Object Oriented Programming

BE(CSE) II-Semester

S. Durga Devi ,CSE,CBIT

Course Objectives:

1. Describe the principles of Object-Oriented Programming.
2. Enable the students to solve problems using OOPs features.
3. Handling of coding errors in programs and files.
4. Use of library modules to develop GUI applications.

Course Outcomes:

At the end of the course, students will be able to:

1. Understand the concepts Object-Oriented Programming Languages.
2. Adequately use the constructs like selection, repetition, functions, aggregated data .
3. Develop applications in modular approach with classes/modules.
4. Identify and rectify coding errors in a program.
5. Build packages for the simple real world problems.
6. Use library software in building graphical interface, mathematical software

Syllabus

Unit-I

Introduction to Object-Oriented Programming: Computer Programming and Programming Languages, Generations of Programming Languages, Programming Paradigms, Features of Object-Oriented Programming, Merits and Demerits of OOPs

Basics of Python Programming: Features of Python, Variables, Identifiers, Datatypes, Input/ Output operations, Operators and Expressions, operations on strings, Type conversion.

Unit-II

Decision Control Statement: Selection/Conditional Branching, Loop Control Structures, Nested loops.

Functions and Modules: Uses of functions, Function definition, function call, Variable scope and Lifetime,Recursion, Lambda functions, Recursive Functions, Modules, Packages.

Unit-III

Classes and Objects: Introduction, Classes and Objects, __init__ method, Class variables, and Object variables, Public and Private Data members , calling methods from other methods, built-in class attributes, garbage collection, class methods, static methods.

Unit-IV

Inheritance: Introduction, Inheriting classes, Polymorphism and method overloading, Composition or Containerships, Abstract classes and inheritance.

Operator Overloading: Introduction, Implementation of Operator Overloading, Overriding.

File Handling: File types, opening and closing files, reading and writing files, file positions.

Unit-V

Error and Exception Handling: Introduction, to errors and exceptions, Handling Exceptions Simple GUI Programming with tkinter package, Sample Graphics using Turtle, Plotting Graphs in Python.

Text Books

1.ReemaThareja "Python Programming", Oxford Press, 2017.

2. Mike McGrath "Python in easy steps: Makes Programming Fun", Kindle Edition, 2017.

References:

1. https://anandology.com/python-practice-book/object_oriented_programming.html
2. http://python-textbok.readthedocs.io/en/1.0/Object_Oriented_Programming.html
3. http://www.tutorialspoint.com/python/python_classes_objects.htm

Unit-1

Introduction to Object-Oriented Programming:

- Computer Programming and Programming Languages
- Generations of Programming Languages
- Programming Paradigms
- Features of Object-Oriented Programming
- Merits and Demerits of OOPs .

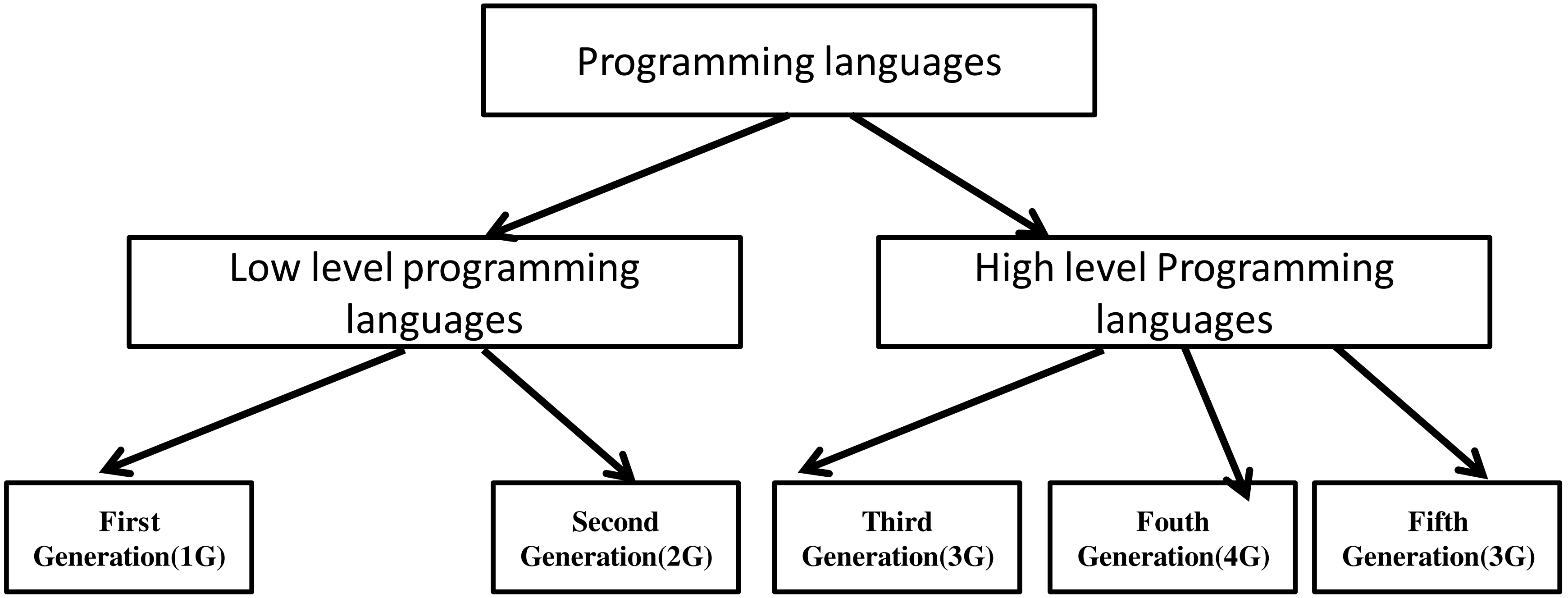
Basics of Python Programming:

- Features of Python, Variables, Identifiers, Datatypes
- Input/ Output operations
- Operators and Expressions
- operations on strings
- Type conversion

Computer programming and programming languages

- Computer program is a sequence of instructions written in some computer language to perform a particular task.
- The act of writing computer programs is called computer programming.
- Programming Languages: language is designed to implement an algorithm that can be performed by a computer to solve a problem.
- Programming languages are such as BASIC(Beginners All purpose Symbolic Instruction Code), C, C++, COBOL(Common Business Oriented Language), FORTRAN(Formula Translator), Python, Ada, and Pascal etc.
- Machine language: language which can understand only by computer consists of only numbers 1's and 0's.
- Assembly language: it is a low level language for the micro processor and other programmable devices. Symbolic names are used to represent the machine language. Used in device drivers and low level embedded systems.
- High level language: every human being can read this language like english statements. Ex: all programming languages.

Generations of programming languages



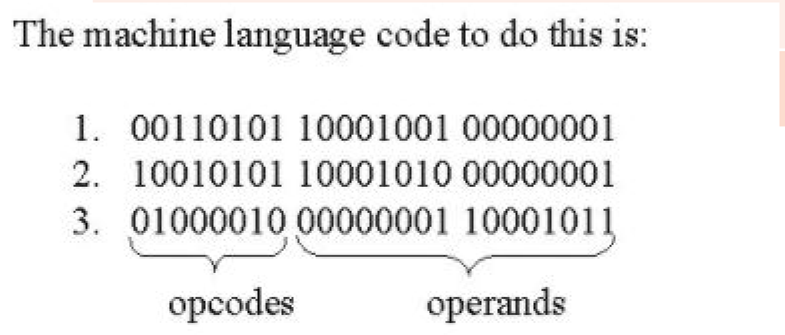
1. First generation (machine level language):

- machine language was used to program the first stored-program computer systems.(stores instructions in computer memory).
- In first generation programming instructions were entered through front panel switches.
- in 1950s, each computer had its own native language, and programmers had primitive systems for combining numbers to represent instructions such as add or subtract.
- in machine language, all instructions , memory locations, numbers and characters are represented in strings of 0's and 1s.

Front panel switches

advantages and disadvantages of machine language

Advantages	disadvantages
Code can be directly executed by the computer	Code is difficult to write
Execution is fast and efficient	Code is difficult to understand by other people
Programs can be written to efficiently utilize memory	There is possibility of error to creep in
	It is difficult to detect and correct errors
	Code is machine dependent and thus non-portable


The machine language code to do this is:

1. 00110101 10001001 00000001
2. 10010101 10001010 00000001
3. 01000010 00000001 10001011

opcodes operands

Adding two numbers in machine language

S. Durga Devi ,CSE,CBIT

2. Second generation (assembly language):

- it consists of series of statements and the statement consists of label, operation code and one or more operands.
- it needs assembler to convert code to machine language.
- ex MOV AX,4 ; MOV BX, 6 ; ADD AX,BX;
- Used in linux kernel and device drivers and also in video games.

Advantages	Disadvantages
It is easy to understand	Code is machine dependent and thus non portable
Easy to write in AL than ML	Programmers should have a good knowledge of the hardware internal architecture of the CPU
It is easy to detect and correct errors	The code cannot be directly executed by the computer, as it needs assembler to convert AL to ML
It is easy to modify	
It is less prone to errors	

3. Third generation (high level languages):

# these languages are machine independent languages and programmer friendly.

# C,C++,Java etc.

Advantages	Disadvantages
The code is machine independent	Code may not be optimized
It is easy to learn and use the language	The code is less efficient
They are portable	It is difficult to write a code that controls the CPU, memory and registers.
It is easy to document and understand the code	
It is easy to maintain the code	
It is easy to detect and correct errors	
Note: python ruby and perl are 3GL languages that combine some 4GL abilities within a general purpose 3GL environment.	

4. Fourth generation

characteristics of 4GL

1) the instructions of the code are written in english like sentences.
2) they are nonprocedural, so users concentrate on 'what' instead of 'how' aspect of the task.
3) the code written in a 4GL is easy to maintain.
4) the code written in 4GL become 10 times more productive that 3GL as fewer lines of code is written.

(ex: query language(SQL), which can be used to access the content of database using english like sentences). easier to learn than COBOL and C

to generate the report which displays the total number of students enrolled in each class and in each semester.

```
TABLE FILE ENROLMENT
SUM STUDENTS BY SEMESTER BY CLASS.
```

5) still evolving so difficult to define or standardize them.
6) it doesnot make efficient use of a machine's resources.

5. Fifth generation

1) centered on solving problems using constraints given to a program rather than using an algorithm.
2) most constraint based and logic programming languages and some declarative languages form a part of 5GL.
3) widely used in AI research and contains visual tools to help develop a program. ex: PROLOG, OPS5, Mercury , Visual Basic.
4) 5GL s were built upon LISP ex:ICAD. there are many frame languages such as KL-ONE

Programming Paradigms

- Paradigm means way of doing something(like programming).
- When you programming you need to follow some strategy that strategy is called programming paradigm.
- A programming paradigm is a style or way of programming that defines how the structure and elements of a program will be built.
- Or the way of writing computer program.
- There are different styles of programming paradigms.

1. Monolithic programming
2. Procedural programming/imperative paradigm
3. Structured programming/modular paradigm
4. Object oriented programming
5. Logic oriented programming
6. Rule oriented programming
7. Constraint oriented programming.

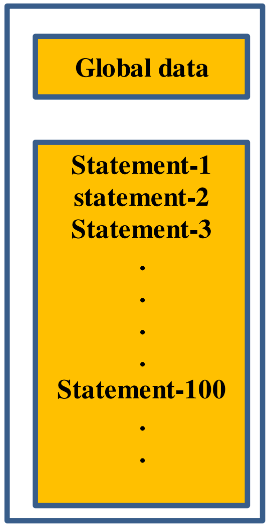
- Every paradigm has its own style of writing, capabilities and limitations

1. Monolithic programming paradigm:

- monolithic means a single block of statements.
- monolithic paradigm consists of sequence of statements uses global data.
- Useful for developing high level language (Basic) and assembly language (languages related to micro processors).
- Global data defined on top of the all the statements.
- Global data can be accessed by any sections of a program.
- How big the program all the statements should be in a same program
- There is no subroutines.(functions)
- We can jump from one section to another section within program using goto and jump instructions.

Drawbacks

- no security for the data
- since all statements embedded in single program, debugging is difficult.
- no reusability.



Monolithic paradigm Structure

2. Procedural programming

- Main program task is sub divided into sub task, each sub task implement as a separate procedure.

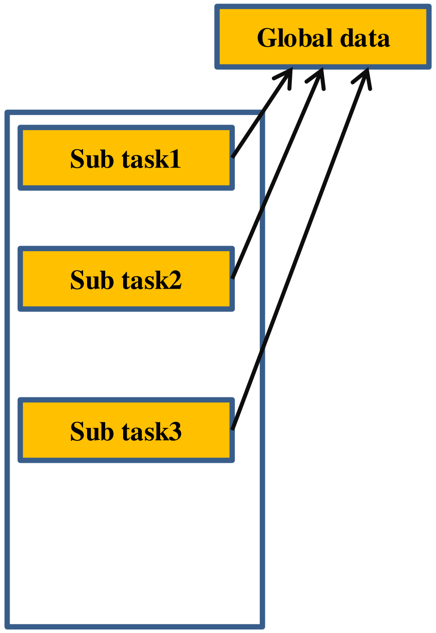
- Each procedure perform a specific task.
- Each procedure is also called as modules.
- The sequence of execution of instructions can be altered.
- There is no repetition of the code.
- Fortran an and cobol languages developed with this paradigm.

Advantages:

- Program can be controlled better than monolithic paradigm.
- Debugging is easy.
- Avoids repetition of the code.
- Provides reusability.

Disadvantages:

- Global data can be accessed by the all the sub tasks of a program so no security for the data.



Procedural paradigm Structure

3. Structured programming/modular programming:

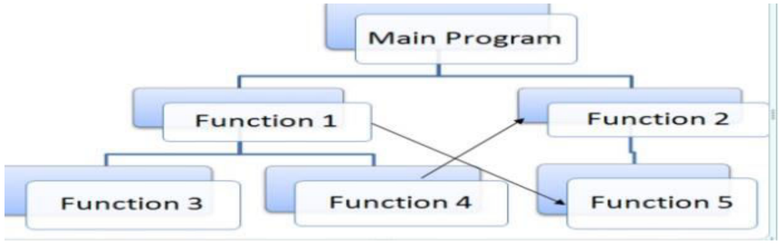
- -Entire program is divided into modules.
- Each module has set of functions each function performs single identifiable sub task.
- Each function has a local data and can also access global data if required.
- Local data of one function cannot access from another function.
- Follows the top down approach.
- Removes the goto statement is replaced with else if etc.
- Ex: C,C++,java, C#

Advantages:

- Larger programs are implemented easily.
- Debugging is easy.
- Security for the data as local data cannot access outside of the function.
- Reusability.
- Control of the project is easy
- Maintenance is easy

Disadvantages:

- Global data can be shared among the modules.
- Main focus on functions than the data



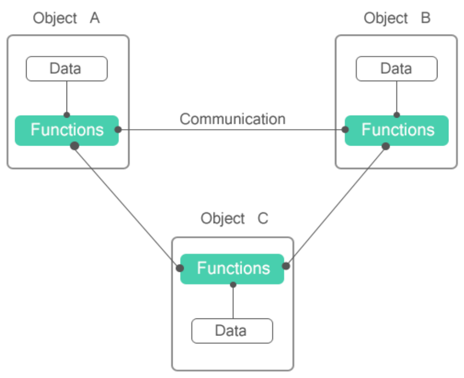
structured paradigm
Structure

4. Object oriented programming paradigm:

- Models real world objects
- Uses bottom up approach
- Problem is decomposed into objects and build the data and functions around these objects.
- Program organized around the objects , grouped into classes.
- Data of the objects can be accessed only with the function associated with that objects.
- Objects can communicate with each other
- Empasize more on data.
- Ex: c++,java, python, ruby and php.

Advantages:

- Provides security to data
- Reusability.
- Models the real world.



Object oriented paradigm Structure

Features of Oop

1. Class
2. Object
3. Data hiding
4. Data abstraction
5. Data encapsulation
6. Inheritance
7. Polymorphism
8. Containership or composition
9. Delegation
10. Method and message passing

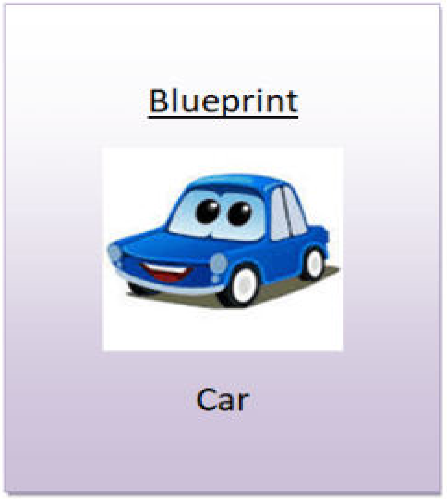
1. class: acts as blue print of an object which defines properties and methods

example: planning of a building is a class

each building is an object

2. Object: Physical existence of a class in computer memory is called object.

example: consider a person is a class. Every person has name, Gender, legs, hands etc. People are like an objects. They have properties specified in a person class.



Objects
Audi
Sports car

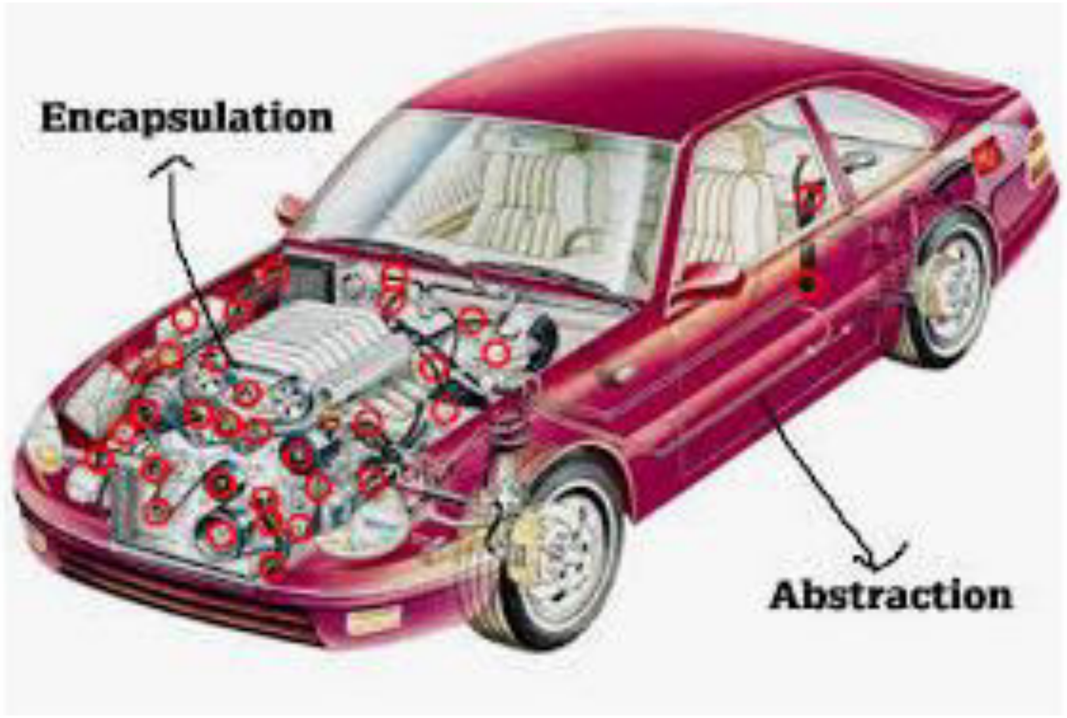
```
Class Car:
        color
        model
        noofwheels
        fueltype
        startengine()
        stopengine()
        drive()
        reverse()
```

3. Data hiding: hide data from others or outside world.

4. Data Abstraction: defines data and functions and hide implementation details is called data abstraction.

5. Data Encapsulation: the process of binding data and functions into a single unit is called data encapsulation.

Encapsulation= data hiding+data abstraction

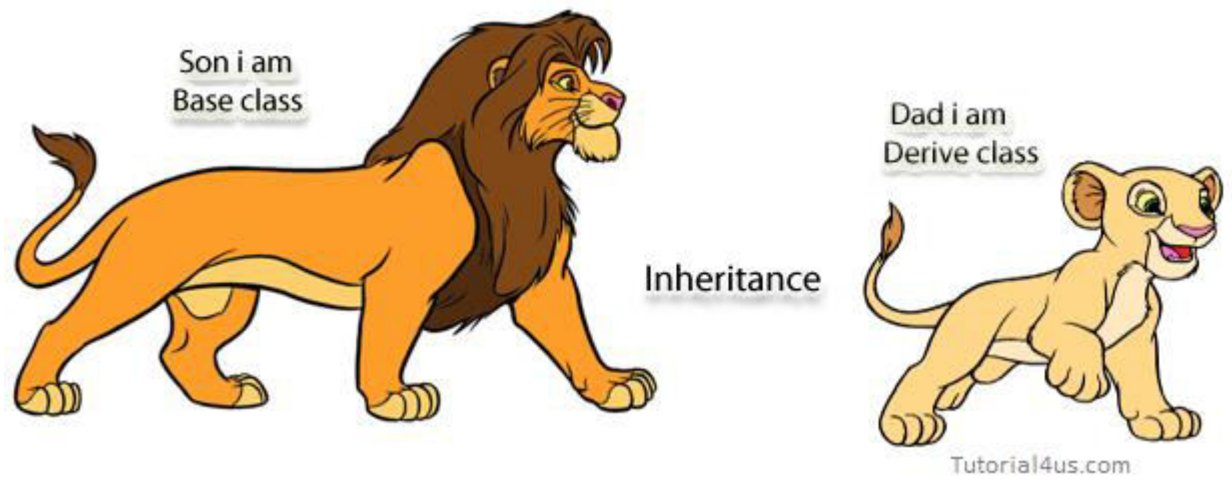


6. Inheritance:

- Also known as is a relationship.
- one class acquires features of (properties and methods) another class is called iheritance.

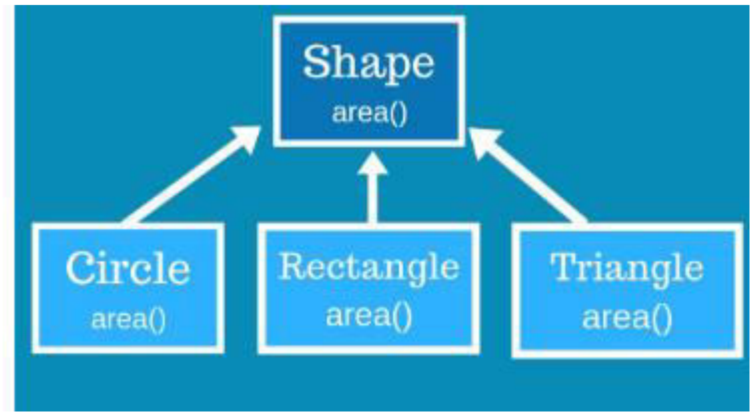
Advantages:

- Code reusability
- Save time and reduce project cost.



7. Polymorphism

- Having several forms
- Polymorphism is related to methods.



8. Containership or composition

- A class contains object of another class is called containership or composition.
- Increases reusability.
- Containership specifies has a relationship.

Example

```
    class car:
          e=Engine()
   class engine:
                ----
               -----
```

S. Durga Devi ,CSE,CBIT

9. Delegation

- delegation is a relationship between objects where one object forward request to a method call to another method.
- Advantage is change the behavior of an object during run time.

10. Method and message passing

- method is a group of statements performs a specific task and returns to caller.
- message passing is a communication between objects
- message passing involves name of object, function and information to be send.
- message passing is like calling methods with object and may send some parameters

Merits and Demerits of Oops

Merits

1. Model real world objects.
2. Redundancy is eliminated through inheritance
3. Reduces development time due to reusability
4. Secure data through data abstraction and data encapsulation
5. Maintainability is easy.
6. Easy to use.

Demerits

1. Requires more resources
2. Applicable for large and complex programs
3. Requires more skills to learn and implement the objects.

Application of Oops

1. To create user interfaces such as design GUI for windows.
2. Client- server systems
3. To create data bases
4. Artificial interlligence
5. Automation systems
6. Network programming, routers, and firewalls
7. Computer aided design systems

Introduction to Python

- Python is a high level programming and scripting language.
- Developed by Guido Van Rossom, who is father of python in 1989 in nehterland.
- In 1991 officially available to the world.

Why this language got the name python?

- There was a popular fun show 'monty Python Circus ' in BBC TV channel during 1969 to 1974. Python is also fun to learn that's why author named this language as Python.

Python is borrowed from other languages

1. Functional programming from C
2. Oop from C++
3. Scripting languages features from perl and shell script
4. Modular programming features from Modula-3.

Features of python

1. Simple and easy to learn
2. Free ware and open source
3. High level programming language
4. Platform independent
5. Portability
6. Dynamically typed language
8. Both procedure oriented and object oriented
9. Interpreted
10. Extensible – we can use other programming languages in python
11. Embedded - we can use python programs in another languages.
12. Extensive libraries in python.

1. Simple and easy to learn

- ➢ Python is a simple programming language as we can read python statements like a english language.
- ➢ We can write python programs with less code. Hence more readability and simplicity.
- ➢ Syntaxes of python are very easy to learn and

2. Freeware and open source

- ➢ we can use python software without any license and it is available free of cost.
- ➢ Source code of python is open to all that code can be customized as per our requirements.
- ➢ Ex: linux is open source it is customized to different versions like min, redhat, unix etc.

3. High level programming language

- ➢ Every human being can understand . Programmer need not aware of memory management and security etc.

4. Platform independent

- ➢ A python program can run on any operating system without rewriting agrain.
- ➢ Write once and run on any operating system.

5. Portability

- ➢ Python program can move form one platform to another platform without effecting performance and changes.

6. Dynamically typed language

- ➢ Python does not require any variable declaration. Based on type of value assigned to variable python will allocate the memory automatically.

7. Both procedure oriented and object oriented

- ➢ Python barrowed its syntaxes from c, and java so it supports both procedure and object oriented features.

8. Interpreted

- ➢ Line by line interprets the code need not compile it.

9. Extensible

- ➢ Python is extensible to any language
- ➢ We can use other language programs in python
- ➢ We can bring other languages functionalities and benefits into python where performance is required.

10. Embedded

- ➢ We can use python in any other languages .
- ➢ Python we can use in java called Jpython

11. Extensive library:

- ➢ python provides rich set of inbuilt libraries.

- ➢ Limitations of python

1. Due to interpreter python performance wise not up to the mark

2. Not used for mobile applications.

Applications of Python

1. Desktop applications (calculators, Excel etc) standalone applications
2. Web Applications (Django in python to develop web applications)
3. Database applications.
4. Machine learning
5. Networking applications
6. Games
7. Data analysis
8. Artificial Intelligence
9. IOT applications
10. Mobile based applications.

Currently following companies using python

- Google, Youtube, Dropbox, NASA, stock exchange, national security.

Flavours of python

1. Cpython (C)
2. Jython/Jpython (java)
3. IronPython (developed with C# platform)
4. PyPy (Python for Performance added Just in time compilers – python)
5. Ruby python – (Ruby)
6. AnacondaPython- (to handle big data applications and handles large volume of data for processing)
7. Stackless (Python for concurrency threading concepts)

- Latest version of python is 3.7.2 download from https://www.python.org/downloads/

Using python interpreter

Python execution in ubuntu

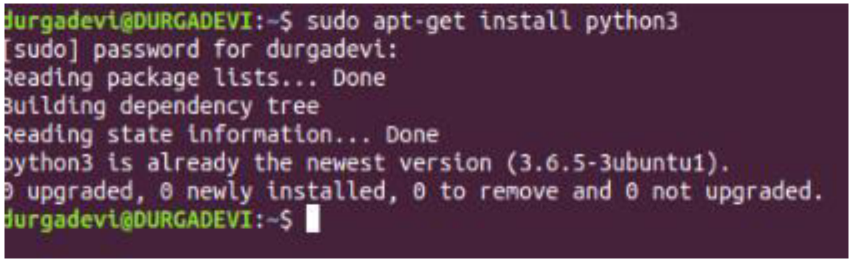
Python programs are executed in two modes

1. Interactive mode   2. scripting mode

Step1   open terminal

Step2   install python3

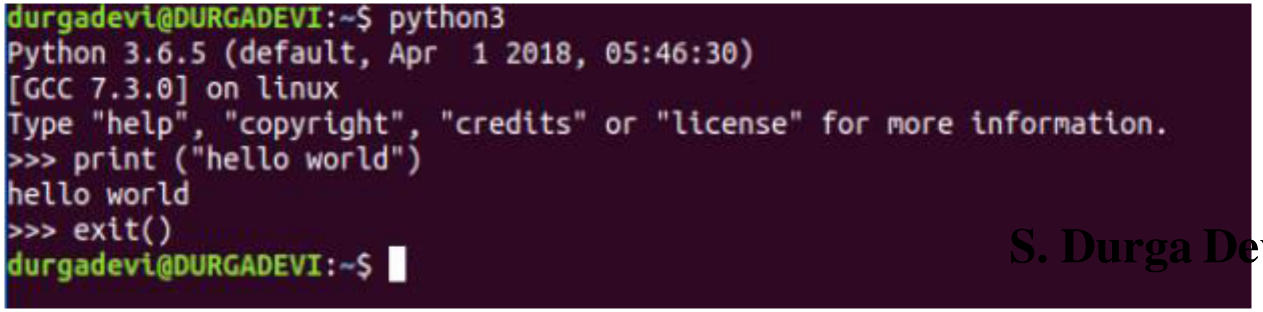
sudo apt-get install python3



```
durgadevi@DURGADEVI:~$ sudo apt-get install python3
[sudo] password for durgadevi:
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.6.5-3ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
durgadevi@DURGADEVI:~$
```

Step3 **execute in interactive mode**

type python3



```
durgadevi@DURGADEVI:~$ python3
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("hello world")
hello world
>>> exit()
durgadevi@DURGADEVI:~$
```

Step 4 to exit interactive mode

type exit()

2. Scripting mode

1. Open text editor and save python programs with .py extension.

```
#!/usr/bin/python3
#hello.py
print (“hello world”)
```

2. Go to terminal
3. Go to your directory where you saved your python programs
4. Execute the python program by giving following command

```
./hello.py    or python    hello.py
```

Python interpreter in windows

Step1 download python IDE from https://www.python.org/downloads/

Step2 install it

Step3 go to start button and type python click on the python IDE

Step4   same as your ubuntu

Variables, Identifiers, Datatypes

Variables: reserved memory location where values stored

Identifiers: names used to identify a variable, function, class.

Rules to follow for identifiers

- Identifiers can be in lower case or uppercase
- Digits can be used but should not start with digit
- Underscore is allowed
- Keywords not defined as identifiers
- Identifiers are case sensitive

  cash is different from Cash
- Identifiers should not start with $, #,% and numbers(0-9)

Literal constants: the values are defined directly in expression without assigning to variable is called literal constant.

ex 7+5- 7 and 5 are literal constants

Note: identifiers starts with __ two underscore symbols are also valid.

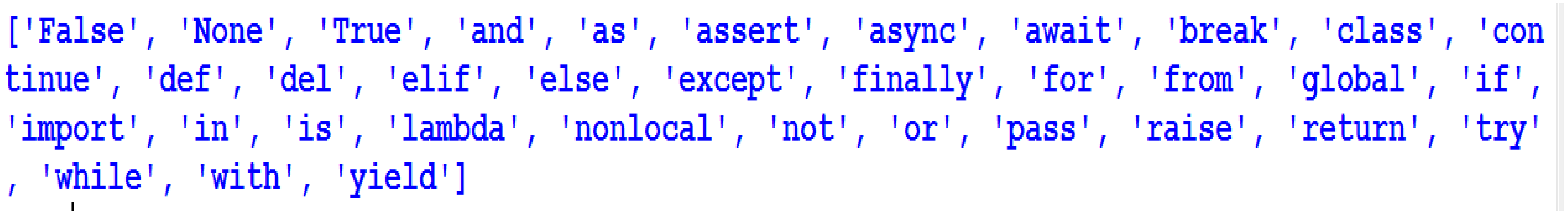
which of the following are valid identifiers

1. $total # invalid
2. 1python # invalid
3. python1 # valid
4. main # invalid as main is a keyword\
5. python_ex #valid
6. __x # valid

reserved words or key words

- Words which are defined by python has some specific meaning and functionality are called reserved words or key words.
- Python provides total 33 keywords
- All keywords in lower case except True,False and None keywords.
- To find total keywords present in python run the following statements

```
import keyword
keyword.kwlist
```



```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'con
tinue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try'
, 'while', 'with', 'yield']
```

Comments in python

- # used to comment single line
- '" triple quoted string is used to comment multiple lines

  example

'"

this is a python programming

'"

Data types in python

Python does not require to specify the type explicitly. Based on value provide, the type will be assigned automatically. Hence python is called dynamically typed language.

- There are 14 data types in python

1. int
2. float
3. str
4. complex
5. bool
6. bytes
7. bytesarray
8. range
9. list
10. tuple
11. set
12. dict
13. frozenset
14. None

int, float, bool, complex and str are fundamental data types in python

built in functions in python

Several built in functions in python few are

1. type(): gives type of the variable
2. id(): address of a variable
3. print(): prints a value

1. int data type:

stores the integral values

Ex x=10

type(x) # <class 'int'>

➢ int values can also be represented in following ways

1. decimal(base-10)

a=10

2. binary form (base-2)

- represents numbers in 0 or 1
- binary numbers prefix with 0B or 0b

Ex a=0B0001111/0b0001111

3. Octal(base-8)

- represents numbers from 0 to 7
- numbers prefix with zero followed by Big oh(O) or small oh(o)

Ex x=0O3456 or 0o234

4. hexadecimal(base-16)

- represents numbers from 0 to 15
- numbers prefix with zero followed by capital X or smal x

x= 0Xa1ab23 or 0xbb1122

- To convert one base to another python provides built in functions

1. bin(): converts any base to binary number

   bin(15) # '0b1111‘

2. oct(): converts any base to octal
3. hex(): converts any bae to hexadecimal.

2. float data type:

- represents float values

ex f=12.89

type(f) # <class ‘float’>

- we can represent float values in exponential form also

f=1.2e3 #1.2×10^3

- we can also use capital E

3. Complex data type

to represent complex numbers

a+bj

x=2+30j

y=3+4.3j

- Complex data types has some built in attributes to retrieve real and imaginary part
- x=10+20j
- x.real #10
- x.imag # 20
- Note : for imaginary values only letter ‘j’ is to be used

4. bool data type:

- stores True or False

True stores value 1

False stores value 0

```
x=True  # <class ‘bool’>
y=False  # <class ‘bool’>
```

5. str data type:

- allows strings there is no char data type in python even single character treated as string data type
- a string can be defined with single quote or double quotes

Ex: x=‘Durga Devi’

x=“Durga Devi”

- triple single quotes(‘’’) or triple double quotes(“””) are used to represent multiple lines

Slicing of strings:

[:] operator is called slice operator used to retrieve part of the string

-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
D	U	R	G	A		D	E	V	I
0	1	2	3	4	5	6	7	8	9

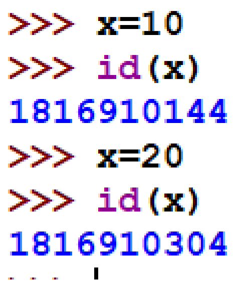
- s[1]#’U’
- s[7]#’E’
- s[-8]#’R’
- s[12]#IndexError as no index value 12
- s[6:]#’DEVI’
- s[1:]#’URGA DEVI’
- s[:8]#’DURGA DE’
- s[-9:-4]#’URGA’
- s*3# `'DURGA DEVIDURGA DEVIDURGA DEVI'`

All fundamental data types are immutable

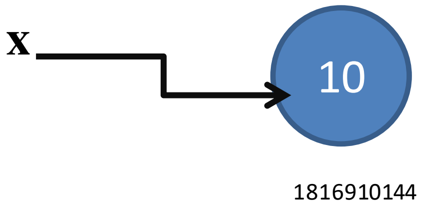
What is immutable?

Once you create an object , the value of the object cannot be changed. If you are trying to change t
a new object is created and old object will be destroyed is called immutable.

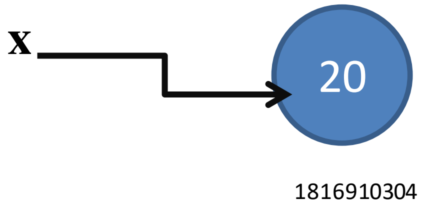
- All fundamental data types like int, float,bool,str,complex are immutable.
- Ex



```
>>> x=10
>>> id(x)
1816910144
>>> x=20
>>> id(x)
1816910304
```



When x=20 new object will be created



Hence, x is a immutable object

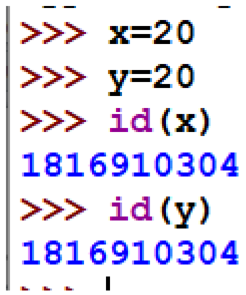
Advantage of immutable objects: performance and memory utilization is improved

S. Durga Devi ,CSE,CBIT

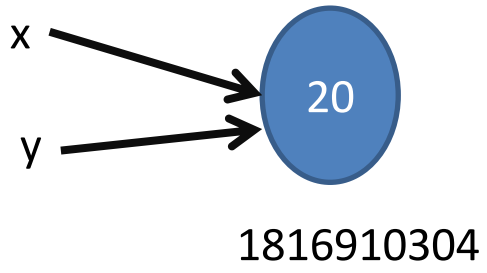
Reusable objects in python

-In python same object is used by multiple reference variable if all have same value that is called reusable objects.

Example



```
>>> x=20
>>> y=20
>>> id(x)
1816910304
>>> id(y)
1816910304
```



- Reusable objects concept not applicable to all the fundamental data types

- Int data type

```
>>> x=257
>>> y=257
>>> x is y
False
```

for integer object, objects will be reused only for the values (0-256)
only. Above 256 , new object is created.

-

- Float data type

```
>>> x=20.5
>>> y=20.5
>>> x is y
False
```

- for float data type reusable objects not supported

- Complex data type

```
>>> x=20+5j
>>> y=20+5j
>>> x is y
False
```

- for complex data type reusable objects not supported

S. Durga Devi ,CSE,CBIT

Input and output statements

- To read data dynamically from keyboard two methods are used

1. raw_input() # in python2
2. input() #python2 and python3

Example

```
#  add two numbers read them from keyboard
 x=int(input("enter x value");
y= int(intput("ente y value");
Print ("adding two numbers=", x+y)
```

Read multiple values from the keyboard at a time

- Use split method

Example

```
   a,b,c=[int(x) for x in input(“Enter two numbers”).split()]
                  or
#we can also use eval() function
A,b,c=[eval(x) for x in input(“enter two numbers”).split()]
 print (“the sum is=“, a+b+c)
```

Multiple assignments in python

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

X,Y,Z=2.3,5, True

a=b=c=20
```

print() method

- print() method is used to output the data

```
Example
 a,b,c=10,20,30
 print(“the values are”, a,b,c,sep=‘:’)

Output  10:20:30
```

here sep is an attribute used to separate value with specified delimiter

If you don’t want to print every time in new line then you can use **end** attribute

```
print(“hello”, end=‘ ‘)
print(“python”, end=‘ ‘)
print(“you are great”, end=‘ ‘)
```

Output

```
hello python you are great
```

S. Durga Devi, CSE,CBIT

```
print(1,2,3,4)
# Output: 1 2 3 4
print(1,2,3,4,sep='*')
# Output: 1*2*3*4
print(1,2,3,4,sep='#',end='&')
# Output: 1#2#3#4&
```

Output formatted string

- Formatting strings are same as your c programming
- Basic format specifier are

%s : string as well as any object which can be represented in string like numbers.

%d/%i: integers

%f: float

%x/%X: integers in hexa decimal.

Output formatted string

- Formatting strings are same as your c programming

```
x=89.1234567
print("x value is %3.2f" %x)
a,b,c=10,20,30
print("a,b,c values are %i %i %i" %(a,b,c))
```

```
s="CBIT"  # string type
l=[10,20,30,40] # list type
print("Hello %s the list is %s"  %(s,l))

Output Hello CBIT the list is [10,20,30,40]
```

-you can replace format string using format function

- syntax **str.format(arguments)**

Example

```
name="SreeRam"
Rollno=99
branch="CSE"
print("Hello{0} your rollno is {1} and you belong to {2}
    branch".format(name,Rollno,branch))
```

Output

```
HelloSreeRam your rollno is 99 and you belong to CSE branch
```

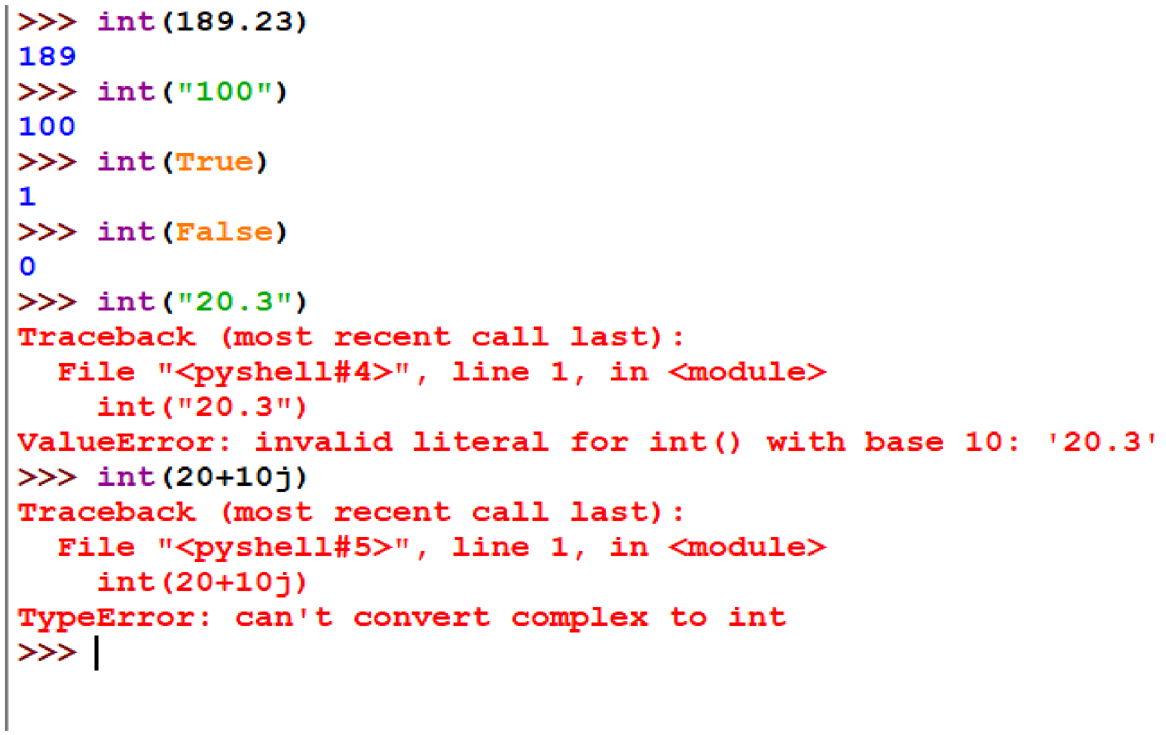
Type Casting

Type casting: converting one data type to another is called type casting or type coersion.

- Following are built in functions for type casting

1. int():

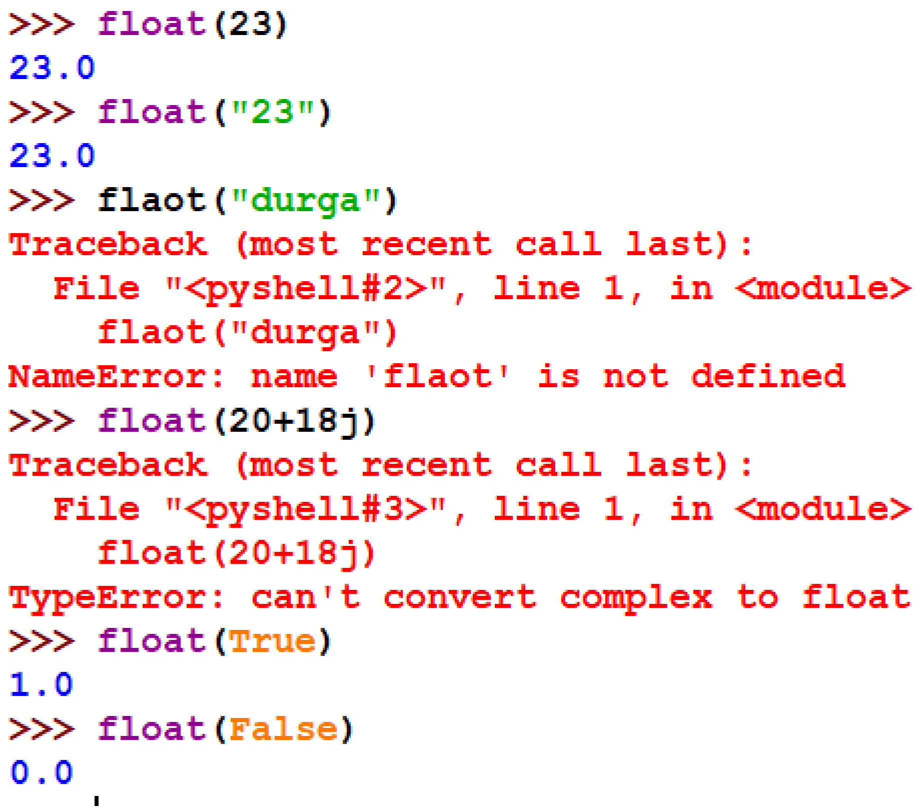
- We can convert any type to int except the complex type.
- If str has base -10, str can also be converted into the int type



```
>>> int(189.23)
189
>>> int("100")
100
>>> int(True)
1
>>> int(False)
0
>>> int("20.3")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    int("20.3")
ValueError: invalid literal for int() with base 10: '20.3'
>>> int(20+10j)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    int(20+10j)
TypeError: can't convert complex to int
>>> 
```

S. Durga Devi ,CSE,CBIT

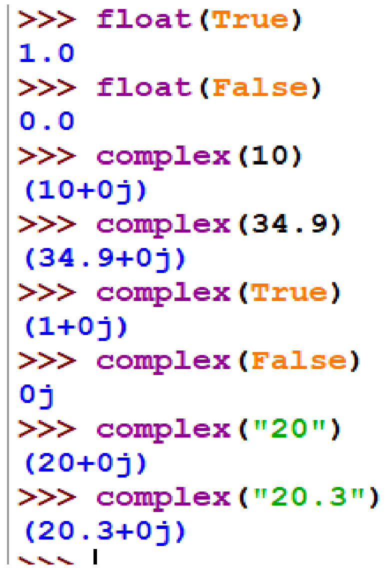
2. float(): converts other type value to float value



```
>>> float(23)
23.0
>>> float("23")
23.0
>>> flaot("durga")
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    flaot("durga")
NameError: name 'flaot' is not defined
>>> float(20+18j)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    float(20+18j)
TypeError: can't convert complex to float
>>> float(True)
1.0
>>> float(False)
0.0
```

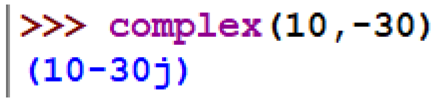
Note: complex and string with no base-10 type cannot convert to float

3. complex(x): converts the x into complex real part and imaginary part is zero default



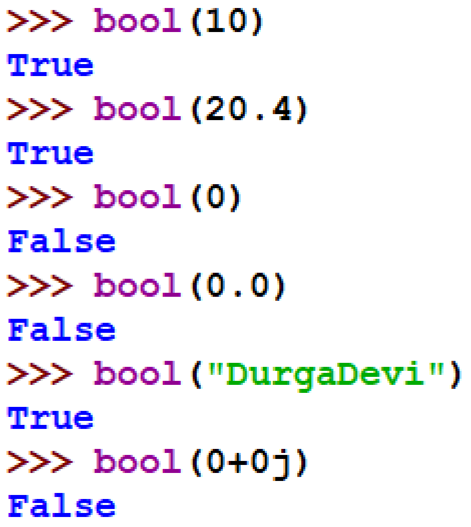
```
>>> float(True)
1.0
>>> float(False)
0.0
>>> complex(10)
(10+0j)
>>> complex(34.9)
(34.9+0j)
>>> complex(True)
(1+0j)
>>> complex(False)
0j
>>> complex("20")
(20+0j)
>>> complex("20.3")
(20.3+0j)
```

- complex(x,y): converts x into real part and y in imaginary part



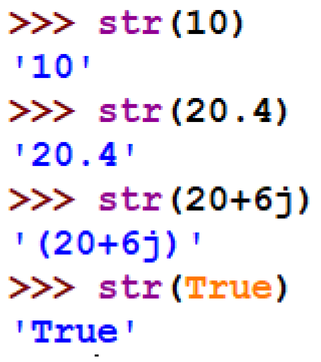
```
>>> complex(10,-30)
(10-30j)
```

4. bool(): converts to bool type value



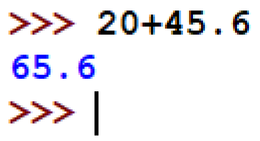
```
>>> bool(10)
True
>>> bool(20.4)
True
>>> bool(0)
False
>>> bool(0.0)
False
>>> bool("DurgaDevi")
True
>>> bool(0+0j)
False
```

5. str(): convert to string



```
>>> str(10)
'10'
>>> str(20.4)
'20.4'
>>> str(20+6j)
'(20+6j)'
>>> str(True)
'True'
```

Type coersion: convert type to other type implicitly during the compile or run time

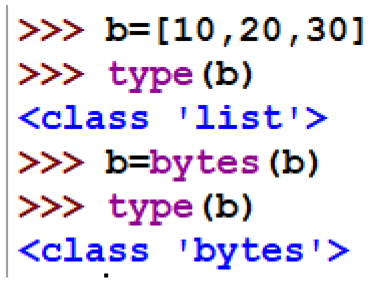


```
>>> 20+45.6
65.6
>>> 
```

- range() method
- List, tuple and set.

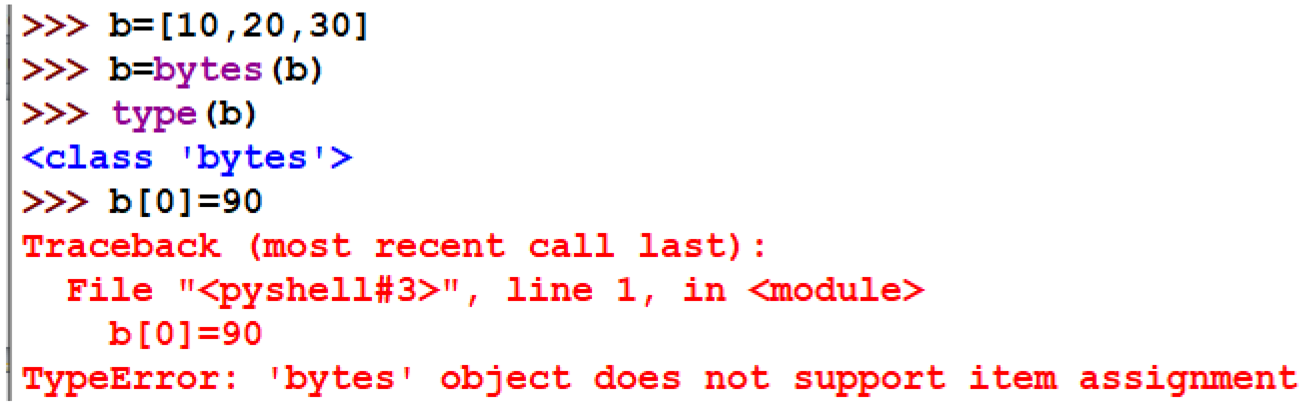
6. byte data type: represents byte numbers

- Use built in function bytes() to convert into byte data type
- Allows only numbers from 0 to 255 and is immutable.



```
>>> b=[10,20,30]
>>> type(b)
<class 'list'>
>>> b=bytes(b)
>>> type(b)
<class 'bytes'>
```

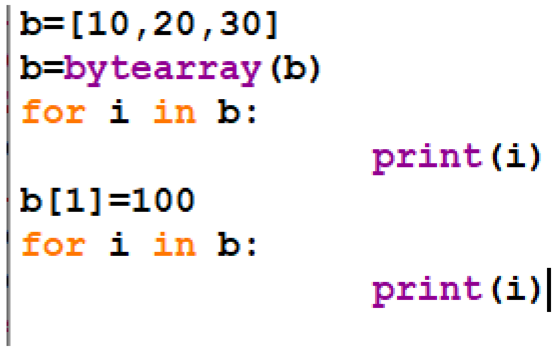
- Once byte data type created value cannot be changed.



```
>>> b=[10,20,30]
>>> b=bytes(b)
>>> type(b)
<class 'bytes'>
>>> b[0]=90
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    b[0]=90
TypeError: 'bytes' object does not support item assignment
```

7. bytesarray:

- same as bytes type but in bytesarray type values can be changed.
- Use method bytearray()



```
b=[10,20,30]
b=bytearray(b)
for i in b:
                print(i)
b[1]=100
for i in b:
                print(i)
```

```
Output
10
20
30
100
20
30
```

-Byte and bytearray data types are used to represent binary data such as images, video files etc. and also in computer networks.

8. range(): built in method used to generate sequence

syntax: range(1,n): generate number from 1 to n-1

range(n): generate 0 to n-1

Ex:

```
for  x in range(1,10):
            print(x)# prints 1 to 9
 for x in range(10):
            print(x) # prints0 to 9
```

9. List data type

- Insertion order is preserved
- Duplicate values are allowed
- List is mutable – elements can be modified.
- Null values are not allowed in the list
- Stores any type of value
- List of values enclosed in [ ]
- to access list elements we can use either +ve or –ve index
- Example
- list=[10,20,30,40,'b']

- Example
- type(list)

  output is <class 'list'>
- index value of list starts with 0
- ex list[0] gives 10

Build in methods in list data type

1. list.append(x) : append value at end of the list
2. list.insert(i, x): insert element x in specified position i
3. list.remove(x) : removes first item matched with value x else returns error.
4. list.pop(i): removes item from specified position and returns the removed item.
5. list.count(x): returns no of times the elements appeared
6. list.clear() : removes all the elements from the list.
7. list.extend(x): elements of x added to end of the list object.

https://www.geeksforgeeks.org/list-methods-python/

Example on list

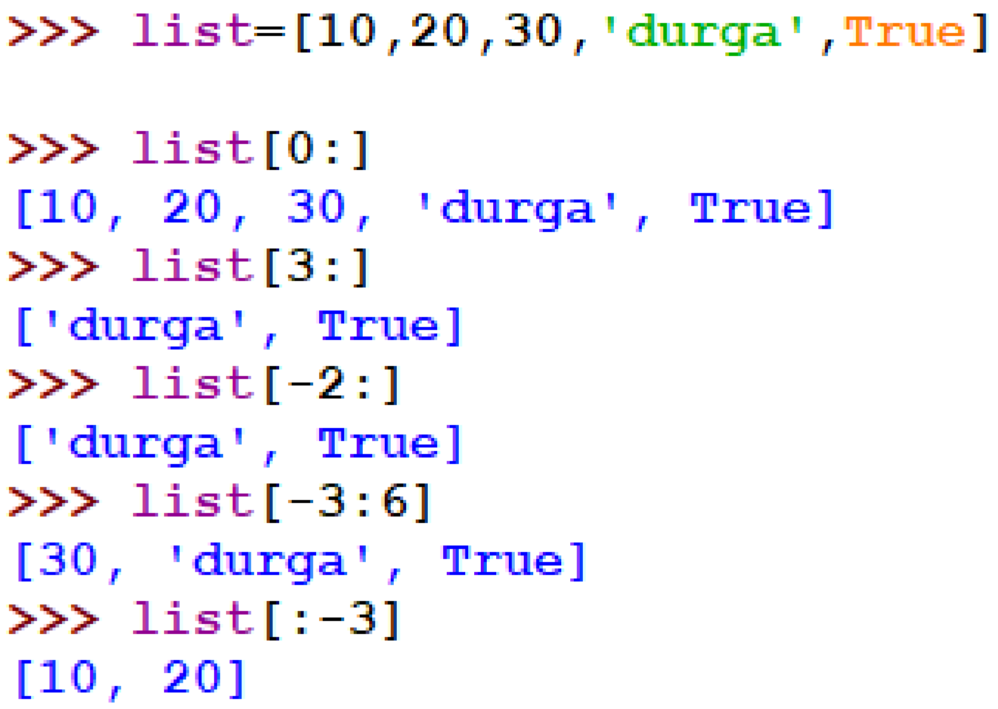
```
list=[10,20,30,'durga',True]
print(list)
```

```
Output  [10, 20, 30, 'durga', True]
```

- Example
- # display list elements repeated number of time

```
 s=[10,20,30,’durga’,True]
s1=s*2;   # * is a repetition operator
 print(s1)
```

Slice operator on list



```
>>> list=[10,20,30,'durga',True]

>>> list[0:]
[10, 20, 30, 'durga', True]
>>> list[3:]
['durga', True]
>>> list[-2:]
['durga', True]
>>> list[-3:6]
[30, 'durga', True]
>>> list[:-3]
[10, 20]
```

10. tuple data type

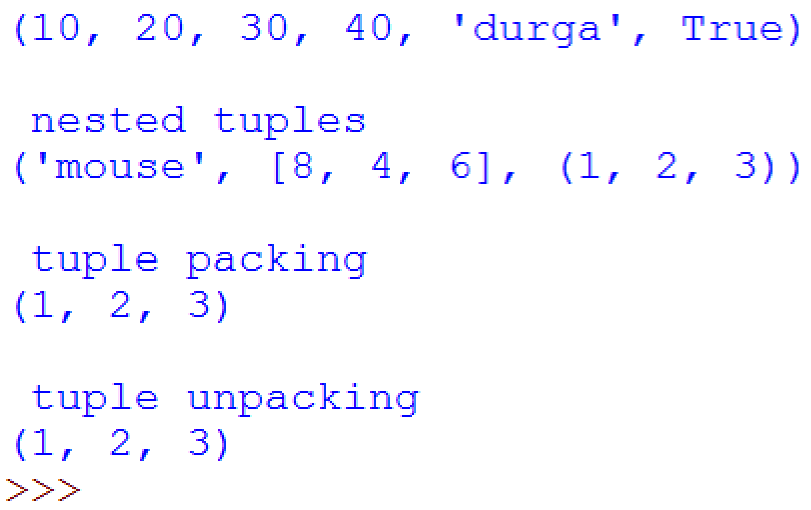
- tuple is immutable whereas list is mutable.
- Preserves insertion order.
- Duplicates values allowed.
- Tuple is represented in rounded parenthesis
- t=(10,20,30,40,’durga’,True)
- Null values allowed.
- Allows multiple data values.

Advantages of tuple

1. Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
2. If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Creating tuples

```
t1=(10,20,30,40,'durga',True)
print(t1)
#nested tuple
print("\n nested tuples");
t2 = ("mouse", [8, 4, 6], (1, 2, 3))
print(t2)
# tuples can be created without parenthesis is called tuple packing
print("\n tuple packing")
t3=1,2,3
print(t3)
# tuples unpacking also possible
print("\n tuple unpacking")
x,y,z=t3
print(t3)
```



```
(10, 20, 30, 40, 'durga', True)

 nested tuples
('mouse', [8, 4, 6], (1, 2, 3))

 tuple packing
(1, 2, 3)

 tuple unpacking
(1, 2, 3)
>>>
```

- Accessing tuples

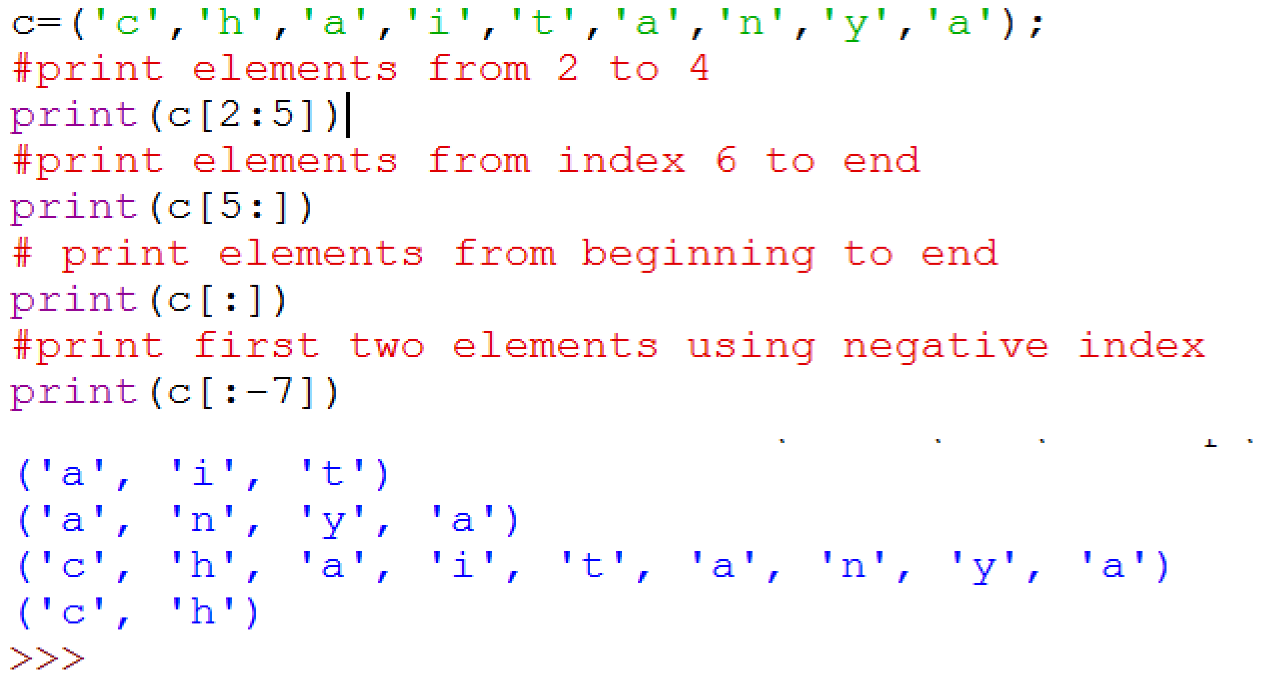
```
t2 = ("mouse", [8, 4, 6], (1, 2, 3))
print(t2[0][3]) # prints s
print(t2[1][2]) # prints 6
```

- Accessing elements through negative index allows to access from last element
- -1 indicates last element

```
t=('c','b','i','t‘)
print(t[-1]) # prints t
```

Slicing operator

- We can access range of elements in a tuple by using the slicing operator - colon ":".



```
c=('c','h','a','i','t','a','n','y','a');
#print elements from 2 to 4
print(c[2:5])
#print elements from index 6 to end
print(c[5:])
# print elements from beginning to end
print(c[:])
#print first two elements using negative index
print(c[:-7])
```

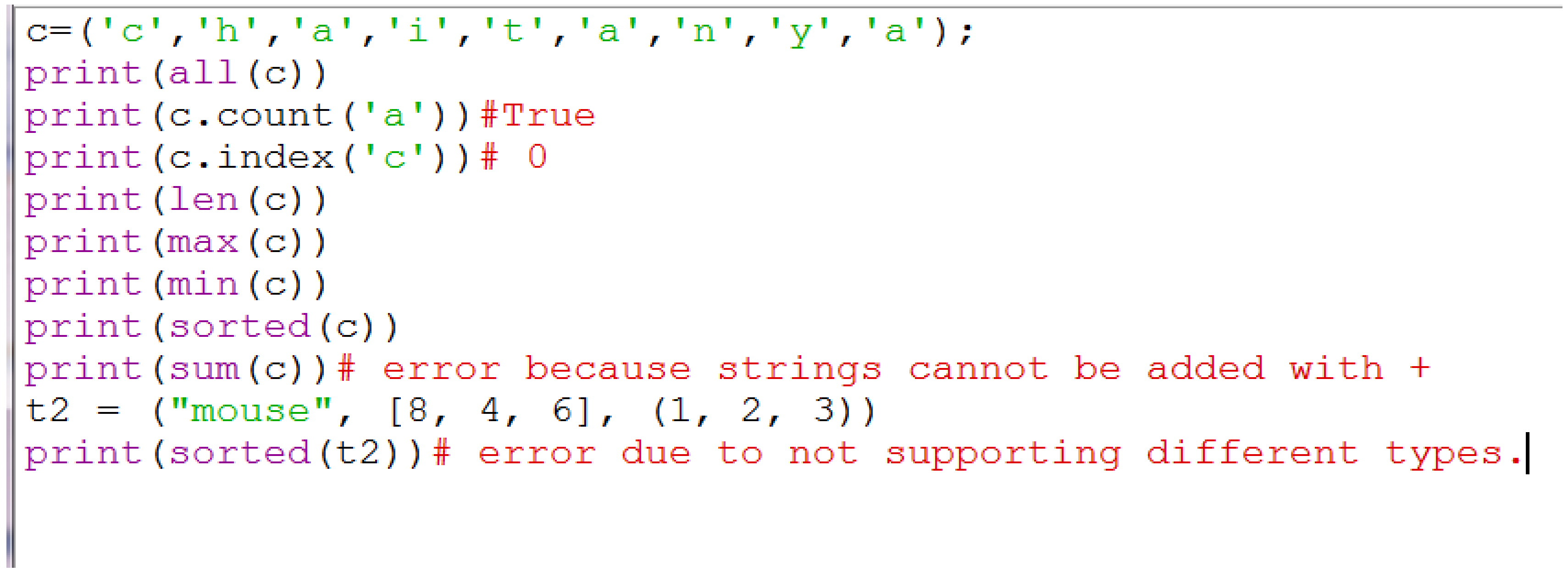
```
('a', 'i', 't')
('a', 'n', 'y', 'a')
('c', 'h', 'a', 'i', 't', 'a', 'n', 'y', 'a')
('c', 'h')
>>>
```

Built in methods in tuple

Assume t1 is a tuple object

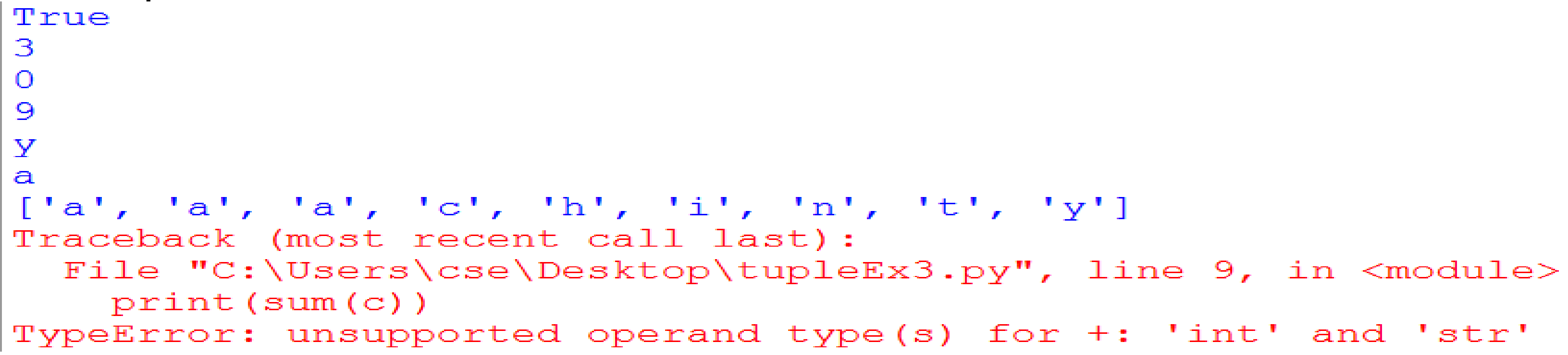
Method	Meaning
all	Return True if all elements of the tuple are true (or if the tuple is empty).-> all(t1)
count(x)	count no of elements that are equal to x t1.count(10)
index(x)	Finds index of a specified element x -> t1.index(10)
len(t1)	returns no of elements -> len(t1)
max(t1)	largest element in tuple
min(t1)	smallest element in tuple
sum(t1)	sum of all the elements of tuple
sorted(t1)	returns new sorted tuple (does not sort tuple itself) -> t2=sorted(t1)

example



```
c=('c','h','a','i','t','a','n','y','a');
print(all(c))
print(c.count('a'))#True
print(c.index('c'))# 0
print(len(c))
print(max(c))
print(min(c))
print(sorted(c))
print(sum(c))# error because strings cannot be added with +
t2 = ("mouse", [8, 4, 6], (1, 2, 3))
print(sorted(t2))# error due to not supporting different types.
```

output



```
True
3
0
9
y
a
['a', 'a', 'a', 'c', 'h', 'i', 'n', 't', 'y']
Traceback (most recent call last):
  File "C:\Users\cse\Desktop\tupleEx3.py", line 9, in <module>
    print(sum(c))
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

S. Durga Devi, CSE,CBIT

11. set data type

- Insertion Order does not preserved
- Duplicates not allowed
- Set will be represented in { }
- Does not support indexing
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary , as its element.

- Properties of set data type

1. no indexing
2. no slicing
3. no order
4. no duplicates
5. it is mutable
6. growable.
7. Allows mixed data types except mutable elements

```
s={1,2,3,"CBIT",(10,20,30)}
print(s)
```

Output

```
{1, 2, 3, 'CBIT', (10, 20, 30)}
```

To create empty set use set() function

```
x=set()
```

Methods in set

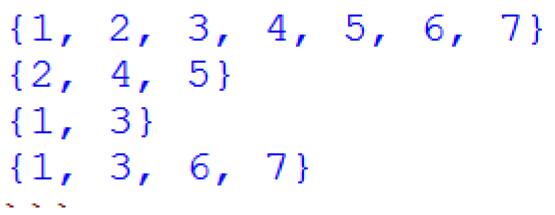
Method	Meaning
add(x)	Add element x s.add(10)
discard(x)	discard element x from set -> s.discard(10)
clear()	removes all the elements s.clear()
remove(x)	same as discard(), removes specified element s.remove(10)
pop()	removes first element and returns removed element
copy()	Copies elements
union	union of two sets as new set
difference	Difference of two sets as new set
symmetric_difference	symetric_ difference of two sets as new set

- Set can also carry out mathematical set operations like union, difference, symmetric difference and intersection.
- We can do with either operators or methods

Example

```
s1={1,2,3,4,5}
s2={2,4,5,6,7}
print(s1.union(s2))
print(s1.intersection(s2))
print(s1.difference(s2))
print(s1.symmetric_difference(s2))
```

output

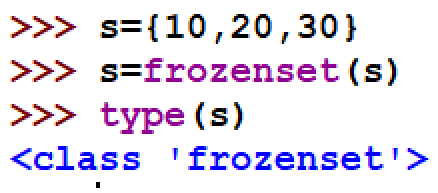


```
{1, 2, 3, 4, 5, 6, 7}
{2, 4, 5}
{1, 3}
{1, 3, 6, 7}
```

12. frozen set data type:

- same as set data type but it is immutable
- cannot use add or remove functions

Example:



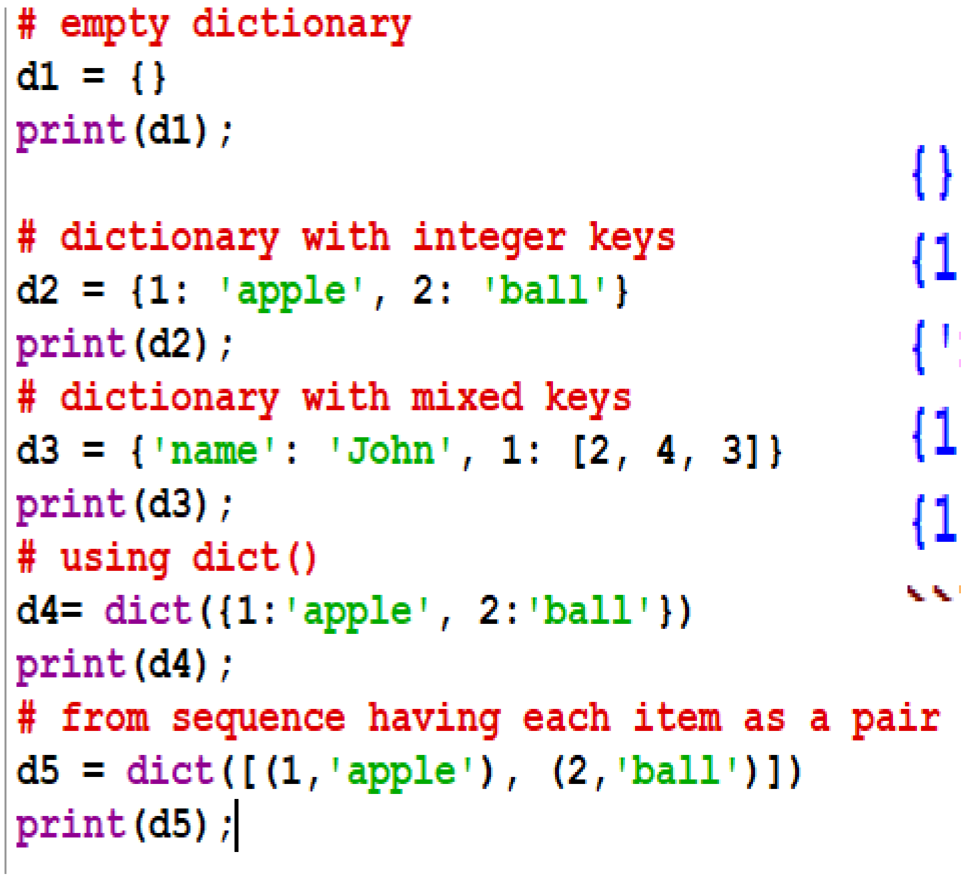
```
>>> s={10,20,30}
>>> s=frozenset(s)
>>> type(s)
<class 'frozenset'>
```

13.dict data type

- A dictionary has a values as a key : value pair
- Order does not preserve.
- No duplicate keys but values can be duplicate
- Values are accessed through key.
- Keys in dictionary should be immutable (string, number or tuple with immutable elements) and must be unique.
- Elements are placed in { } separated by comma
- Key and its corresponding value separated with :
- We can create a dictionary by using built in function **dict().**

- **Note:dict is mutable and order does not preserved.**

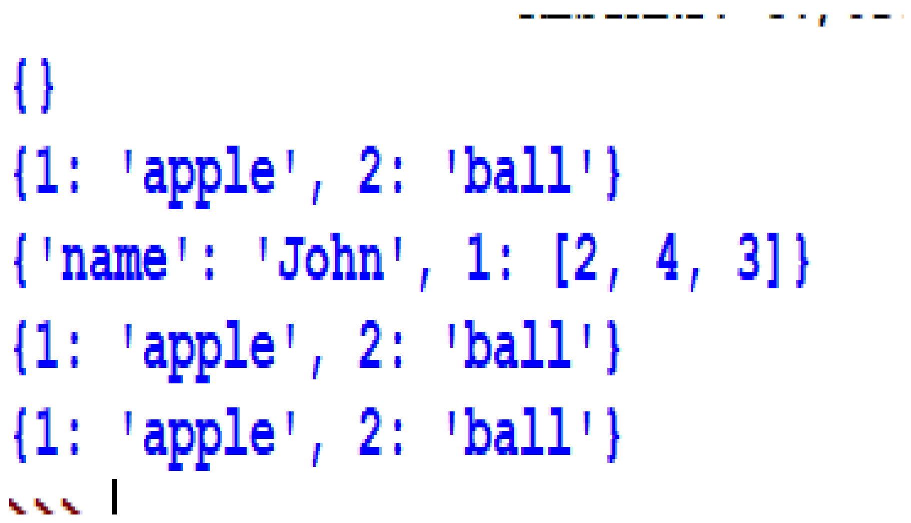
Example



```python
# empty dictionary
d1 = {}
print(d1);

# dictionary with integer keys
d2 = {1: 'apple', 2: 'ball'}
print(d2);
# dictionary with mixed keys
d3 = {'name': 'John', 1: [2, 4, 3]}
print(d3);
# using dict()
d4= dict({1:'apple', 2:'ball'})
print(d4);
# from sequence having each item as a pair
d5 = dict([(1,'apple'), (2,'ball')])
print(d5);
```

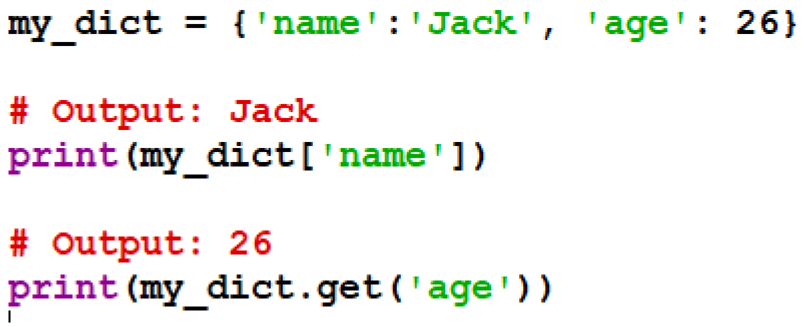
output



```
{}
{1: 'apple', 2: 'ball'}
{'name': 'John', 1: [2, 4, 3]}
{1: 'apple', 2: 'ball'}
{1: 'apple', 2: 'ball'}
```

Access dictionary elements

- Keys are used to access dictionary elements
- Key can be used either within the square brackets or with get() method.
- When get() method is used it returns none when key element is not found.



```
my_dict = {'name':'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))
```

```
Output
 Jack
  26
```

Change or add elements in dictionary

- Dictionaries are mutable so that we can add or change elements in dictionary

Example

```
my_dict = {'name':'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

https://www.programiz.com/python-programming/dictionary#what

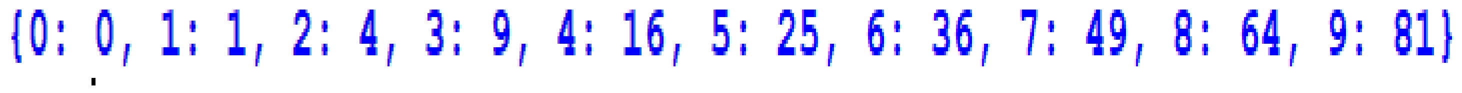
Remove elements from dictionary

- ***pop(key):*** deletes specified key value and returns the value.
- ***popitem():*** deletes and return arbitrary element.
- ***del is keyword*** used to delete particular element or entire dictionary.

Example

```
 store n number of square values
 sqaure={x:x*x for x in range(10)};
print(sqaure);
```

Output



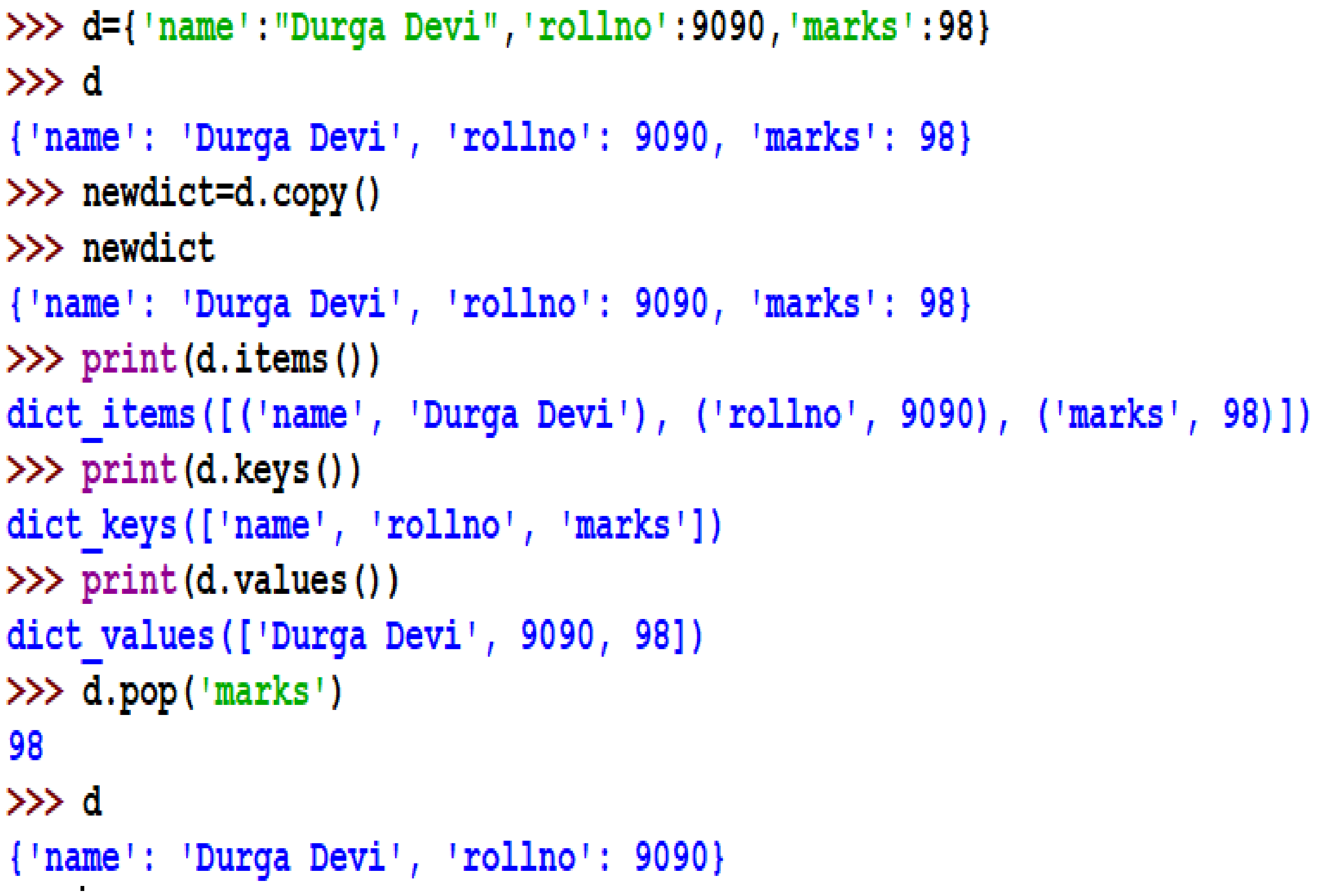
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

built in function in dict object

name	Purpose
copy()	Copies the dictionary values and return new dictionary
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
values()	Return a new view of the dictionary's values
pop(key):	deletes specified key value and returns the value.
popitem():	deletes and return arbitrary element.
del	delete particular element or entire dictionary
clear()	Removes all the elements

#Task: Practice all the methods of dict object and how to add and change dict values

example



```
>>> d={'name':"Durga Devi",'rollno':9090,'marks':98}
>>> d
{'name': 'Durga Devi', 'rollno': 9090, 'marks': 98}
>>> newdict=d.copy()
>>> newdict
{'name': 'Durga Devi', 'rollno': 9090, 'marks': 98}
>>> print(d.items())
dict_items([('name', 'Durga Devi'), ('rollno', 9090), ('marks', 98)])
>>> print(d.keys())
dict_keys(['name', 'rollno', 'marks'])
>>> print(d.values())
dict_values(['Durga Devi', 9090, 98])
>>> d.pop('marks')
98
>>> d
{'name': 'Durga Devi', 'rollno': 9090}
```

Two dimensional dictionary

Store students details key is student rollno.

S={121:{‘name’:’Ram’,’Branch’:’CSE’, ‘Section’:3,’marks’:100},

122:{‘name’:’SreeRam’,’Branch’:’ECE’, ‘Section’:1,’marks’:90}}

# to access above elements

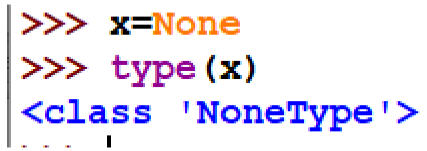
S[121]

S[121][‘name’] # ‘Ram’

14. None

- no value or nothing.

- If value is not available for particular object or a field None data type is used.
- Similar to null value in java.



```
>>> x=None
>>> type(x)
<class 'NoneType'>
```

summary of data types in python3

Datatype	Description	Is immutable	example

int	We can use to represent the whole/integral numbers	Immutable	>>> a=10
			>>> type(a)
			<class 'int'>

float	We can use to represent the decimal/floating point numbers	Immutable	>>> b=10.5
			>>> type(b)
			<class 'float'>

bool	We can use to represent the logical values(Only allowed values are True and False)	Immutable	>>> flag=True
			>>> flag=False
			>>> type(flag)
			<class 'bool'>

complex	We can use to represent the complex numbers	Immutable	>>> c=10+5j
			>>> type(c)
			<class 'complex'>
			>>> c.real
			10.0
			>>> c.imag
			5.0

str	represents sequence of characters	Immutable	>>> s='durga'
			>>> type(s)
			<class 'str'>
			>>> s="durga"
			>>> s='''Durga Devi
			CBIT
			... CSE''
			>>> type(s)
			<class 'str'>

Datatype	Description	Is immutable	example

bytes	To represent a sequence of byte values from 0-255	Immutable	>>> list=[1,2,3,4]
			>>> b=bytes(list)
			>>> type(b)
			<class 'bytes'>

bytearray	To represent a sequence of byte values from 0-255	Mutable	>>> list=[10,20,30]
			>>> ba=bytearray(list)
			>>> type(ba)
			<class 'bytearray'>

range	To represent a range of Values	Immutable	>>> r=range(10)
			>>> r1=range(0,10)
			>>> r2=range(0,10,2)

list	To represent an ordered collection of objects	mutable	>>> l=[10,11,12,13,14,15]
			>>> type(l)
			<class 'list'>

tuple	To represent an ordered collection of objects	Immutable	>>> t=(1,2,3,4,5)
			>>> type(t)
			<class 'tuple'>

set	To represent an unordered collection of unique objects	Mutable	>>> s={1,2,3,4,5,6}
			>>> type(s)
			<class 'set'>

frozenset	To represent an unordered collection of unique objects	Immutable	>>> s={11,2,3,'Durga',100, 'Ramu'}
			>>> fs=frozenset(s)
			>>> type(fs)
			<class 'frozenset'>

dict	To represent a group of key value pairs	Mutable	d={101:'durga',102:'ramu',103:'hari'} >>> type(d) <class 'dict'>

Escape characters

- Escape characters can be used in string literals

Characters	Meaning
\n	New line
\t	Horizontal tab
\r	Carriage return
\’	Single quote
\”	Double quotes

Operators and expressions

Operator is symbol that performs certain operations

Types of operators:

1. Arithmetic operators
2. Relational or comparison operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Special operators

1. arithmetic operators

a,b=10,20

operator	Description	Example	Output
+	Addition	print(a+b)	30
-	Subtraction	print(a-b)	-10
*	Multiplication	print(a*b)	200
/	Division gives output in float	print(a/b)	0.5
//	floor division If both inputs are int output is int . If any in float result in float	print(a//b)	0
%	Modulus	print(a%b)	10

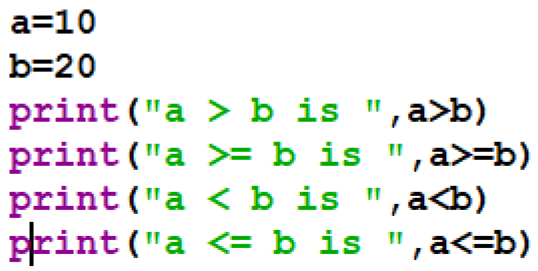
**	Exponential/ power operator	print(a**b)	a=2
			b=3
			a**b=8

Eg:
10/2==>5.0
10//2==>5
10.0/2===>5.0
10.0//2===>5.0

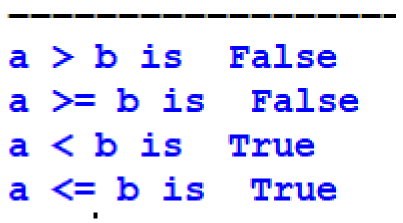
Note: + and * also used on strings
ex:
‘cbit’+’cse’ # cbitcse
‘cse’*3 # csecsecse

2. relational operators

- - compare the object values
- - >,<,<=,>=, != ,==
- - result into True or False



```
a=10
b=20
print("a > b is ",a>b)
print("a >= b is ",a>=b)
print("a < b is ",a<b)
print("a <= b is ",a<=b)
```



```
-------------------
a > b is  False
a >= b is  False
a < b is  True
a <= b is  True
```

- 10<20<30<40 # True
- 10<20>30<40 # False

True>False #True
True<False #False

- applicable to string objects also.
- 10==10.0 # True
- 10==‘10’ # False since both are incompatible type

3. Logical operator

- and, or and not
- Boolean type
  - and -> if both arguments are True then result is True
  - or -> if at least one argument is True then result is True
  - not -> False returns True and vice versa

- Non- boolean type
  - 0- False
  - non zero- True
  - empty string is always treated as False

- x and y: if any one of the argument is false that argument will return.

  ex: 0 and 20 # 0

- if x and y are true second argument will return

  ex: 10 and 20 # 20

- x or y: if x is True then result is x otherwise result is y

  ex: 0 or 10 # 10

  10 or 20 # 10

- not x:

  False- True

  True- False

Ex:

‘cbit’ and ‘cse’ # cse

‘cbit’ or ‘cse’ # cbit

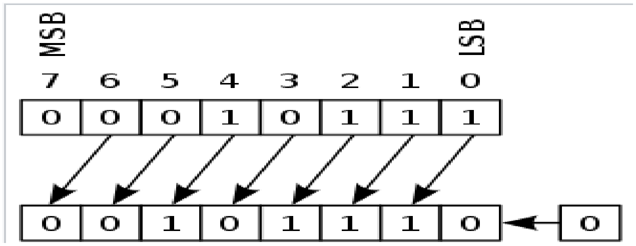
“ “ and ‘cse’ #” “

Note: Logical operators applied on bits of the objects, returns result in bool value

4. bitwise operators

- we can apply bitwise operators on int and bool type only. Performs operations on individual bits and result is also bits.
- &,|,^,~,<<,>>
- &: if both bits are 1 then result is 1 else 0
- | : at least one bit is 1 result is 1 else 0
- ^: if both the bits are different then result is 1 otherwise result is 0
- ~: 0->1, 1->0
- <<(left shift operator): shift bit left and zero is added at the least significant bit position.

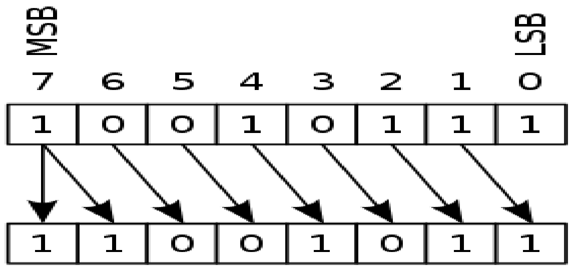
  ex: print(2<<2)# 8 shift two bits to left when any number to be multiply by 2 then left shift operator to be used



- >>(right shift operator): shift bit right and zero is added at most significant bit position

  ex: print(4>>1)# 2

  number is divide with the 2 when we use right shift operator.



5. Assignment operator

- assigns value to a variable
- We can combine assignment operator with some other operator to form compound assignment operator.
- The following is the list of all possible compound assignment operators in Python

+=	a=a+b
-=	a=a-b
*=	a=a*b
/=	a=a/b
%=	a=a%b
//=	a=a//b
**=	a=a**b
&=	a=a&b
|=	a=a|b
^=	a=a^b
>>=	a=a>>b
<<=	a=a<<b

6. special operators

1. Ternary operator
2. Identity operator
3. Membership operator

1.Ternary operator: Replacement of if else statement

Syntax:

a= x if condition else y

If condition is true x will be assigned to a else y will be assigned.

```
a,b=20,30
min=a if a<b else b
print(min)#20
```

- Program to find maximum of three numbers

```
a=int(input("Enter First Number:"))
b=int(input("Enter Second Number:"))
c=int(input("Enter Third Number:"))
max=a if a>b and a>c else b if b>c else c
print("Maximum Value:",max)
```

2. identity operator: uses is, is not key words

- used to compare address of two operands or objects

compare a and b objects

```
a=10
b=20
print( a is b) # False
print(a is not b)# True
```

```
x=100
y=100
print (x is y)# True
print(x is not y) # False
```

2. identity operator: uses is, is not key words

- used to compare address of two operands or objects

compart a and b objects

```
a=10
b=20
print( a is b) # False
print(a is not b)# True
```

```
x=100
y=100
print (x is y)# True
print(x is not y) # False
```

3. member ship operator(in and not in)

- To check whether the given object present in the given collection or not
- Collection can be (string, list, tuple, set or dict)
- Uses in and not in as keywords

```
x='cbit'
print('c' in x) #True
print('d' in x)#False
print('d' not in x)#True

names=["sunny","bunny","munny","chunny","pinny"]
print("sunny" in names)# True
print("chinny" in names)# False
```

Operator Precedence

If multiple operators present in expression which operator to be evaluated first is determined by operator precedence. Highest precedence should be evaluated first.

Operator precedence chart
()
**
~, - unary operators
*/,% ,//
+, -
&
^ XOR
|
Relational operator
assignment operator
Is, is not

references

https://docs.python.org/2.0/tut/node6.html

http://thomas-cokelaer.info/tutorials/python/lists.html

https://docs.python.org/3/tutorial/datastructures.html

https://www.programiz.com/python-programming/tuple#index

https://www.programiz.com/python-programming#tutorial

http://interactivepython.org/courselib/static/thinkcspy/index.html important

Questions

What are the differences between tuple and list?

List out the methods in list object?

What are immutable objects in python

What are the fundamental data types in python.

What are the features of object oriented programming

What are the features and applications of python?

Differentiate between set and tuple

Differential between list and set data type

#Task: Practice all the methods of dict object and how to add and change dict values.

- List out built in methods in dict object and explain each with example.
- Create a dict object for the 5 students details such as name,branch,section,phoneno. Consider rollno as a key.