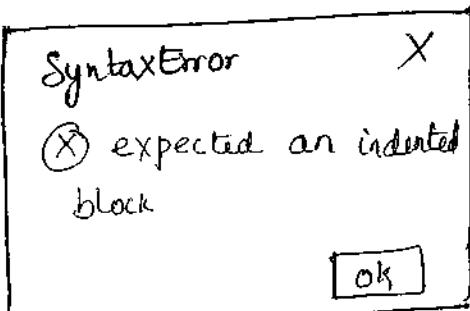
b=5

if a>b:

    print(a)

else:

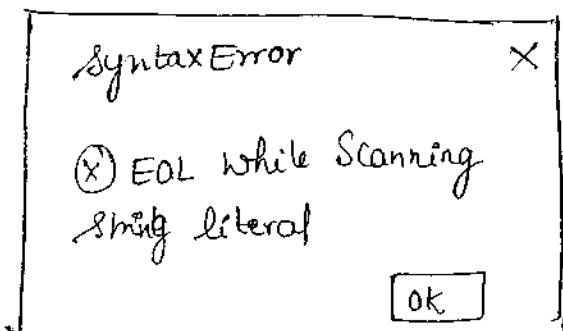
    print(b)



Python  
Narayana

Eg:4

print ('Hello Narayana')



## Runtime Errors these are also called Exceptions Python Narayana

when the program is executing, if something goes wrong 464 because of end user input or programming logic or memory problems etc then we call them runtime errors.

→ An exception is nothing but an unwanted (or) unexpected block which disturb the normal execution flow of program.

these exceptions are two types.

1. System defined exceptions.

2. Userdefined exceptions.

### System defined exceptions

These exceptions are defined by system so these are called system defined or predefined exceptions. Some of system defined exceptions are.

ZeroDivisionError.

NameError

IndexError

TypeError

ValueError

KeyError

Python  
Narayana

## Python Narayana

(H65)

KeyboardInterrupt Error

sleepingError

FileNotFoundException

ModuleNotFoundError

LookupError

IndentationError

NotImplementedError

Note Exception handling does not mean repairing exception, we have do define an alternative way to continue rest of the program Normally.

python supports handling the exceptions by using

Try and except block

Ex:1

```
a = int(input('Enter first value'))
```

```
b = int(input('Enter second value'))
```

```
c = a/b
```

```
print ('The result is', c)
```

Output

```
Enter first value : 10
```

```
Enter second value : 4
```

```
The result is 2.5.
```

Python Narayana

python Narayana

466

Enter First Value : 10.5

Enter Second Value : 25

the result is 4.2

In this example, we don't have any exception, so its not required to use try and except block, if user gives denominator value 0 then it will throw an exception "ZeroDivisionError"

Eg:

```
a = eval(input('Enter first value:'))
```

```
b = eval(input('Enter second value:'))
```

```
c = a/b
```

```
print ('The result is', c)
```

Output:

Enter first value : 10

Enter second value : 0

Narayana  
Python

Traceback (most recent call last):

File "C:/Users/Narayana/AppData/Local/Programs/Python/

python36-32/exceptionsfile.py", line 3, in <module>

c = a/b

ZeroDivisionError: division by zero.

## Python Narayana

In order to handle errors, you can set up exception handling blocks in your code. The keywords try and except are used to catch exceptions. When an error occurs within the tryblock, python looks for a matching except block to handle it. If there is one, execution jumps there.

```
a = eval(input('Enter first value:'))
b = eval(input('Enter Second value:'))
```

try:

```
c = a/b
```

```
 print('The result is', c)
```

except:

```
 print('Error Occurred')
```

Output:

```
Enter first value: 10
```

```
Enter Second Value: 0
```

```
Error Occurred.
```

If user gives the value of b is '0' then automatically try block will raise ZeroDivisionError exception.

immediately the exception will be thrown to the except block and it will display 'Error Occurred' message.

→ If user give the value of  $b$  more than '0', then try block will execute and return output results. Here except block will not execute.

(468)

→ This except block is called default except block, because this except block will return same message for all types of exceptions which are thrown by try block.

Eg:2  $a = \text{eval}(\text{input}(\text{'Enter first values'}))$   
 $b = \text{eval}(\text{input}(\text{'Enter second Value: '}))$

try:

$c = a/b$

print('the result is'; c)

except

print('Error occurred')

Output

Enter first Value: 10

Enter Second Value: 5

Error occurred

In the example the try block has thrown TypeError for the except block, but the except block has returned same 'Error occurred'.

Eg:3

```
a = eval(input('Enter first value:'))
```

(H69)

```
b = eval(input('Enter Second value:'))
```

```
try:
```

```
 c = a/b
```

```
 print('The result is:', c)
```

```
except:
```

```
 print('Error Occurred')
```

Output:-

```
Enter first value: 10
```

```
Enter Second value: 5
```

```
Error Occurred.
```

Python Narayana

In the example the try block has thrown NameError to the except block, but except block has return same 'Error Occurred'.

### Multiple Except blocks

We can use different except blocks to handle different exceptions separately.

A try statement may have more than one except clause for different exceptions. But at most one except clause will be executed.

Eg:- Using separate except block for handling

ZeroDivisionError.

Python Narayana

```
a = eval(input('Enter first value'))
b = eval(input('Enter second values'))
```

(170)

Try:

```
c = a/b
print ('the result is', c)
except ZeroDivisionError:
 print('please enter non-zero for denominator')
except:
 print('Error Occurred')
```

Output: 1

Enter first value: 10

Enter Second value: 0

please enter non-zero for denominator.

Output: 2

Enter First Value : 10

Enter Second value : 3

the result is 3.3333333333335

Output: 3

Enter first value : 10

Enter Second value : 'g'

Error Occurred

Python Narayana

## Python Narration

In the above example, we have used two except blocks  
the earlier one is for handling ~~ZeroDivisionError~~<sup>(17)</sup> and  
later one to handle remaining all other exceptions.

→ The default except block must be last except block  
when we have multiple blocks in the program.

Eg:- Using multiple except blocks for handling multiple  
exceptions separately.

```
a = eval(input('Enter first value:'))
```

```
b = eval(input('Enter second value:'))
```

try :

```
c = a/b
```

```
 print ('The result is', c)
```

```
except ZeroDivisionError:
```

```
 print ('Please enter non-zero for denominator')
```

```
except NameError:
```

```
 print ('Please use defined names only')
```

```
except TypeError:
```

```
 print ('Please use proper types only')
```

```
except s:
```

```
 print ('Error occurred')
```

Python

Narration

(472)

Output:

Enter first value : 10

Enter second value : 0

please enter non-zero for denominator.

Eg.1

try:

fh = open ("testfile", "w")

fh.write ("This is my test file for exception handling!")

except IOError:

print "Error: Can't find file or read data."

else:

print "Written content in the file successfully!"

fh.close ()

Output:

written content in the file successfully.

a = eval(input('Enter first value: '))

b = eval(input('Enter second value: '))

try:

c = a/b

print ('please the result is', c)

except ZeroDivisionError:

print ('please enter non-zero for denominator')

except zeroDivisionError:

(H73)

```

print('Please use defined names only')
except TypeError:
 print('please use proper types only')
except:
 print('Error Occurred')

```

Output:-

Enter first Value : 10

Enter Second Value : 2

please use defined names only.

2). a = eval(input('Enter first value:'))
b = eval(input('Enter second value:'))

try:

c = a/b

print('The result is', c)

except ZeroDivisionError:

print('please enter non-zero for denominator')

except NameError:

print('please use defined names only')

except TypeError:

print('please use proper types only')

except:

print('Error Occurred').

## Output:-

python Narayana

(U74)

Enter first value : 10

Enter second value : '5'.

Please use proper types only.

## Else blocks

The try...except statement has an optional else clause. An else block has to be positioned after all the except clauses. An else clause will be executed if the try clause doesn't raise an exception.

```
a = eval(input('Enter first Value:'))
```

```
b = eval(input('Enter second Value:'))
```

```
try:
```

```
c = a/b
```

```
print('the result is', c)
```

```
except ZeroDivisionError:
```

```
 print('please enter non-zero for denominator')
```

```
except NameError:
```

```
 print('please use defined names only')
```

```
except TypeError:
```

```
 print('please use proper types only')
```

```
except
```

```
 print('Error Occured')
```

```
else:
```

```
 print('program executed successfully').
```

Output:1

Enter first value : 10

Enter second value : 2

the result is 5.0.

program executed successfully.

Output:2

Enter first value : 10

Enter second value : 'a'

please use proper types only.

Output:3

Enter first value : 10

Enter second value : 0

please enter non-zero for denominator.

Finally:

until python 2.5, the try statement came in two flavours. you could use a finally block to ensure that code is always executed, or one or more 'except' block to catch specific exceptions. you couldn't combine both except blocks and a finally block, because generating, the right byte code for the combined version, was complicated and it wasn't clear what the semantics of the combined st should be.

Eg:

```
a = eval(input('Enter first value:'))
b = eval(input('Enter second value:'))
```

(476)

try :

c = a/b

print ('the result is', c)

except ZeroDivisionError :

print ('Please enter non-zero for denominator')

except NameError :

print ('please use defined names only')

except TypeError :

print ('please use proper types only')

except :

print ('Error Occurred')

else :

print ('Program executed successfully')

finally :

print ('Thankyou')

Output

Enter first value : 10

Enter Second value : 4

The result is 2.5

Python Narayana

program executed successfully.

Thank you.

### Output:2

Enter first value : 10

Enter second value : '5'.

Please use proper types only  
thankyou.

### Output:3

Enter first value : 10

Enter second value : 0

Please enter non-zero for denominator  
thankyou.

Assert :- it is used to check the value, if it is satisfied  
the condition then it will start the executing the  
flow.

→ If it is not satisfied the condition, then will return  
AssertionError exception.

→ The assert statement is intended for debugging statements  
It can be seen as an abbreviated notation for a  
conditional raise statement, i.e., an exception is only

python Narayana

raised, if a certain condition is not True.  
without using the assert statement, we can formulate  
it like this in python.

(478)

```
if not <Some-test>:
 raise AssertionError(<message>)
```

Eg:

```
a = eval(input('Enter first value:'))
b = eval(input('Enter second value:'))
```

```
assert(b>0)
```

try:

```
c = a/b
```

```
print('The result is', c)
```

except ZeroDivisionError:

```
 print('Please enter non-zero for denominator')
```

except NameError:

```
 print('Please use proper types only')
```

except:

```
 print('Error occurred')
```

else:

```
 print('Program executed successfully')
```

finally:

```
 print('Thankyou')
```

Python Narayana

python Narayana

479

Output-1:

Enter first value : 10

Enter Second value : 0

Traceback (most recent call last):

file "C:/users/Narayana/AppData/Local/programs/python  
/python 36-32/exceptionsfile.py", line 3,  
in <module>  
assert (b>0)

AssertionError.

Output-2

Enter first value : 10

Enter Second value : 5

the result is 2.0

Program executed successfully

thank you

Python Narayana

Python Narayana

## \* User defined Exceptions

Python new chapter

(480)

Sometimes user can also define exceptions to indicate something is going wrong, those are called Customized exceptions or programmatic exceptions.

Developers can write their exceptions as per their requirements, so they are called User defined Exceptions.

Exceptions need to be derived from the exception class, either directly or indirectly.

We use "raise Keyword" to raise user defined exceptions.

Eg:-

Class TooYoungException (Exception):

def \_\_init\_\_(self, arg):

Self.arg = arg.

age = int(input('please enter your age:'))

if age <= 20:

    raise TooYoungException('you are too young, so wait for sometime!')

elif age >= 40:

    raise TOOOldException('you are too old, so don't get marry!')

python Narayana:-

(K8)

else:

    print('Now you are', 'age', 'so you can happily marry')

Output:1

please enter your age : 25

Now you are 25. So you can happily marry.

Output:2

please enter your age : 10

TOOYoungException: you are too young, so wait for some time.

Output:3

please enter your age : 55

TOOOldException: you are too old, so don't get marry.

Eg:2

class LessThanZeroException (Exception):

    def \_\_init\_\_(self, arg):

        self.arg = arg.

class MoreThanHundredException (Exception):

    def \_\_init\_\_(self, arg):

        self.arg = arg.

per = int(input('please Enter your Percentage:'))

if per < 0:

Python Narayana

raise LessThanZeroException ('you entered less than 0 marks')

(482)

elif per > 100:

raise MoreThanHundredException ('you entered more than 100 marks')

else:

if per < 35:

print('you are failed')

elif per >= 35 and per < 50:

print('you got 3rd class')

elif per >= 50 and per < 60:

print('you got 2nd class')

elif per > 60 and per < 75:

print('you got 1st class').

else:

print('you got distinction')

Output:1:

please enter your percentage : 10

you are failed.

Python Narayana

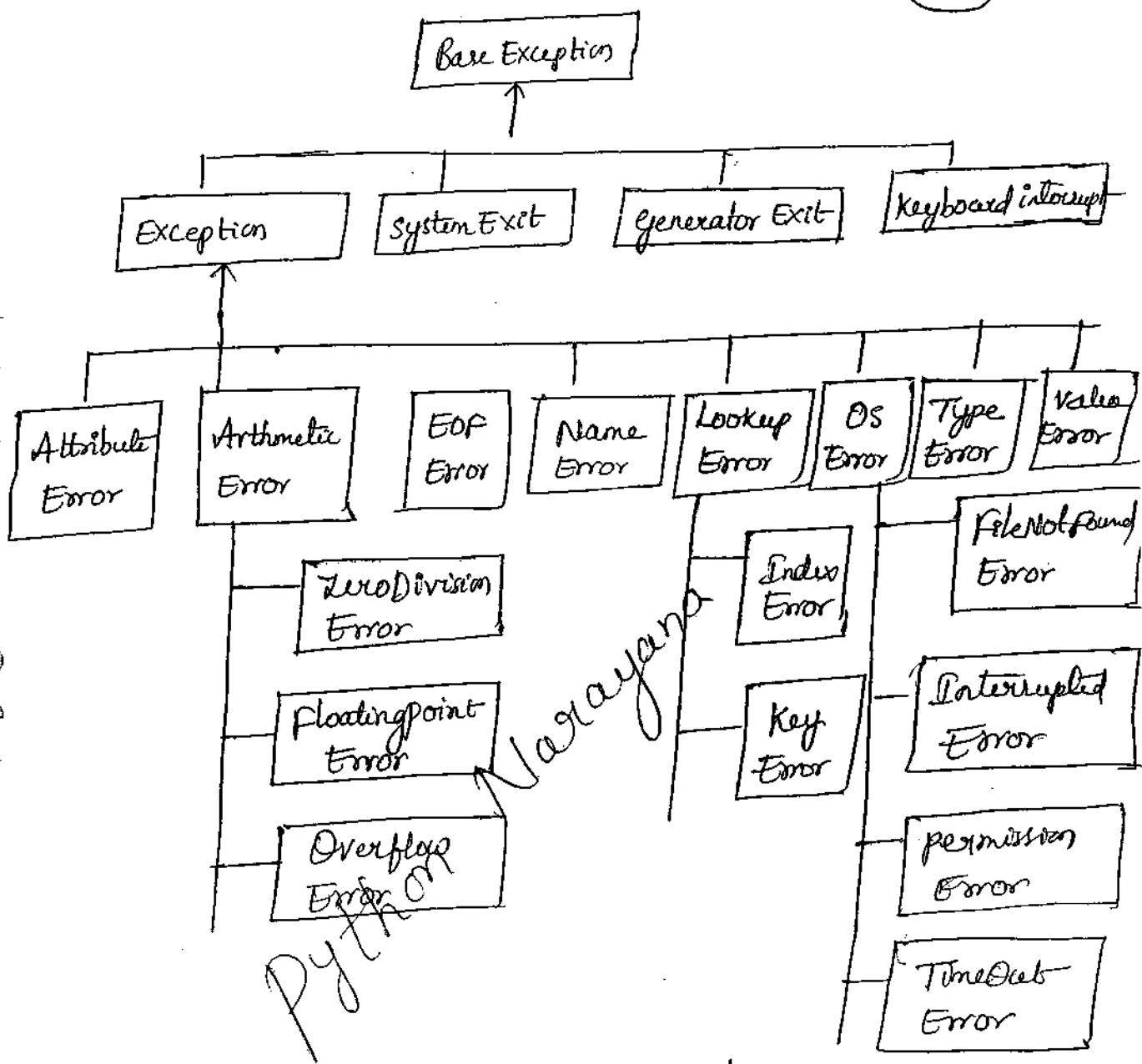
Output:2:

please Enter your percentage : 50

You got 2nd Class.

# \* Python's Exception Hierarchy \*

U83



→ Every Exception in python is a class.  
All exception classes are child classes of `BaseException`.  
i.e., every exception class extends `BaseException`, either  
directly or indirectly. Hence `BaseException` acts as  
root for python Exception Hierarchy.

python Narayana

Most of the times being a programmer we have, to concentrate exception & its child classes.

## Customized Exception Handling by Using try-except

- It is highly recommended to handle exceptions.
- The code which may raise exception is called risky code & we have to take risky code inside try block. the corresponding handling code we have to take inside except block.

try:

Risky code  
except XXX:

Handling code/ Alternative code

Without try-except

1. print("stmt-1")
2. print(10/0)
3. print("stmt-3")
- 4.
5. Output
6. stmt-1
7. ZeroDivisionError: division by zero.

Python

Nagayana

Abnormal termination / Non- Graceful Termination.

with try-except:-

(H85)

```
1. print("stmt-1")
2. try:
3. print(10/0)
4. except ZeroDivisionError:
5. print(10/2)
6. print("stmt-3")
7.
8. Output
9. Stmt-1
10. 5.0
11. Stmt-3.
```

Python Narayana

Normal termination / Graceful Termination.

Control flow in try-except:-

try:

Stmt-1

Stmt-2

Stmt-3.

except XXX:

Stmt-4

Stmt-5.

Case 1 If there is no exception.

1, 2, 3, 5 and Normal Termination.

Case:2 If an exception raised at stmt-2 and corresponding except block matched 1,4,5 Normal Termination.

(486)

Case:3 If an exception raised at stmt-2 & corresponding except block not matched 1, Abnormal Termination

Case:4 If an exception raised at stmt-4 or stmt-5 then it always abnormal termination.

### Conclusions:-

1. within the try block if anywhere exception raised then rest of the try block won't be executed even though we handled that exception. Hence we have to take only risk code inside try block & length of the try block should be less as possible
2. In addition to try block, there may be a chance of raising exceptions inside except and finally block also.
3. If any statement which is not part of try block raises an exception then it is always abnormal termination.

How to print exception information:-

try:

1. print (10/0)

python Narayan.

Python Narayana

2. except zeroDivisionError as msg:

3. print ("exception raised and its")

u87

4

5 Output exception raised & its description is division  
by zero.

try with multiple except blocks

The way of handling exception is varied from exception to exception. Here for every exception type

Hence for every exception type a separate except block we have to provide. i.e., try with multiple except blocks is possible and recommended to use

Eg:-

try:

=====

except zeroDivisionError:

perform alternative

arithmetic operations.

except fileNotFoundErr:

use Local file instead of remote file.

If try with multiple except blocks available then based on raised exception the corresponding except block will be executed.

# Python Narayana

488

Eg:- 1. try:  
2. x = int(input("Enter First Number:"))  
3. y = int(input("Enter Second Number:"))  
4. print(x/y)  
5. except ZeroDivisionError:  
6. print("Can't Divide with zero")  
7. except ValueError:  
8. ~~print~~.print("please provide int value only")  
9.  
10. D:\python\_classes> py test.py  
11. Enter first Number: 10.  
12. Enter Second Number: 2  
13. 5.0

14. ~~Control flow in try-except finally:-~~

try:

stmt-1

stmt-2

stmt-3

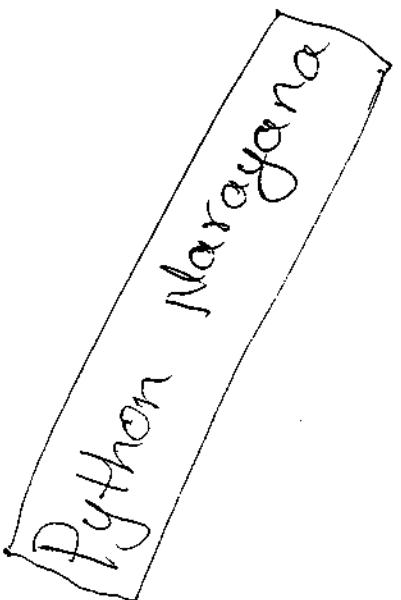
except:

stmt-4

finally:

stmt-5

stmt-6



Case:1 If there is no exception - Python Narayana  
1, 2, 3, 5, 6 Normal Termination.

(489)

Case:2 If an exception raised at stmt2 and the  
Corresponding except block matched 1, 4, 5, 6 Normal  
Termination.

Case:3 If an exception raised at stmt2 but the Corresponding  
except block not matched 1, 5 Abnormal A Termination.

Case:4 If an exception raised at stmt4 then it always  
abnormal termination but before that finally block  
will be executed.

Case:5 If an exception raised at stmt-5 or at  
stmt-6 then it is always abnormal termination.

Nested try-except-finally blocks

We can take try-except-finally blocks inside try  
or except or finally blocks, nesting of try except  
finally is possible-

try:

    =====

    try :

        =====

# Python Narayana.

(490)

except:

=====

except:

=====

Control flow is nested by-except-finally:

try:

stmt-1

stmt-2

stmt-3

try:

stmt-4

stmt-5

stmt ~~6~~ on

except:

stmt-7

finally:

stmt-8

stmt-9

except 4:

stmt-10

finally:

stmt-11

stmt-12

Narayana

## else block with try-except-finally, Python Narayana

We can use else block with try-except-finally blocks  
else block will be executed if and only if there are  
no exceptions inside try block.

try:

Risky code

except:

will be executed if exception inside try

else:

will be executed if there is no exception inside try

finally:

will be executed whether exception raised or

not raised & handled or not handled.

Ex:

try:

print("try")

print(10/0) → 1

except:

print("except")

else:

print("else")

finally:

print("finally")

Python Narayana

## Python Narayana

- If we comment line-1 then else block will be executed because there is no exception inside try.

(492)

In this case the output is:

```
try
else
finally.
```

- If we are not commenting line-1 then else block won't be executed because there is exception inside try block. In this case output is:

```
try
except
finally.
```

*Python Narayana*

Output:3

please Enter your percentage : 88

(493)

~~you got distinction.~~

Output:4

please Enter your ~~percentage~~ : -20

~~LessThanZeroException: you entered less than 0 marks.~~

Python Narayana

10

## \* Unit testing \* Python Narayana

(49)

1. unittest is the batteries-included test module in the python standard library.
2. A testing unit should focus on the tiny bit of functionality and prove it correct
3. Each test unit must be fully independent. Each test must be able to run alone. Try hard to make test that run fast. If one single test needs more than a few milliseconds to ~~run~~, development will be slowed down or the ~~test~~ will not be run as often as is desirable
4. Learn your tools and learn how to run a single test or a test suite. Even when developing a function inside a module, run this function's test frequently.
5. Always run the full test suite before a coding session, and run it again after this will give you more confidence that you did not break anything in the rest of the code.
6. If you are in the middle of a development session and have to interrupt your work,

Python Narayana

it is a good idea to write a broken unit test  
about what you want to develop next (125)

7. When coming back to work, you will have a pointer to where you were and get back on track faster.
8. The first step when you are debugging your code is to write a new test pinpointing the bug.
9. Use long and descriptive names for testing functions.

Python supports different inbuilt methods,

1. assertEquals()
2. assertNotEqual()
3. assertTrue()
4. assertFalse()
5. assertIs()
6. assertIsNot()
7. assertIsNone()
8. assertIsNotNone()
9. assertIn()
10. assertNotIn()
11. assertIsInstance()
12. assertIsNotInstance()

# Python Narayana

496

```
Eg:1
import unittest
def fun(x):
 return x+1

class MyTest(unittest.TestCase):
 def test(self):
 self.assertEqual(fun(3), 4)

if __name__ == "__main__":
 unittest.main()
```

## Output

Ran 1 test in 0.016s

ok.

```
Eg:2
import unittest
def sum1(m,n):
 return m+n

def diff1(m,n):
 return m-n

def mult1(m,n):
 return m*n

class Myunit(unittest.TestCase):
 def test_sum1(self):
 self.assertEqual(sum1(2,3), 5)

 def test_diff1(self):
 self.assertEqual(diff1(4,2), 2)

 def test_mult1(self):
```

(49)

```

 self.assertEqual(meet((2,3),6)
def test_casetesting(self):
 st='python'
 self.assertEqual(st.islower(),True)
def test_casetesting(self):
 st='pytHON'
 self.assertEqual(st.isupper(),False)
for testing all test cases of the class.
unittest.main()

```

Output:

Ran 4 tests in 0.016s

Ok.

Eg3:

```

import unittest
def sum1(m,n):
 return m+n
def diff1(m,n):
 return m-n
def mult1(m,n):
 return m*n

```

Python Narayana

class Myunit(unittest.TestCase): Python Narayana  
498

498

```
def test_sum1(self):
```

```
self.assertEqual(sum1(2,3),6)
```

```
def test_diff_1(self):
```

Self-assertEqual (diff1 (4,12), 2)

```
def test_onel1(self):
```

self.assertEqual(mul1(2, 3), 8)

```
def test_casttesting (self):
```

```
st='python'.
```

Self.assertTrue(st.islower(), True)

def test - cast testing (self):

```
str = 'pytTHON'
```

```
self.assertEqual(s[1].isupper(), False)
```

unittest.main()

## Output:

FAIL: test-mult (-main-->MyUnit)

AssertionError: 6 ] =8

FAIL: test-Sum1 (-main--Myunit)

AssertionError: 5 != 6

Ran 4 tests in 0.016s

FAILED (failures = 2)

We can also test a specific test rather than all tests.

(199)

Eg:4

```
import unittest
def Sum1(m,n):
 return m+n
class Myunit(unittest.TestCase):
 def test_Sum1(self):
 self.assertEqual(Sum1(2,3), 6)
 def test_Casetesting(self):
 self.assertTrue(st.islower(), True)
 def test_Casetesting(self):
 st = 'pytTHON'
 self.assertFalse(st1.isupper(), False)
obj = Myunit()
obj.test_Sum1.
```

Eg:5

```
import unittest
def Sum1(m,n):
 return m+n
class Myunit(unittest.TestCase):
 def test_Sum1(self):
```

Self.assertEqual(sum1(2, 3), 8) Python Narayana

def test\_casetesting(self):  
 st = 'python'

(500)

self.assertTrue(st.islower(), True)

def test\_casetesting(self):

st = 'pytTHON'

self.assertFalse(st1.isupper(), False)

obj = Myunit()

obj.test\_sum1()

Output:

AssertionError : 5} !=

Python Narayana

## Iterators:

Python Narayana

(50)

Iterator in python is used with a 'for in loop'.  
python lists, tuples, dicts and sets are all examples  
of inbuilt iterators. These types are iterators because  
they implement following methods. In fact, any  
object that wants to be an iterator must implement  
following methods.

1. `__iter__` method is used to start new iteration,  
this method is called on initialization of an iterator  
this should return an object that has a `next`  
`__next__` (in python 3) method.
2. `__next__` method should return the next value  
for the iterable. When an iterator is used with  
a 'for in' loop, that for loop implicitly call  
`next()` on the iterator object. This `object` method  
should raise a `StopIteration` to signal the  
end of the iteration.

Python Narayana

Eg:1 Iterating over string object: Python Narayana.

(502)

st = 'python Narayana'

for i in st:

print(i)

Output:

P  
y  
t  
h  
o  
n  
  
a  
n  
a  
r  
a  
g  
a  
n  
a

Python Narayana

Eg:2 Iterating over list object

Lst = [10, True, 'Venkat', 10.5, 20+5j]

for i in Lst:

Print(i)

Output

10

True

Venkat

10.5

(20+5j).

Eg:3 Iterating over tuple object

```
tuple = (1, False, 'sira', 4+6j, 1.8)
```

```
for i in tuple:
```

```
 print(i)
```

Output:

1

False

Sira

(4+6j)

1.8.

Python Narayana

Eg:4 Iterating over Set objects

```
Set1 = {10, 'Narayana', 'Venkat', 'sira', False, 20.8}
```

```
for i in Set1:
```

```
 print(i)
```

Output

10

Venkat

sira

Narayana

20.8

## Eg:5 Iterating Over dict object Python Narayana

(504)

```
dict1 = { 'id': 101, 'Name': 'Venkat', 'Company': 'TCS',
 'Sal': 20000}
```

```
for i in dict1:
```

```
 print(i)
```

Output:-

id

Name

Company

Sal

Python Narayana

## Generators:

Python Narrator:-

(505)

- generators are very easy to implement, but a bit difficult to understand.
- generators are used to create iterators, but with a different approach. Generators are simple functions which return an iterable set of items, one at a time, in a special way.
- when an iteration over a set of item starts using for statement, the generator is run. Once the generator's function code reaches a "yield" statement, the generator yields its execution back to the for loop, returning a new value from the set. The generator function can generate as many values (possibly infinite) as it wants, yielding each one in its turn.
- Here is a simple example of a generator function which returns 7 random integers.

## Python Narayana

(506)

Eg:-  
import random

def Lottery():

# returns 6 numbers between 1 and 40.

for i in range(6):

yield random.randint(1, 40).

# returns a 7<sup>th</sup> number between 1 and 15.

yield random.randint(1, 15).

for random\_number in Lottery():

print("And the next number is --- %d! "% (random\_number))

Eg:1

```
1. def mygen():
2. yield 'A'
3. yield 'B'
4. yield 'C'
5.
6. g = mygen()
7. print(type(g))
8.
9. print(next(g))
10. print(next(g))
11. print(next(g))
12. print(next(g))
```

Python  
Narayana  
Py3

# Python Narayana

(507)

- 13)
- 14) Output
- 15) <class 'generator'>
- 16) A
- 17) B
- 18) C
- 19) Traceback (most recent call last):
- 20) file "test.py", line 12, in < module >
- 21) print(next(g))
- 22) StopIteration.

Eg: 2:

1. def Countdown(num):
2. print("Start Countdown")
3. while(num>0):
4. yield num
5. num = num - 1
- 6.
7. values = Countdown(5)
8. for x in values:
9. print(x)
- 10.
11. Output
12. Start Countdown.
13. 5
14. 4
15. 3
16. 2
17. 1

Python Narayana

Eg:3 To generate first n numbers: Python Narayana

(508)

```
1. def firstn(num):
2. n=1
3. while n<=num:
4. yield n
5. n=n+1
6.
7. values = firstn(5)
8. for x in values:
9. print(x)
10.
11. Output
12. 1
13. 2
14. 3
15. 4
16. 5
```

We can convert generator into list as follows.

```
values = firstn(10)
l1 = list(values)
print(l1) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Eg:4 To generate fibonacci Numbers---

the next is the sum of previous 2 Numbers.

Eg: 0, 1, 1, 2, 3, 5, 8, 13, 21---

# python Narayana

609

```
1. def fib():
2. a, b = 0, 1
3. while True:
4. yield a
5. a, b = b, a+b
6. for f in fib():
7. if f > 100:
8. break
9. print(f)
10
11. output
12. 0
13. 1
14. 1
15. 2
16. 3
17. 5
18. 8
19. 13
20. 21
21. 34
22. 55
23. 89
```

fibonacci

Python

## Advantages of generator functions

Python Narayana

(510)

1. When compared with class level iterators, generators are very easy to use.
2. Improves memory utilization & performance
3. Generators are best suitable for reading data from large number of large files.
4. Generators work great for web scraping & crawling.

Python Narayana

## Decorators:

A decorator a function that takes another function and returns a newer and prettier version of that function.

- The decorator will not modify the existing functionality of other functions, it will just add new functionality to the existing functionality of the functions.
- Decorators concept is the most beautiful and most powerful design possibilities in python, but at the same time the concept is considered by many as complicated to get into. To be precise, the usage of decatos is very easy, but writing dec decorators can be complicated, especially if you are not experienced with decorators and some functional programming concepts.

Eg:-

```
def dec_fun(func): #decorator function
 def inner():
 pass
```

Python Narayana

```
print("Hello, I am decorator function and i
will pass data to the display()") (512)
func)
```

return inner.

@ dec-fun

```
def display(): # ordinary function
print("I am ordinary function and i am going to
dec-func() to take new functionality from
dec-func() and i will come back with decorator
data")
display() # ordinary function call.
```

Output:

Hello, I am decorator function and i will pass data to  
the display()

I am ordinary function and i am going to dec-func()  
to take new functionality from dec-func() and i will  
come back with decorator data.

Eg:2

```
def Smart_divide(func):
 def inner(a,b):
```

## Python Narayana

```
print ("I am going to divide", a, "and", b)
if b==0:
 print ("whoops! Cannot divide")
 return
 return func(a,b)
return inner
```

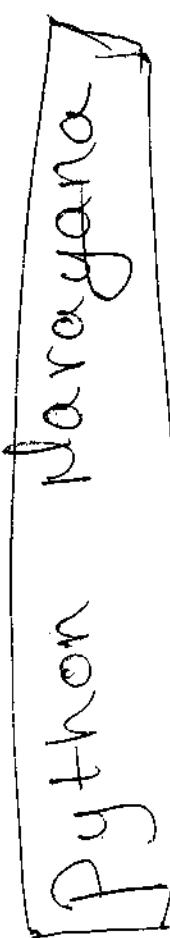
513

### @ Smart-divide

```
def divide (a,b):
 return a/b.
```

#### Output 1 :-

```
>>> divide (3,6)
I am going to divide 3 and 6
0.5.
```



#### Output 2 :-

```
>>> divide (15,0)
I am going to divide 15 and 0.
Whoops! Cannot divide
```

#### Output 3 :-

```
>>> divide (4,10)
I am going to divide 4 and 10
0.4
```

Output 4:

```
>>> divide(12,4)
```

I am going to divide 12 and 4

3.0

Eg:3:

```
import time
```

```
def decor(func):
```

```
 def wrap(*args):
```

```
 start = time.time()
```

```
 result = func(*args)
```

```
 end = time.time()
```

```
 print(func.__name__, "has taken", end-start)
```

```
 return result
```

```
return wrap.
```

```
@decor
```

```
def sqrs(m):
```

```
x = []
```

```
start = time.time()
```

```
for i in m:
```

```
x.append(i*i)
```

```
end = time.time()
```

```
print('sqrs has taken', end-start)
```

return x

Python Narayana

(55)

@decor

def cubes(m):

y = []

# start = time.time()

for i in m:

y.append(i\*i\*i)

# end = time.time()

# print('cube has taken', end-start)

return y.

@decor

def powers(m):

z = []

# start = time.time()

for i in m:

z.append(i\*i\*i\*i)

# end = time.time()

# print('powers has taken', end-start)

return z

lst = range(1000000)

a = squares(lst)

# print(a)

Python Narayana

```
b = cubes(lst)
```

```
#print(b)
```

```
c = fours(lst)
```

```
Print(c)
```

### Output:

Sqrs has taken 0.3651684326171875

Cubes has taken 0.5173401832580566

Fours has taken 0.525939709876709

Eg:- 1. def decor(func):

2. def inner(name):  
3. if name == "Narayana":  
4. print("Hello Narayana Bad Morning")  
5. else:  
6. func(name)

7. return inner  
8.

9. @decor

10. def wish(name):  
11. print("Hello", name, "Good morning")

12.

13. wish("Santhoshi")

14. wish("Shiva")

15. wish("Narayana")

16.

17. output

18. Hello Santhoshi Good Morning

19. Hello Shiva good morning

20. Hello Narayana bad morning.

# Multi Threading:

python Narayana

(517)

- Multithreading is nothing but running multiple operations parallel.
- Running several threads is similar to running several different programs concurrently, but with the following benefits.
  - \* Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
  - \* Threads are sometimes called light-weight processes and they don't require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

Ex:1

```
import threading.
class Thread1(threading.Thread):
```

```
 def run(self):
```

```
 i=1.
```

```
 while i <= 10:
```

```
 print(i)
```

```
 i=i+1.
```

```
class Thread2(threading.Thread):
```

```
 def run(self):
```

```
 j=11.
```

```
 while j <= 20:
```

```
 print(j)
```

```
 j=j+1.
```

```
class Thread3(threading.Thread):
```

```
 def run(self):
```

```
 k=21
```

```
 while k <= 30:
```

```
 print(k)
```

```
 k=k+1.
```

```
obj1 = Thread1()
```

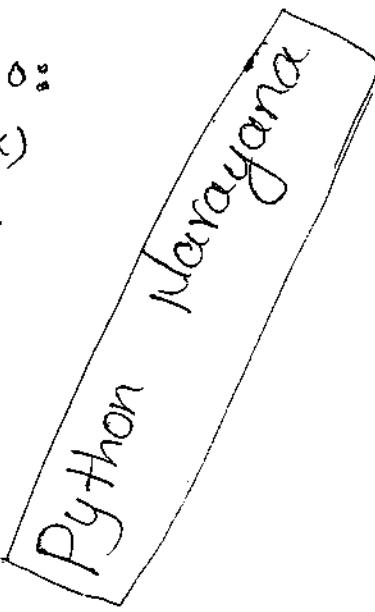
```
obj2 = Thread2()
```

```
obj3 = Thread3()
```

```
obj1.start()
```

```
obj2.start()
```

```
obj3.start()
```



## Output

11121  
21222  
31323  
41424  
51525  
61626  
71727  
81828  
91929  
102030.

python Narayana

(519)

## Eg:2

```
import threading
```

```
def print_cube(num):
```

```
 print ("Cube : {} ".format (num * num * num))
```

```
def print_square(num):
```

```
 print ("Square : {} ".format (num * num))
```

```
if __name__ == "__main__":
```

```
 t1 = threading.Thread(target = print_square, args = (10,))
```

```
 t2 = threading.Thread(target = print_cube, args = (10,))
```

```
t1.start()
```

```
t2.start()
```

```
t1.join()
```

```
t2.join()
```

```
print("Done!")
```

Output

square: 100 cube: 1000

Done!

(520)

Python

Narayana

## \* Database Connection \*

(52)

Step:1 Open my SqL

Step:2 Enter password 'root' to login to the mysql database

Enter password : \*\*\*

Welcome to the MySQL monitor. Commands end with; or

\q.

Your MySQL connection id is 1.

Server Version: 5.0.41-Community-nt MySQL Community  
Edition (GPL)

Type 'help'; or 'h' for help. Type '\c' to clear the buffer.

Step:3 To see list of databases.

mysql> show databases;

Database
information_schema
mysql
test

3 rows in set (0.01 sec).

## Step:4 Creating user defined database Python Narayana

mysql > create database pythondb;

(522)

Query Ok, 1 row affected (0.00sec).

## Step:5 To See list of databases.

mysql > show databases;

Database
information schema
mysql
pythondb
test

4 rows in set (0.00 sec)

## Using specific database

mysql > use pythondb;

Database changed.

## Creating a table in pythondb:-

mysql > create table pytabl(id int, name varchar(10),  
loc varchar(10), sal int);

Query OK, 0 rows affected (0.11 sec) Python Narayana

To see table description

(523)

mysql > desc pytab;

Field	Type	Null	Key	Default	Extra
id	int(11)	YES		NULL	
Name	varchar(10)	YES		NULL	
Loc	varchar(10)	YES		NULL	
Sal	int(11)	YES		NULL	

4 rows in set (0.02 sec)

Inserting data into the table:

mysql > insert into pytab values (1, 'Narayana', 'Hyd', 1000);

Query OK, 1 row affected (0.08 sec)

mysql > insert into pytab values (2, 'Satya', 'Hyd', 1500);

Query OK, 1 row affected (0.06 sec)

mysql > insert into pytab values (3, 'Anu', 'Bang', 1100);

Query OK, 1 row affected (0.06 sec)

mysql > insert into pytab values (4, 'krishna', 'chennai', 1200);  
Query ok, 1 row affected (0.06sec) 524

mysql > insert into pytab values (5, 'chandu', 'chennai', 1100);  
Query ok 1 row affected 0.06 sec).

### To display table data:-

mysql > select \* from pytab;

id	name	loc	sal
1	Narayana	Hyd	1000
2	satya	Hyd	1500
3	Ane	Bang	1100
4	Krishna	chennai	1200
5	chandu	chennai	1100

5 rows in set (0.00 sec).

### Installing pymysql:-

Go to Command prompt,

1. C:\Users\Narayana>cd ..
2. C:\Users>cd:
3. C:\>cd C:\python27\scripts
4. C:\python27\scripts>pip install matplotlib.

Collecting matplotlib.

Python Narayana

5. C:\python27\scripts> pip install pymysql.

625

6. Collection pymysql.

Using cached pymysql-0.7.11-py2.py3-none-any.whl  
installing collected packages: pymysql.

Successfully installed pymysql-0.7.11.

1. Open python

2. Open new file and write the below code

```
import pymysql
conn = pymysql.Connection(host='localhost', user='root',
 password='root', db='pythondb')

conn.cursor()
sql = "select * from pytab;"
conn.execute(sql)
d = conn.fetchall()
print(d).
```

Output:

```
((1, 'Narayana', 'Hyd', 1000), (2, 'Satya', 'Hyd', 1500), (3, 'Anu', 'Bang.',
1100), (4, 'Krishna', 'Chennai', 1200), (5, 'Chander', 'Chennai', 1100)).
```

To get all rows one by one Python Narayana

```
import pymysql
```

526

```
Conn= Connect.Cursor()
```

```
Sql = "Select * from pytab";
```

Conc. execute(SQL)

```
d= conn.fetchall()
```

for i in:

```
print(i)
```

Output:

(1, 'Narayana', 'Hyd', 1000)

(2, 'satya', 'Hyd', 1500)

(3, 'Anu', 'Bang', 1500)

(4) 'krishna', 'Chennai', 1200)

(5, 'chandu', 'chennai', 1000).

To get all employee names

```
import pymysql
```

```
import pymysql
connect = pymysql.Connection(host='localhost', user='root',
password='root', db='pythondb')
```

Conn = connect.cursor()

Python Narayana

(527)

Sql = " select \* from pytab;"

Conn = connect

conn.execute(Sql)

d = Conn.fetchall()

for i in d:

print([i])

Output:

Narayana

Satya

Anu

Krishna

Chandu.

To get all the details of employee, 'Narayana'.

import pymysql

Connect = pymysql.connect (host='localhost', user='root',

password='root', db='python db')

Conn = Connect.cursor()

Sql = "select\* from pytab where name = 'Narayana';"

Conn.execute(Sql)

d = Conn.fetchall()

for i in d:

print(i)

## Python Narayana

(528)

Output:-

('1', 'Narayana', 'Hyd', 1000)

To get all employee names who are working  
in 'Chennai':

```
import pymysql
```

```
Connect = pymysql.Connection(host='localhost', user='root',
 password='root', db='pythondb')
```

```
Conn = Connect.cursor()
```

```
Sql = "select name from pytab where loc='chennai';"
```

```
Conn.execute(Sql)
```

```
d = Conn.fetchall()
```

```
for i in d:
```

```
 print(i)
```

Output:-

('Krishna')

('Chandu')

(d)

```
import pymysql
```

```
Connect = pymysql.Connection(host='localhost', user='root',
 password='root', db='pythondb')
```

```
Conn = Connect.cursor()
```

Python Narayana

```
Sql = "Select * from pytab where loc = 'Chennai';"
Conn = conn.execute(Sql)
d = Conn.fetchall()
for i in d:
 print(i[1])
```

(529)

### Output:

Krishna

Chandu.

Fetchone() :- This function is used to fetch the first record from the table.

### Eg:1

```
import pymysql
Connect = pymysql.Connection(host = 'localhost', user
= 'root', password = 'root', db = 'pythondb')
Conn = Connect.cursor()
Sql = "Select * from pytab;"
Conn.execute(Sql)
d = Conn.fetchone()
print(d).
```

Output: (1, 'Narayana', 'Hyd', 1000)

## Python Narayana

(530)

Eg:2:- import pymysql

```
Connect = pymysql.Connection(host='localhost', user='root',
 password='root', db='pythondb')
```

```
Conn = Connect.cursor()
```

```
Sql = "Select * from pytab where loc='chennai'"
```

```
Conn.execute(Sql)
```

```
d = Conn.fetchone()
```

```
Print(d)
```

Output:-

```
(4, 'krishna', 'chennai', 1200)
```

Fetchmany(n):- this function is used to fetch n number of records.

Eg:1

```
import pymysql
```

```
Connect = pymysql.Connection(host='localhost', user='root',
 password='root', db='pythondb')
```

```
Conn = Connect.cursor()
```

```
Sql = "Select * from pytab;"
```

```
Conn.execute(Sql)
```

## Python Narayana

```
d = Conn.fetchmany(3)
```

```
for i in d:
```

```
 print(i)
```

(531)

### Output

```
(1, 'Narayana', 'Hyd', 1000)
```

```
(2, 'satya', 'Hyd', 1500)
```

```
(3, 'Anu', 'Bang', 1100)
```

### Eg:2

```
import pymysql
```

```
Conn = pymysql.Connection(host='localhost', user='root',
 password='root', db='pythondb')
```

```
Conn = Conn.cursor()
```

```
Sq1 = "Select* from pytab where loc = 'chennai';"
```

```
Conn.execute(Sq1)
```

```
d = Conn.fetchmany(2)
```

```
for i in d:
```

```
 print(i).
```

### Output

```
(4, 'krishna', 'chennai', 1200)
```

```
(5, 'chander', 'chennai', 1100)
```

# \* Numpy \*

python Narayana

(532)

1. The python programming language was not initially designed for numerical computing, but attracted the attention of the specific scientific/engineering community early on.
2. A special interest group called "matrix-sig" was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer/maintainer "Guido Van Rossum" who implemented extensions to make array computing easier.
3. Numpy is an open source extension module for python.
4. The module Numpy provides fast precompiled functions for numerical routines.
5. It also adds to support to python for large, multi-dimensional arrays & matrices.
6. It also supplies a large library of high-level mathematical functions to operate on these arrays.

7. Scipy (Scientific Python) extends the functionalities of Numpy with further useful functions. (533)
8. Both Numpy and Scipy are usually not installed by default. Numpy has to be installed before installing Scipy.
9. Numpy is based on two earlier python array packages. Those are Numeric and Numarray.
10. Numpy is a merger of those two i.e., it is build on the code of Numeric and the features of Numarray.
11. One of the main advantages of Numpy is its advantage in time compared to standard Python.
- Installation Steps:
1. Install numpy and opencv 2-4.13.
  2. Goto opencv folder `C:\Users\Narayana\Downloads\opencv\build\python\2.7\X86` and copy cv2 file.
  3. Goto python folder `C:\python27\lib\site-packages` and paste that cv2 file here.
  4. Goto Command prompt `C:\users\Narayana>C:\Python27\python.exe` and run it.
  5. import numpy.

# Benefits of numpy Over list Python Noragana

(534)

1. Requires less memory
2. Runs very fast
3. Convenient to developer.

## Requires Less Memory

```
import numpy as np
```

```
import sys
```

```
import time
```

```
list1 = range(1000)
```

```
print ("the memory for list1 is", sys.getsize() * len(list1))
```

```
array1 = np.arange(1000)
```

```
print ("the memory for array is", array1.size * array1.itemsize)
```

## Output

the memory for list1 is 12000

the memory for array is 4000.

## Runs very fast

```
import numpy as np
```

```
import sys
```

```
import time
```

List1 = list(range(100000))      Python Numpy and

List2 = list(range(100000))

(535)

arr1 = np.array(List1)

arr2 = np.array(List2)

Start = time.time()

result = [x+y for x, y in zip(List1, List2)]

print("python list took:", (time.time() - start)\*1000)

Start = time.time()

result = arr1 + arr2.

print("numpy took:", (time.time() - start)\*1000)

## Output

python list took : 62.99996316037598.

numpy took : 0.9999275207519531

## Very Convenient for Developers

We can't add elements of two lists directly, but  
two arrays can be added by using numpy.

## Eg:-

import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])

arr2 = np.array([10, 20, 30, 40, 50])

```
add_result = arr1 + arr2
```

```
print(add_result)
```

```
Sub_result = arr2 - arr1
```

```
print(Sub_result)
```

```
Mul_result = arr1 * arr2
```

```
Print(Mul_result)
```

```
Import numpy as np.
```

~~# Creating one dimensional array~~

```
a = np.array([1, 2, 3])
```

```
print(a)
```

```
print(a.ndim)
```

```
print(a.itemsize)
```

```
print(a.shape)
```

```
print(a.dtype)
```

Outputs

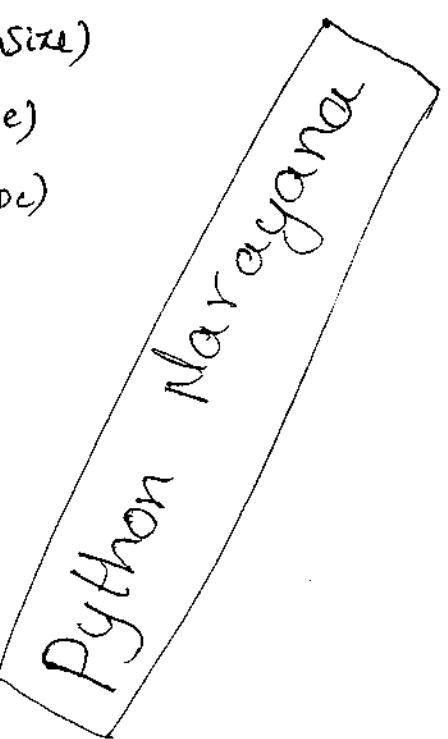
[1, 2 3]

1

4

(3,)

int32.



*Python Naregana*  
# Creating two dimensional array (2\*3 matrix)

a = np.array([[1, 2, 3], [2, 3, 4]])

(537)

print(a)

print(a.ndim)

print(a.itemsize)

print(a.shape)

print(a.dtype).

Output

[[1 2 3]  
[2 3 4]]

2

4

(2, 3)

int32.

# Creating 4\*2 matrix.

a = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])

print(a)

print(a.ndim)

print(a.itemsize)

print(a.dtype)

print(a.shape)

# Python Narayana

638

## Output

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

2

4

int 32

(4, 2)

# creating array with float type data.

```
a = np.array([[1, 2], [2, 3], [3, 4], [4, 5]], dtype=np.float64)
```

```
print(a).
```

```
print(a.ndim)
```

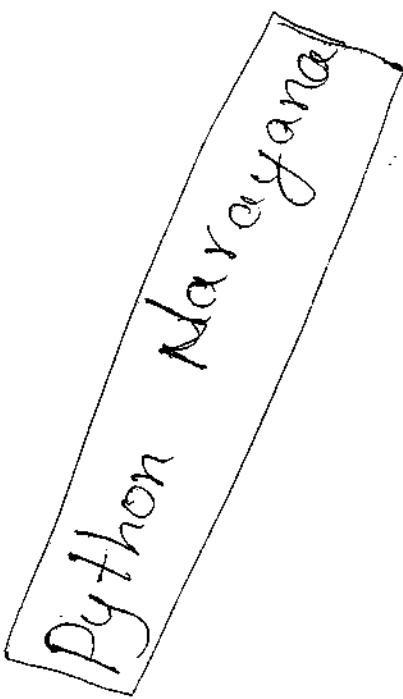
```
print(a.itemsize)
```

```
print(a.dtype)
```

```
print(a.shape)
```

## Output

```
[[1. 2.],
 [2. 3.],
 [3. 4.],
 [4. 5.]]
```



2

8

(53)

float 64  
(4, 2)

#Creating matrix with Complex numbers.

```
a = np.array([[1, 2], [2, 3], [3, 4], [4, 5]], dtype=np.Complex)
print(a)
print(a.ndim)
print(a.itemsize)
print(a.dtype)
print(a.shape).
```

Output

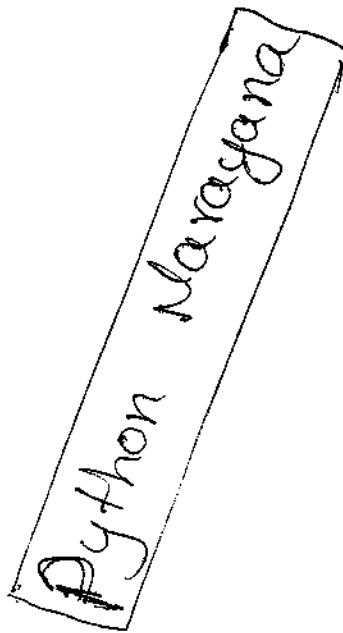
```
[[1.+0.j 2.+0.j]
 [2.+0.j 3.+0.j]
 [3.+0.j 4.+0.j]
 [4.+0.j 5.+0.j]].
```

2

16

Complex 128

(4, 2).



# Creating matrix with placeholder numbers

Eg:1

a = np.zeros((2, 3))

print(a)

## Python Numpy array

(540)

Output:-

```
[[0. 0. 0.]
 [0. 0. 0.]].
```

Eg:2

```
a = np.ones((2,3))
print(a)
```

Output:-

```
[[1. 1. 1.]
 [1. 1. 1.]].
```

# creating a list with range function.

```
I = range(5)
print(I)
print(I[0])
print(I[1])
print(I[2])
print(I[3])
print(I[4])
```

Output:-

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

0

1

2

3

4

Python Numpy array

# Creating an array with arange method.

a = np.arange(175)

python <sup>(guru)</sup> Narayana

print(a)

print(a[0])

print(a[1])

print(a[2])

print(a[3])

Output

[ 1 2 3 4].

1

2

3

4.

a = np.arange(10, 10, 2)

print(a)

print(a[0])

print(a[1])

print(a[2])

print(a[3])

print(a[4])

Output is

[ 1 3 5 7 9 ]

1

3

5

7

9

# Creating an array with linspace python Narayana

(612)

Eg:1

a = np.linspace(1, 5, 10)

print(a)

Output:-

[1. 1.44444444 1.88888889 2.33333333 2.77777778  
3.22222222 3.66666667 4.11111111 4.55555556 5.]

Eg:2

a = np.linspace(1, 5, 5)

print(a).

Output:-

[1. 2. 3. 4. 5.]

# Reshaping array.

a = np.array([[1, 2, 3], [2, 3, 4], [4, 5, 6], [5, 6, 7]])

print(a)

print(a.shape)

Output:-

[[1 2 3]]

[2 3 4]

[4 5 6]

[5 6 7]]

(4, 3)

print(a.reshape(3, 4)).

[[1 2 3 2]]

[3 4 4 5]]

[6 5 6 7]].

print(a.reshape(2,6))

[ [ 1 2 3 2 3 4 ]

[ 4 5 6 5 6 7 ] ].

print(a.reshape(6,2))

[ [ 1 2 ]

[ 3 2 ]

[ 3 4 ]

[ 4 5 ]

[ 6 5 ]

[ 6 7 ] ].

# flatten array.

a = np.array([ [ 1, 2, 3 ], [ 3, 4 ], [ 4, 5, 6 ], [ 5, 6, 7 ] ])

print(a)

print(a.ravel())

print(a).

Output:

[ [ 1 2 3 ]

[ 2 3 4 ]

[ 4 5 6 ]

[ 5 6 7 ] ]

[ 1 2 3 2 3 4 4 5 6 5 6 7 ].

[ [ 1 2 3 ]

[ 2 3 4 ]

[ 4 5 6 ]

[ 5 6 7 ] ].

Python Narayana

(543)

## # Aggregate Operations 2

Python Narayana

(5/4)

```
a = np.array([[1, 2, 3], [2, 3, 4], [4, 5, 6], [5, 6, 7]])
```

```
print(a.max())
```

```
print(a.min())
```

```
print(a.sum())
```

```
print(a)
```

```
print(a.sum(axis=0))
```

```
print(a.sum(axis=1)).
```

## Output

7

1

48

[ [ 1 2 3 ] ]

[ 2 3 4 ] ]

[ 4 5 6 ] ]

[ 5 6 7 ] ]

[ 12 16 20 ] ]

[ 6 9 15 18 ].

## # arithmetic operations:

```
a1 = np.array([[1, 2, 3], [2, 3, 4]])
```

```
a2 = np.array([[4, 5, 6], [3, 4, 5]])
```

```
print(a1)
```

[ [ 1 2 3 ] ]

[ 2 3 4 ] ]

Python  
Narayana

# Python Narayana -

545

```
print(a2)
```

```
[[4 5 6]]
```

```
[[3 4 5]]
```

```
print(a1+a2)
```

```
[[[5 7 9]]]
```

```
[[5 7 9]] .
```

```
print(a1-a2)
```

```
[[[-3 -3 -3]]]
```

```
[[-1 -1 -1]]]
```

```
print(a1*a2)
```

```
[[[4 10 18]]]
```

```
[[6 12 20]] .
```

```
print(a1/a2)
```

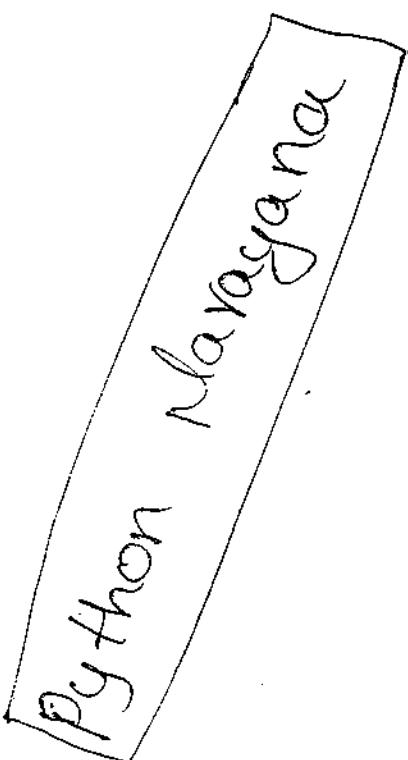
```
[[[0 0 0]]]
```

```
[[0 0 0]]]
```

```
print(a1% a2)
```

```
[[[1 2 3]]]
```

```
[[2 3 4]] .
```



## Logging :-

- Logging is used to track the runtime information of an application into a separate log file.
- We use Logging Module to track the run data in a file.
- We use "Logging.basicConfig()" to create a new log file and to change the logging Level.

## Syntax:-

basicConfig(filename, Level)

- filename argument, it used to create a new log file and the filename must have 'log'
- Level argument is used to change or specify the logging Level.
  - 1. DEBUG It represents that the current function has detailed data.
  - 2. INFO It represents that the current function running well but we can cross check whether it is running as you expected (or) not.
- Python supports different types of Logging Levels.

3. WARNING: It represents that the current function is running well currently but it may fail in the near future.

(549)

4. ERROR: It represents that the current function has some errors.

5. CRITICAL: It represents that the entire application may fail. the default level is WARNING.

Eg:-

```
import logging
logging.basicConfig(filename='mylog.log', level=
logging.CRITICAL)
```

```
def sum1(a,b):
 return a+b
```

```
def diff1(a,b):
 return a-b
```

```
def mult1(a,b):
 return a*b
```

```
def div1(a,b):
 return a/b
```

```
def mod1(a,b):
 return a%b
```

m=20

n=10

Python Narayana

# Python Narayana

(548)

res-sum1 = sum(m,n)

logging.info (the sum of \$3 and \$3 is \$3. format(m,n,res-  
sum1))

res-diff1 = diff1(m,n)

logging.debug (the diff of \$3 and \$3 is \$3. format(m,n,res-  
diff1))

res-div1 = div1(m,n)

logging.error (the div of \$3 and \$3 is \$3. format(m,n,res-  
div1))

res-mod1 = mod1(m,n)

logging.critical (the mod of \$3 and \$3 is \$3. format(m,n,  
res-mod1))

O/p

```
File Edit format - □ X
INFO: root: the sum of 20 and 10 is 30
DEBUG: root: the diff of 20 and 10 is 10
WARNING: root: the multiplication of 20 & 10 is 200
ERROR: root: the division of 20 and 10 is 2.0
CRITICAL: Root: the mod of 20 and 10 is 0.
```

go to python directory & open my log file

## Comprehensions:

python Narayana

(54)

Python supports a concept called "List Comprehensions". It can be used to construct lists in a very natural, easy way.

→ It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.

The expressions can be anything, meaning you can put in all kinds of objects in lists.

The result will be a new list resulting from evaluating the expression in the context of the for and if clauses which follows it.

The list comprehension always returns a result list. List comprehension is powerful and must know Concept in python.

List comprehensions are 35% faster than for loop and

45% faster than map function

\*result\* = [\* Expression\* \* Iteration\* \* filter\*].

# Python Narayana

(550)

Egii

```
lst = []
list1 = [1, 2, 3, 4, 5]
for i in list1:
 lst.append(i*i)
print(lst)
```

Output:

[1, 4, 9, 16, 25]

The alternative to the above script is list comprehension like.

```
lst = [1, 2, 3, 4, 5]
lst = [i*i for i in lst]
print(lst)
```

Output:

[1, 4, 9, 16, 25].

(Q)

```
print([i*i for i in [1, 2, 3, 4, 5]])
```

Output:

[1, 4, 9, 16, 25].

(Q)

```
x = [i*i for i in range(10) if i>5]
```

```
print(x)
```

Output:

[36, 49, 64, 81]

the above comprehension has three arguments, like expression, iteration and condition.

(551)

Eg:2 display all even numbers from the given range by using function.

```
x = range(20)
```

```
b = list(filter(lambda a: a%2 == 0, x))
```

```
print(b)
```

Output:

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18].
```

By using list Comprehension.

```
x = range(20)
```

```
r = [i for i in x if i%2 == 0]
```

```
print(r)
```

Output:

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18].
```

Eg:3 display all positive numbers from given range by using function.

```
x = range(-10, 10, 2)
```

```
b = list(filter(lambda a: a > 0, x))
```

```
print(b)
```

O/p:

```
[2, 4, 6, 8].
```

by using list comprehension

Python Narayana

(552)

numbers = range (-10, 10, 2)

newlist = [n for n in numbers if n > 0]

print (newlist)

O/p: [2, 4, 6, 8]

Eg:-4 adding elements of two lists

lst1 = [1, 2, 3]

lst2 = [2, 3, 4]

by using function

print (list (map (lambda x, y: x+y, lst1, lst2)))

Output: [3, 5, 7]

by using list comprehension

print ([x+y for x, y in zip (lst1, lst2)])

O/p: [3, 5, 7]

Eg:-5 getting first letter of each student by  
using normal for loop

stuNames = ['Durga', 'Narayana', 'Rohit', 'Sai']

lst = []

for i in stuNames:

```
lst.append(i[0])
print(lst)
```

python Narayana

(553)

Output

[`D`, `N`, `R`, `S`]

by using list comprehension.

```
stuNames = ['Dwiga', 'Narayana', 'Rohit', 'Sai']
```

```
names = [i[0] for i in stuNames]
```

```
print(names)
```

O/p [`D`, `N`, `R`, `S`].

Eg:6 extracting only number from the given string

```
string = "python 9010 Oracle"
```

```
numbers = [x for x in string if x.isdigit()]
```

```
print(numbers)
```

O/p [`9`, `0`, `1`, `0`].

Eg:7 extracting only characters from given string

which contains both numbers & string data

```
String = "python 9010 Oracle"
```

```
numbers = [x for x in string if x.isalpha()]
```

```
print(numbers).
```

Output

['P', 'Y', 'T', 'H', 'D', 'N', 'O', 'R', 'A', 'C', 'L', 'E']

Eg:8 display squares of all positive numbers.

```
def Squarethis (numbers):
 return [n**2 for n in numbers if n>0]
print (Squarethis [1,3,5,-8]).
```

O/P :- [ 1, 9, 25 ]Eg:10 display only vowels from the given file

Sampledata - Notepad

file Edit format view help

Narayana, Venkat, Siva,  
 Santhosh, Ideas, intelligence  
 plan, thought, Success, belief  
 guess, innovation, clue, thinking.

a = open ('Sampledata.txt')

r = a.read()

b = [x for x in r if x in ['a','e','i','o','u']]

print(b)

O/p:-

```
[a, a, a, a, e, a, i, a, d, e, a, l, e, e, a, o,
u, u, e, e, i, e, u, e, i, o, a, i, d, u, e, i, i]
```

Eg: 11) Display squares of all elements by using def function, map function & List Comprehension By using def function

```
def Square_for(arr):
 result = []
 for i in arr:
 result.append(i**2)
 return result
print(Square_for([1, 2, 3, 4, 5]))
```

O/p:- [1, 4, 9, 16, 25]

By using map function.

```
Lst = [2, 3, 4, 5, 6, 2]
```

```
print(list(map(lambda x: x**2, Lst)))
```

O/p:- [4, 9, 16, 25, 36, 4]

By using list comprehension.

```
print([x**2 for x in Lst])
```

O/p:- [4, 9, 16, 25, 36, 4]

Python Narayana

Eg12 How to remove all vowels from existing string.

(556)

```
def lc(sentence):
 vowels = 'aeiou'
 filtered_list = []
 for l in sentence:
 if l not in vowels:
 filtered_list.append(l)
 x = ''.join(filtered_list)
 print(x)
st = 'python developer'
lc(st)
```

Output:

pytn dvpr  
or

```
def lc(sentence):
 vowels = 'aeiou'
 return ''.join([l for l in sentence if l not
 in vowels])
st = 'python narayana hyderabad'
print(lc(st))
```

Output:

pytn nryna hydrbd

# pickling and unpickling of objects Python Narayana

(55)

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"

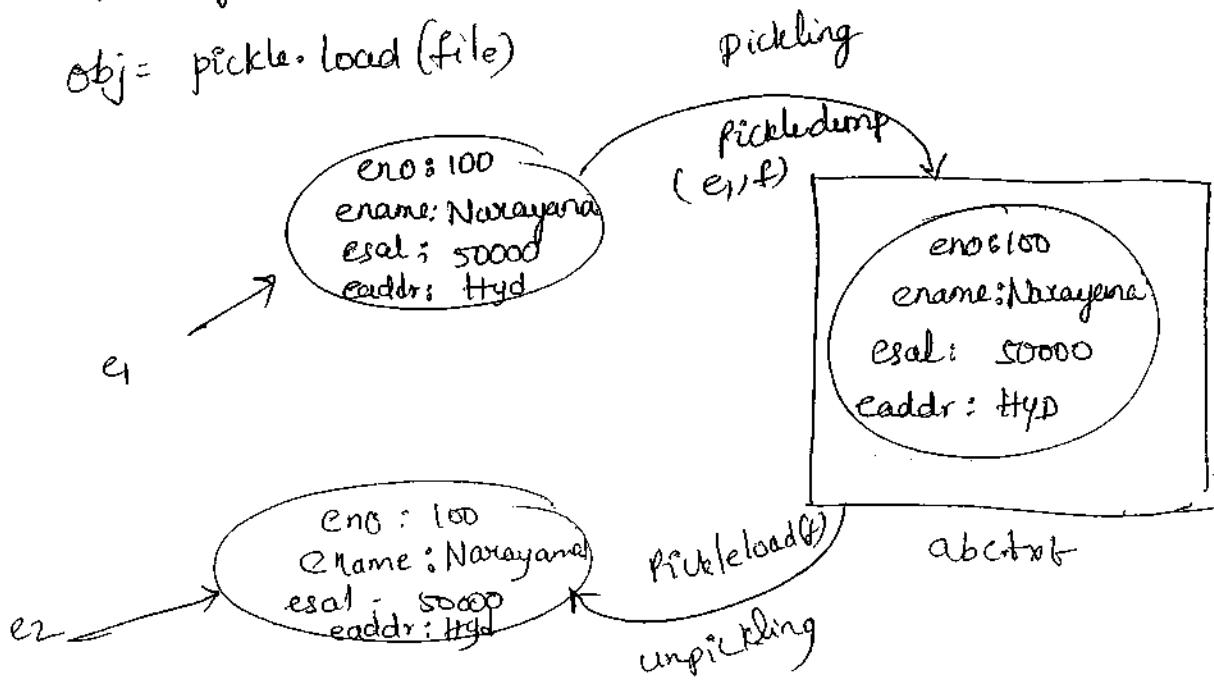
pickle module contains dump() function to perform pickling

`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

Modeler:-

558  
Python Narayana.

1. import pickle
2. Class Employee:
3. def \_\_init\_\_(self,eno,ename,esal,eaddr):
4.     self.eno=eno;
5.     self.ename=ename;
6.     self.esal=esal;
7.     self.eaddr=eaddr;
8. def display(self):
9.     print(self.eno, " ", self.ename, " ", self.esal, " ", self.eaddr)
10. with open("emp.dat", "wb") as f:
11.     with open e = Employee(100, "Durga", 1000, "Hyd")
12.     pickle.dump(e,f)
13. print ("pickling of employee object completed")
- 14.
15. with open("emp.dat", "rb") as f:
16.     obj=pickle.load(f)
17. print ("printing Employee information after unpickling")
18. obj.display()

## Writing Multiple Employee objects to the file

SS9  
Python Narayana

### emp.py :-

```
1. class Employee:
2. def __init__(self,eno,ename,esal,eaddr):
3. self.eno = eno;
4. self.ename = ename;
5. self.esal = esal;
6. self.eaddr = eaddr;
7. def display(self):
8.
9.
10. print(self.eno, "\t", self.ename, "\t", self.esal, "\t",
 self.eaddr)
```

### Pickl.py :-

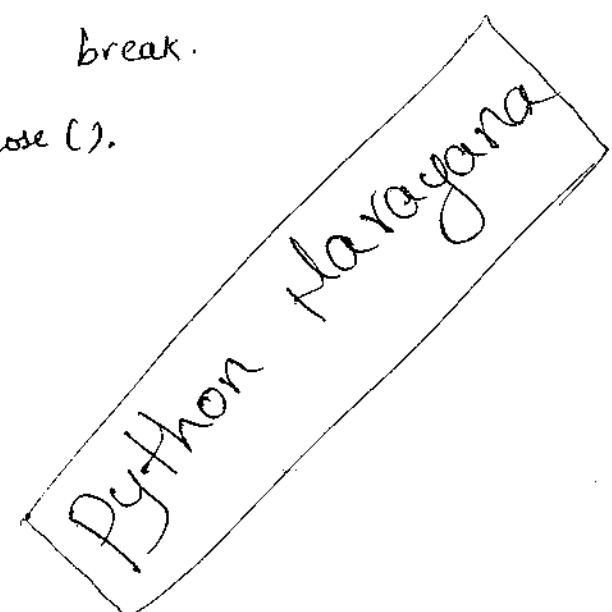
```
1. import emp, pickle
2. f = open("emp.data", "wb")
3. n = int(input("Enter the number of Employees"))
4. for i in range(n):
5. eno = int(input("Enter Employee Number"))
6. ename = input("Enter Employee Name")
7. esal = float(input("Enter Employee Salary"))
8. emp = {"eno": eno, "ename": ename, "esal": esal}
9. pickle.dump(emp, f)
10. f.close()
```

Python Narayana

```
8. eaddr = input ("Enter Employee Address:")
9. e = emp.Employee (eno, ename, esal, eaddr) 560
10. pickle.dump(e,f)
11. print ("Employee objects pickled successfully")
```

### Unpick.py :-

```
1. import emp, pickle
2. f= open ('emp.dat', "rb")
3. print ("Employee Details:")
4. while True:
5. try:
6. obj= pickle.load (f)
7. obj.display()
8. except EOFError:
9. print ("All employees completed")
10. break.
11. f.close ()
```



## Regular Expressions

Python Narayana

(56)

- These are also called re, reg ex.
- These are mainly to check the patterns of mobile & email id.

### Special characters:

- \| Matches the expression to its right at the start of a string. It matches every such instance before each \n in the string.
- \\$ | Matches the expression to its left at the end of a string it matches every such instance before each \n in the string
- .| Matches ~~any~~ character except line terminator like \n or denotes character classes
- \| Escapes special character
- A|B| matches expression A or B. If A is matched first B is left untried.
- +| Greedily matches the expression to its left 1 or more times.
- \*| Greedily matches the expression to its left 0 or more times.

→ ? | Greedily matches the expression to its left 0 or 1 times. But if ? is added to qualifiers (+, \*, (562) and ? itself) it will perform matches in a non-greedy manner.

→ {m} | matches the expression to its left m times & not less.

→ {m, n} | matches the expression to its left m to n times & not less.

→ {m, n}? | matches the expression to its left m times & ignores n. See ? above.

### Character classes (a.k.a Special Sequences):

\w | Matches alphanumeric characters, which means a-z A-Z and 0-9 It also matches the underscore, -

\d | Matches digits which means 0-9.

\D | Matches any non-digits.

\s | Matches whitespace characters, which include the

\t, \n, \r, Space characters.

\S | matches non-whitespace characters

→ \b | Matches the boundary (or empty string) at the start  
end of the word, that is b/w \w and \w. (563)

→ \B | matches where \b doesn't, that is, boundary of \w  
characters.

→ \A | matches the expression to its right at the absolute  
start of the string whether in single or multi-line  
mode

→ \Z | matches the expression to its left at the absolute  
end of a string whether in single or multi-line  
mode

### Sets

→ [] | contains a set of characters to match.

→ [ank] | matches either a, n, or k if it doesn't  
match ank.

→ [a-zA-Z] | matches any alphabet from a to Z

→ [a\-zA-Z] | matches a, -, or Z. It matches - because \ escape it

→ [a-] | matches a or -, because - is not being used  
to indicate a series of characters.

- [-a] | As above, matches a or -
- [a-zA-Z0-9] | matches characters from a to z and also from 0 to 9.
- [(+\*)] | special characters becomes literal inside a set, so this matches (, +, \*, and).

### Groups

- () | matches the expression inside the parentheses & groups it
- (?) | inside parentheses like this, ? acts as an extension notation. Its meaning depends on the character immediately to its right.
- (? pAB) | matches the expression AB, and it can be accessed with the group name.
- (? ailmstuX) | Here, a, i, l, m, s, u, & X are flags:
  - a - Matches ASCII only
  - i - ignore Case.
  - L - Locale dependent
  - m - Multi-line
  - s - matches all
  - u - Matches unicode
  - X - very loose

(?:A) | Matches the expression as represented by A,  
but unlike (?P=A) it cannot be retrieved 565  
afterwards.

(?#---) | A comment. Contents are for us to read,  
not for matching.

(A?=B) | lookahead assertion. This matches the expression  
A only if it followed by B.

(A?!=B) | Look Negative Lookahead assertion. This  
matches the expression A only if it is not followed  
by B.

(?-<=B)A | positive Lookbehind assertion. This matches the  
expression A only if B is immediately to its left.  
This can only matched fixed length expressions.

(?-!B)A | Negative Lookbehind assertion. This matches  
the expression A and if B is not immediately to  
its left. This can only matched fixed length expressions

(?P=name) | matches the expression matched by an  
earlier group named "name".

## popular python re module functions

- `re.findall(A,B)` | Matches all instances of an expression A in a string B and returns them in a list 566
- `re.search(A,B)` | Matches the first instances of an expression A in a string B, and returns it as a re match object
- `re.split(A,B)` | split a string B into a list using the delimiter A.
- `re.sub(A,B,C)` | Replace A with B in the string C

Python Narayana

# Assignment - 4

Python Narayana

(567)

1. what is the meaning of data hiding?
2. what is the main purpose of inheritance? Types of inheritance?
3. what is the difference b/w data abstraction & encapsulation?
4. what is method overriding & method overloading?

5. what is the difference between static & class  
Variables?

python Narayana

(368)

6. What is docstring in python?

7. How to print specified docstring in the o/p?

8. what are the different modes in file handling?

9. what is the default mode in the file handling?

10. what is Seek and tell methods?

11. How add new data to the existing file which  
contains data already? 569

12. write the data "python developer" to the empty file?

13. How to add "Oracle Developer" to the same above  
file (this should be second line)?

14. How to find vowels in the string in file?

15. How to count number of words in the file?

(370)

16. What is purpose of logging?

17. What are different Logging Levels?

18. What is the default logging level?

19. How to print the data in Logging when  
it is DEBUG level?

20. Can we use logging.debug when the Logging Level set to WARNING?

571

21 How to create log file?

22 What is exception?

23 How to handle exceptions in Python?

24 What is the 'else' in exception handling? write a small program by using else?

Python Narayana

25. what is the 'finally'? write a small program by using finally!

(572)

26. How to use multiple exception classes in single except?

27. What is assert?

28. What is multithreading?

29. What is numpy?

python Narayana

30. How it is better than python list? Give any three reasons?

(573)

31. How to create an array by using list?

32. Create a two dimensional Numpy array?

Python

33. We use range() to create a list in python, then what do we use in numpy?

34. Create a  $3 \times 4$  array?

35. Create a  $2 \times 4$  array float values?

574

36. How to see shape of an array?

37. What is item size for numeric element in array?

38. How to create an array for place holder ?

39. How to reshape an array? Give one ex?

40. How to a string "python" as character by character from file?

Python Numpy one

# Python FAQs

python Narayana

(575)

## Part - 1

1. what is the difference between title() and Capitalize()?
2. How to check whether the given string is Lowercase or not?
3. How to know how many characters are existed in given string?
4. what is a split() and what is the default delimiter in the split()
5. what is the output format of split()
6. Is insertion order preserved or not in string?
7. How to reverse the given string?
8. How to get ascending order of the given string?
9. How to get descending order of the given string?
10. How to display the given string 'python' as p..y..t..h..o..n
11. is string mutable or immutable?
12. Can we delete a specific character by using del command?
13. Can we clear a string with clear()
14. How to perform a string packing?

15. What is the slicing operator? Python Narayana (576)
16. How to replace a part of string with new string?
17. How to display the given string 'Narayana' as 'NArAYANA'
18. What is a list?
19. Is a list mutable or immutable object?
20. How can we decide a list is a mutable object?
21. The list is a dynamic object? Why?
22. How to know how many lists are available in the list [10, 20, 10, 10, 30]
23. How to know the index number of a specific element in the list?
24. What is the indexing and slicing?
25. What is list packing & list unpacking?
26. How to add a new element to the existing list?
27. What is the diff b/w append() and extend()?
28. How to add a new element at required place in the list?
29. Can we add a sub list to the existing list object?
30. Can we add multiple elements by using append()?
31. What is difference b/w remove() & pop()

32 How to know the highest value in the list? Python Narayana

33 How to convert a list into string? 577

34 How to convert a string into list?

35 How to remove all elements from the list?

36 Can we delete a specific element by using del command?

37 What are the different ways to create a list?

38 What is the range() and what is the syntax of

range()?

39 What is the mandatory arg in the range()?

40 What is the difference b/w python2 and Python 3 in

case of range()?

41 What is the datatype of range() in python3?

42 What is the a tuple?

43 What is the difference b/w tuple & list?

44 Is tuple mutable or immutable object?

45 Is insertion order of tuple elements preserved

or not?

46 Can we give duplicate elements in the tuple?

47 What is all() & any()?

48. How to create a tuple with single element?

49. Can we create a specific tuple using () and also  
tuple()  
578

50. Can we delete a specific element by using clear()  
in the tuple?

51. How to convert a tuple into list? & list into tuple?

52. When to convert do choose a tuple & when do  
we choose a list?

53. Can we use a list in the tuple or not?

54. Can we use tuple in the list or not?

55. How to find sum of all elements in the tuple?

56. Can we use heterogeneous elements in the tuple?

57. What is the set?

58. Is a set allow duplicate elements or not?

59. Is insertion order of set elements is preserved or not?

60. Is a set dynamic object or not?

61. Is a set mutable or immutable object?

62. How to create an empty set?

63. How to represent an empty set?

64. How to add an element to the existing set?

python Narayana

65. Can we add a duplicate element to the existing  
for set?

(679)

66. What is the difference between remove() & discard()?

67. What is the main purpose of set?

68. What is the main purpose of tuple?

69. A List has so many duplicate elements, how to  
remove all duplicate elements from the List?

70. What are the operators that we use to perform  
membership operations?

71. What is the difference between is superset() and  
issubset()?

72. In which case both is superset() and issubset() will  
return True?

73. In which case both isSuperset() and isSubset() will  
return False?

74. If both sets are having completely uncommon elements  
then what is the result of isdisjoint()?

75. What are the different set operations that we  
perform in set?

76. What is the difference b/w list & set?

python Narayana

77. How to convert a set into list? Python Narayana
78. How to convert a set into string?
79. What is dict?
80. What is the main purpose of dict?
81. How to update the values of dict?
82. How to retrieve the values by using keys?
83. How to remove a specific key: value pair?
84. What is the difference b/w pop() and popitem()?
85. How to get all keys for dict?
86. How to update the dict by using another dict?
87. How to get all key: value pairs as items?
88. How to get all values from dict?
89. How to create a dict by using tuple?
90. How to remove all key: value pairs from the dict?
91. Can we use a list or tuple in the set?
92. Can we use a set or list or tuple?
93. Can we convert a heterogeneous list into string?
94. Can we split a list by using split()?
95. How to reverse a tuple()?
96. How to display the descending order of tuple elements?

(580)

q7. Is a dict mutable or immutable object?

q8. Is a dict allow duplicate keys or not? (581)

q9. Can we update keys in the dict?

## Part-2

1. what are the identity operators & membership operators.

2. what is the difference b/w between left shift & bitwise right shift operator.

3. what is diff b/w actual argument & formal arguments?

4. what are positional arguments?

5. what is keyword arguments and non-keyword arguments?

6. what is the difference b/w \*args and \*\*kwargs?

7. what is enumerate()?

8. what is the different b/w def function & Lambda function?

9. what are different types of function which are used for Lambda function?

10) what is filter() & purpose?

11) what is map() & purpose?

12. what is a reduce function() & purpose?
13. why do we use conditional st in our applications?
14. what is the difference b/w elif and nested if statement?
15. How to display 10th table by using for loop?
16. what is the difference b/w for loop and while loop?
17. How to iterate a list by using for loop?
18. what is continue and break statement?
19. what is the meaning of Comprehension?
20. what are the different types of comprehensions supported in python?
21. How to read data from text file?
22. How to get last line from the file?
23. how to count no. of file lines in the file?
24. How to count no. of words in the file?
25. How to get first word for all the lines in the file?
26. How to get specific page data from the file?
27. what is the difference b/w seek() & tell()?
28. what is the difference b/w readline() & readlines()?
29. what are the different types of modes available in the file operations?
30. what ~~is~~ is the default mode in the file operations?  
python Narayana

31. what is the difference b/w append(a) and write(w)  
modes?

(583)

32. what are the different oops concepts?

33. what is the data abstraction?

34. what is the inheritance? & types?

35. what type of inheritance you used in your last project?

36. what is Polymorphism?

37. what is the difference b/w method overloading & overriding?

38. Can we implement method overloading Completely in

python?

39. what is the class & object?

40. what are the different types of variables in python?

41. what is a module? type of modules?

42. List out some 3rd party modules?

43. How to install 3rd party module in python?

44. what is pip? & purpose of pip?

45. what are the different ways of modules to  
import modules in python?

46. Can we use members of other modules, without importing  
the module?

python Narayana

47. what is an exception ? How to handle exception?
48. Can we implement user defined exception in python? 584
49. what is the difference b/w else & finally block.
50. what is the logging ? purpose of Logging.
51. what are different levels of Logging?
52. what is the default log level in the python?
53. what is the extension of Log file?
54. what is the iterator? and one example?
55. what is generator? and one eg?
56. what is decorator & one eg?
57. How to perform unit testing in python?
58. which modules we use to perform unit testing in python?
59. which database you used in your last project?
60. what is the module that we use to get data from Oracle database?
61. How did you connect to your database in your project?
62. What is the difference b/w Numpy arrays & python list?

63. Write a python function to swap two numbers?
64. Write a python function to check whether number is  
585  
prime or not?
65. Write a python function to check whether number  
is palindrome or not?

Python  
Narayana

(586)

Python Narayana