Introduction to Python

Lesson plan

Subject/Course	**Python Programming**
Lesson Title	**Introduction to python**

Lesson Objectives
What is history of Python , Version History of Python
Why you should learn Python?, Features of Python , Amazing facts
Where Python can be used? , Learning Path

History of Python

- Python was conceived in the late 1980‘s by Guido van Rossum at centrum Wiskunde and Informatica in the Netherlands as a successor of the ABC language.
- He has been invariably worked as the lead developer, until 12 July 2018
- In jan 2019, active Python core developers elected a five member council to lead the project

Version History

- Python 1.0 1994
- Python 2.0 2000
- Latest version
- Python 3.11 2022
- Python 3.12 2023
- Python 3.13 2024
- Name – Python
- Python developers aim for it to be fun to use, the name ‘python’ is a tribute to the British comedy group Monty Python

Why you should learn Python ?

- Huge community Support
- Future is with AI
- Easy to learn and implement
- General purpose programming language

Features of Python

- Higly Extensible
- Simple and straight forward syntax
- Multi Paradigm programming language
- Emphasis on code readability
- Dynamic typing automatic memory management
- Dynamic Binding
- Precise Coding
- For time critical Operations, python can use modules written in C language
- Introduction base block of code
- Large library
- Platform independent

Amazing facts

- Ranked #1(TIOBE)
- Highest rise in rating 2007, 2010, 2018, 2020
- It reduces app development time by 1/6
- Large organizations that uses python are Google, Netflix, Dropbox, Youtube, Instagram, CERN, Microsoft, NASA, Amazon, facebook, Mozilla, etc.

Where Python can be used?

- Developing websites
- Task automation
- Data Analysis
- Data visualization
- AI, ML,IOT
- Developing desktop applications
- Python can also be used in developing mobile applications, client side AJAX based applications.

Learning Path

- Learn by doing Projects
- Absolute Beginner →core python →oops →Develop console app using some DB.
  →Develop GUI app
  →Develop Web application
  →Develop ML based app
  →Learn Devops

VARIABLES AND TYPES

Variables:-

- Variables are used to hold data during execution of the program.
- In C, C++, you need to declare variables. Only after declaration you can use them.

```
int a;
float b;
```

- C, C++ are statically typed languages

- Python is dynamically typed language.
- In python you don't declare variables. If there is a need of a variable you think of a name and start using it as a variable

C/C++	Python
• int a; declaration	a = 5 Automatic declaration
• a++;	----------
• print("%d" , a);	a = 5
• a = 7;	a = a + 5

Variable name:-

- Variable name is any combination of alphabet, digit and underscore.
- Variable name cannot start with digit.
- Variable names are case sensitive.
- Keywords cannot be used as variable names.

Deleting variable:-

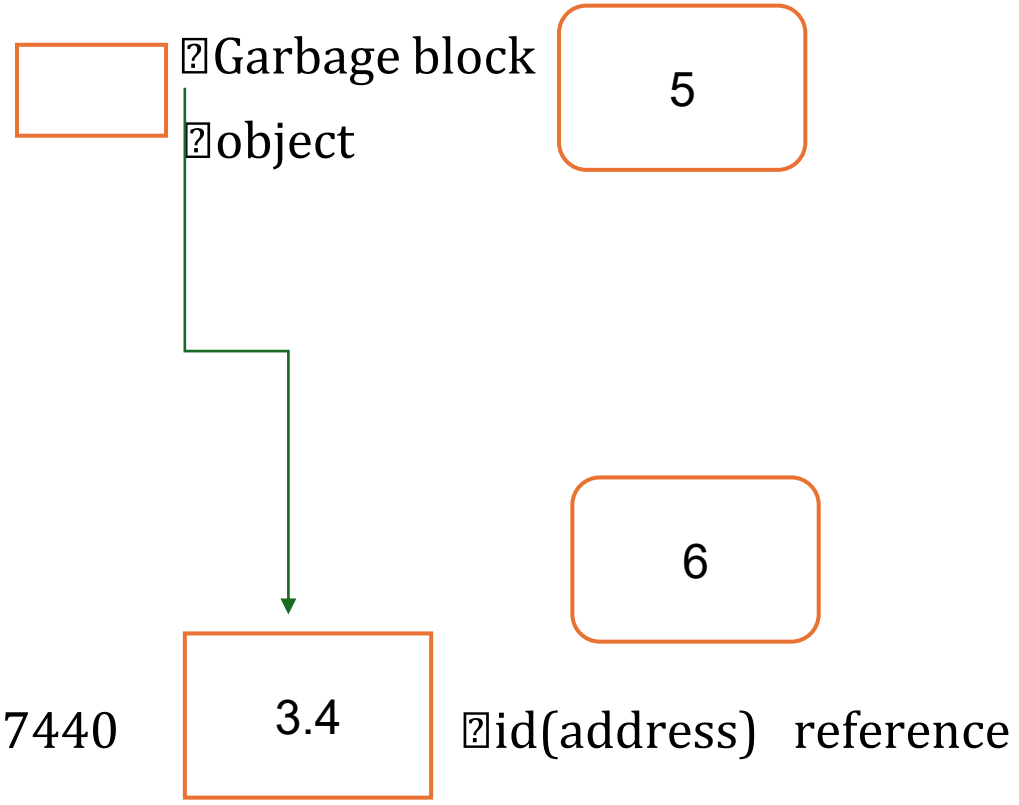
- x = 5
- --------
- del x

Dynamic type:-

- Not only the value of a variable may change during program execution but the type as well.
- x = 5 # type of x is int
- x = 5.7 # type of x is float
- x = True # type of x is bool
- x = “country” # type of x is str

- a = 5
- a = 6
- a = 3.4

a



- int
- float
- bool
- str
-

data type

Classes

Type()

- type() is a predefined function which returns the data type of a specified variable
-
```
x = 5
type(x)
x = 5.7
type(x)
```

Data type is always a class in python

Data types:-

Numbers

int 5

float 3.7

complex 3+5j

Boolean

bool True

False

String

str “college” | “””college”””

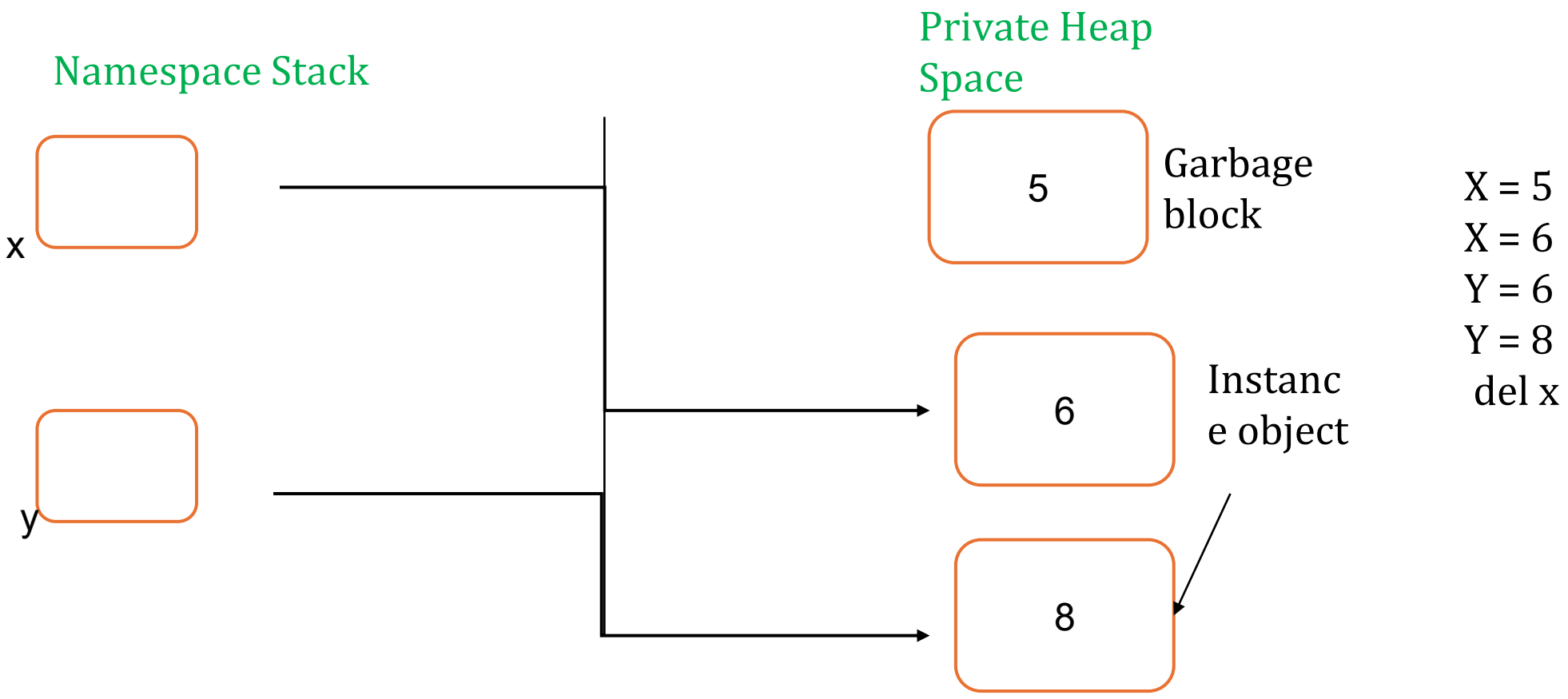
‘college’ | ‘’’college’’’

Double in not there in python

Char in not there in python

Memory management

Memory management:-



Garbage Collection

- It is a program, invoked by python itself , whenever required, whose job is to release memory of garbage blocks

  (object which is not referenced by any name)
- Automatic memory management

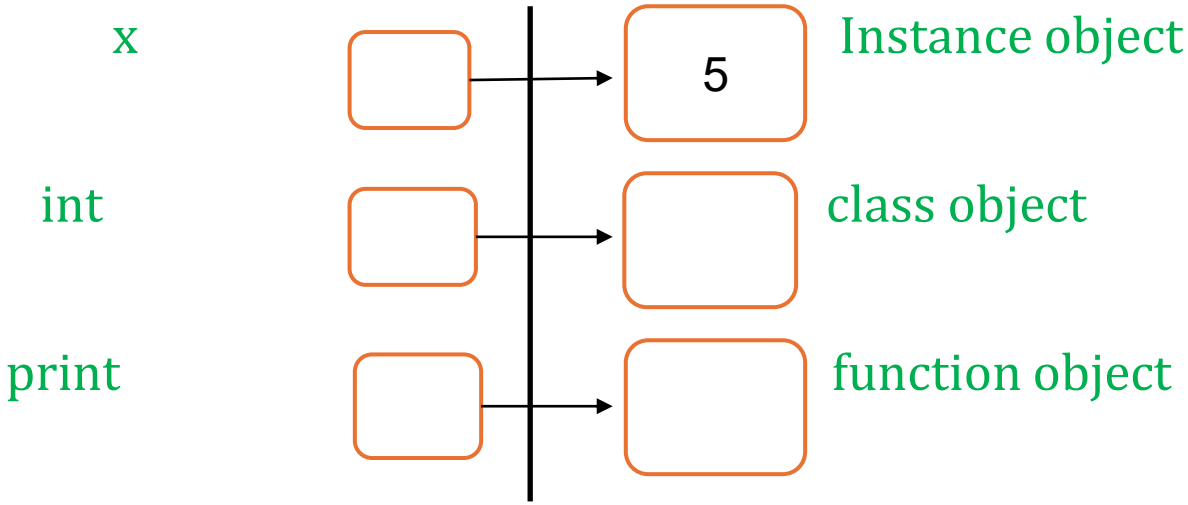
What is an object?

- Object is real world entity
- Object is an instance of a class
- Object is a proper noun
- Class is a common noun
- Doctor common noun class
- Dr . dixit proper noun object
- Dr . Sharma- proper noun object

Whose name?

- Name in namespace is a variable which is used to contain id(reference) of

1. Instance object
2. Function object
3. Class object



In python everything is an object.

Print, keywords and import

Print()

- Print simple text `print("Welcome")`
- Print variable value `print(x)`
- Print expression `print(x+5*3)`
- Print multiple value `print(x , y)`

Sep

```
a = 5
b = 6
c = 7
print(a , b, c)
```

- print(a, b, c, sep = '_') 5_6_7
- print(a, b, c, sep = '#') 5#6#7

End

a = 5

- b = 6
- print(a , b, c, end = ‘,’)
- Print(“Hello”)

Special characters

Escape Sequences

‘\n’ new line

‘\t’ tab space

‘\b’ backspace

‘\r’ Carriage Return

‘\\’ print \

‘\”’ print “

“\’” print ‘

Format Specifiers

- x = 5

  type(x) → int

  value of x is 5

  print(“value of x is” , x)

  value of x is 5

  print(“value of x is %d” %(x))

  value of x is 5

Module, Package and Library

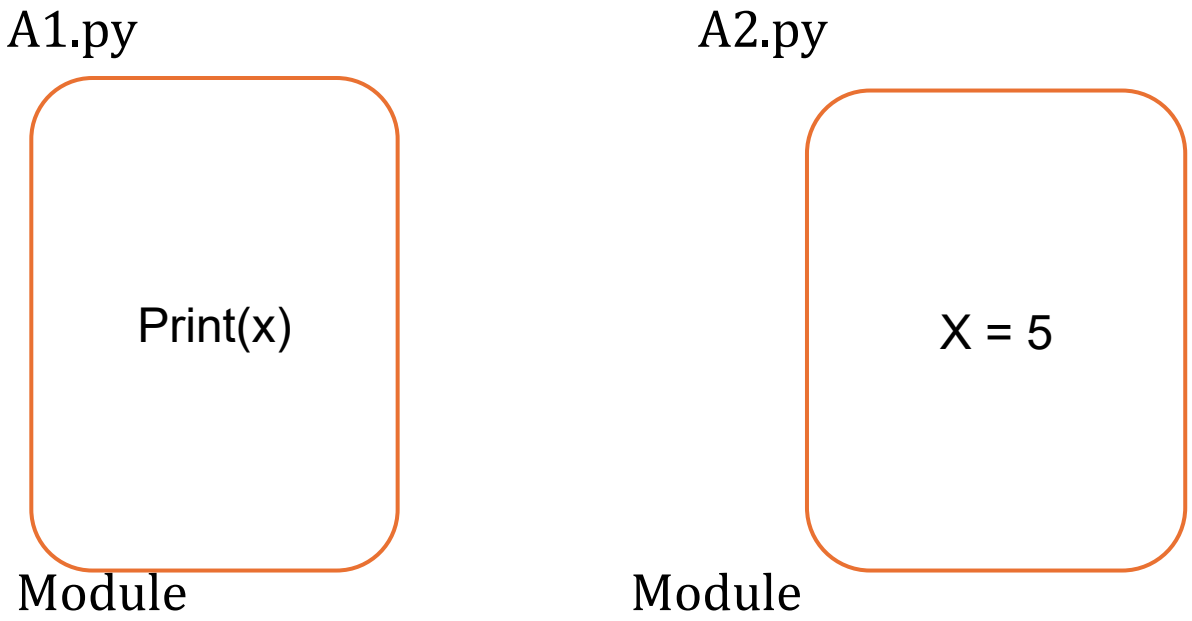
Module : - It is a python file. It can contain instance objects, function objects and class objects. In simple words, it contains variables, functions and classes

Package:- Package is a collection of modules and sub-packages

Library:- Library is a collection of packages.

import

- Import A2 moduleName.moduleElement
- ModuleElement can be
  - variable(instance object)
  - function
  - class



From A2 import x

Print(x)

- A1.py

```
From A2 import x as X
X = 10
Print(X)  5
Print(x)  10
```

A2.py

```
X = 5
```

Keywords

- Predefined words
- Reserved words

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	alse	is	while
async	except	lambda	with
await	finally	non local	yield
break	for	not	

Help()

- >>>help()

Type conversion and number system taking input from keyboard

Type conversion

a = 5 int

b = “5” str

A + b int + str Error

A + int(b) int + int No error

10 int

Str(a) + b str + str No error

“55” str

Type conversion functions

int()

float()

complex()

bool()

str()

Number System

- Binary Number System 0,1
- Octal Number System 0,1,2,3,4,5,6,7
- Decimal Number System 0,1,2,3,4,5,6,7,8,9
- Hexadecimal Number System0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

counting

- Dec oct Hex Bin

Dec	oct	Hex	Bin
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
-	-	-	-
-	-	-	-

Place Value

Decimal 2 1 4 8 = 2000 + 100 + 40 + 8

10^3 10^2 10^1 10^0

Octal 3145

Hexadecimal A10F

Binary 1001110

Conversion of Number System

Dec Bin

25 11001

76

100

96

Number System Conversion functions

x = 25

bin(x) ’0b11001’

oct(x) ’0o31’

hex(x) ’0x19’

---

x = 0b11001

x = 0o31

x = 0x19

Unicode

The Unicode is character encoding, and its goal is to replace the existing character sets with its standard UTF.

UTF Unicode Transformation format

UTF –8 Is the most commonly used character encoding.

It is also backward compatiable with ASCII

Character to Unicode	Unicode to character
x = ‘A’	x = 65
Ord(x) 65	chr(x) ’A’

Taking Input from user

input()

input() can take at most one argument of str type

input() always return str type value

How to input a string?

```
s = input()
```

How to input an int value?

```
x = int ( input())
```

How to input a float value?

```
x = float ( input())
```

How to input a complex value?

```
x = complex (input())
```

Input can take one argument

- To provide direction for input you can pass a string argument in input method.

```
name = input("Enter your name")
age = int(input("Enter your age"))
```

Can you take multiple value?

- You can use multiple input() function to input multiple values

  a = int (input (“Enter first number”))

  b = int (input(“Enter second number”))

You can input multiple values using single input() function but it is little bit tricky and requires knowledge of str class, which we will see in later lessons.

OPERATORS

Bitwise OR	a|b
Exponentiation	a**b
Identity	a is b
Identity	a in not b
Indexed Assignment	obj[i] = a
Indexed Deletion	del obj[i]
Indexing	obj[i]
Left shift	a<<b
Right shift	a>>b
Logical AND	a and b
Logical OR	a or b
Modulo	a%b
Multiplication or Repetition	a*b
Negation	-a

Negation (logical)	not a
Positive	+a
Slice Assignment	s[i:j] = values
Slice Deletion	del s[I : j]
Slicing	s[I : j]
String formatting	s%s1
Subtraction	a – b
Ordering(less than)	a < b
Ordering(greater than)	a > b
Ordering(less than or equal to)	a <= b
Ordering(greater than or equal to)	a >= b
Equality	a==b
Not equal	a !=b

Operators:-

- Arithmetic Operators **, /, //, *, %, +, -
- Relational Operators <, >, <=, >=, ==, !=
- Logical Operators not, and, or
- Bitwise Operators &, |, ^, ~, >>, <<
- Assignment Operators =, +=, -=, //=, %=, **=, *=, &=, |=
- Identity Operators is, is not
- Membership Operators in, not in

++, -- are not the operators in python

Arithmetic Operators:-

**, //, /, *, +, -, %

2**3 → 2*2*2 = 8

/ always return float result

// always return floor value , int type or float type depending on operands.

+, * can be used with str type values also

+ is addition operator when operands are numbers (int, float, complex, bool)

+ is concatenation operator when operands are str

+ operation is invalid when applied between a number and a string.

* is used to multiply two numbers

* is repetition operator when applied between a str and an int

Relational Operators

- <, >, <=, >= inequality operators
- ==, != equality operators never gives error

- Relational operators always give result in True or False.
- When truth value is converted to int, it becomes 1 for True and 0 for False
- Relational operators can also be used to compare two strings
- Only == and != operators can be used between two complex type values.
- == and != never yield error

Logical operators

not

and

or

- Logical operators must be written in lowercase only

not True ⍰False

not False ⍰True

True and True ⍰ True

True and False ⍰False

False and X ⍰False

False or False ⍰ False

False or True ⍰ True

True or X ⍰ True

Every non zero value True

Zero False

Non empty string True

Empty string False

When operands are non-bool then result will also be non-bool

Bitwise Operators

&, |, ^, !, >>, <<

0&0 ⍰0

0&1 ⍰0

1&0 ⍰0

1&1 ⍰1

0|0 ⍰0

0|1 ⍰1

1|0 ⍰1

1|1 ⍰1

x = 25 & 37

25 = 011001

37 = 100101

1 = 000001

x = 44 | 71

44 =

71 =

0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

x = 56 ^ 29

56 =

29 =

~ 0 → 1

~1 → 0

x = ~5

5 = 00000101

~5 = 11111010 = -k = -6

2's → 00000110 = 6

k = b1

-k = b2

k = 1101

1's = 0010

+1

0011 2's → -k

35 >> 2

35 = 00100011

 1000 = 8

12 << 3

12 = 1100000 = 96

Assignment operator

=, +=, -+, *=, /=, //=, &=, |=, ^=, **=, >>=, <<= , %=

X = 4

Must be avariable

3 =4 Error

x = x + 3

x++ Error

x = x+1

x+=1

How to assign values to multiple variables in a single line?

x = 4, y =3, z = 2 Error

x, y, y = 4, 3, 2 correct

x, y, z = 2,3 Error

Identity Operator

is

is not

It checks whether the two references refering to the same object or no.

It results in True or False.

a = 3.5 a □ x = 5 x □

b = 3.5 y = 5 y □

b □

3.5

5

3.5

a = 5 a

b =5 b reference count (2)

5

Membership operators

in

Not in

These operators are applicable only on containers (iterable)

They result True or False

Container is a type which can contain multiple values and also iterable

x 

15, 20, 37, 4, 61

15 in x True

25 in x False

int, float, complex, bool are not iterable

str, range, list, tuple, set, dict are iterable

Decision control

Control statement

1. Decision control if
   - if else
   - if elif else
   - single line if else
2. Match case
3. Iterative control

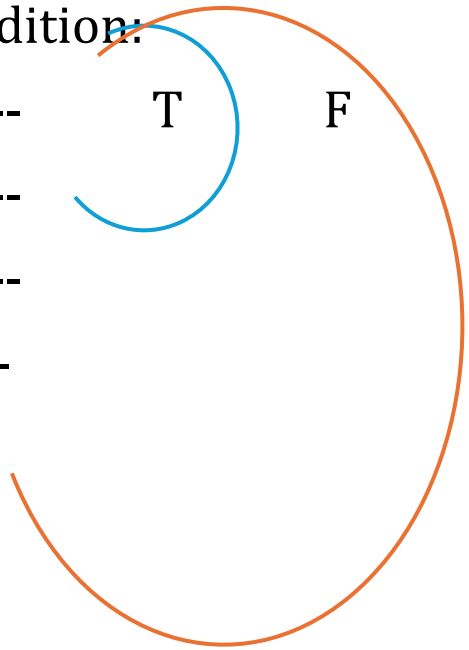
---

if

keyword

if condition:

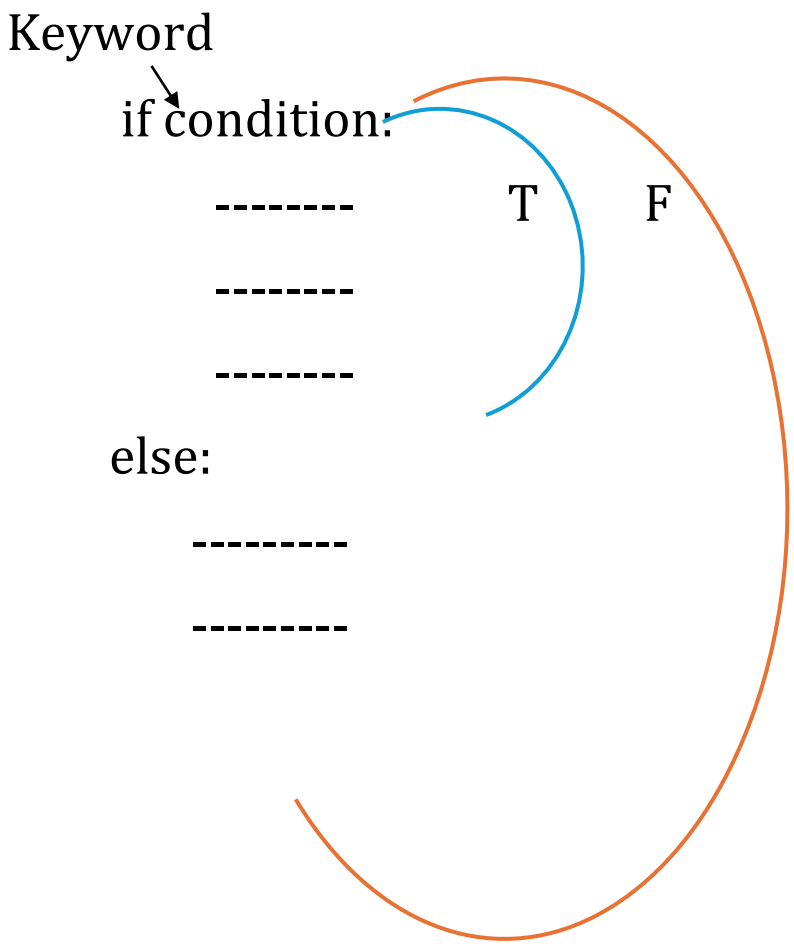
------ T F



- Write a program to check whether a given number is positive or non positive

```
x = int ( input("Enter a number"))
if  x>0:
    print("positive")
if x<=0:
    print("non positive")
```

if else



- Write a program to check whether a given number is positive or non positive

```
x = int(input("Enter a number"))
if  x>0:
    print("Positive")
else:
    print("Non positive")
```

if elif else

if condition:

-------- T

Elif condition:

--------- T

Elif condition:

---------- T

Else:

----

elif = else if

Write a program to print grade obtained in a test. Take marks obtained from user and Display the grade.

90 < marks <= 100	A
80 < marks <= 90	B
70 < marks <= 80	C
60 < marks <= 70	D
50 < marks <= 60	E
below 50	F

Single line if else

code1 if condition else code2

Single line if else is an expression but if else is not an expression

```
x = if ---:                    x = code1 if condition else code2
        ----                       x = code1
        ----
Else:
     ------
     ------
```

Write a python program to check whether a given number is positive or non positive.

```
x = int (input("Enter a number"))
print("positive") if x>0 else print("non positive")
```

Nested if else

```
if condition:
    ----------
    ----------
      if condition:
          ----------
          ----------
       else:
          -------
          -------
else:
   -------
   -------
```

Match, while loop, transfer control statement, and for loop

```
match subject_expression:
        case constant:
                ----------
        case constant:
                ----------
        case constant:
                ----------
        case constant:
                ----------
```

break keyword cannot be used in match block

Example:-

```
x = int(input("Enter a number"))
match x:
case 1:
    print("One")
case 2:
    print("Two")
case 3:
    print("Three")
case 4:
    print("Four")
```

case constant can be of any type.

```
x = eval(input("Enter a number"))
match x:
    case 1.1:
        print("One")
    case "two":
        print("Two")
    case True:
        print("Three")
```

Default matching

```
x = int( input(“Enter a number”))
match x:
    case 1:
        print(“One”)
    case 2:
        print(“Two”)
    case 3:
        print(“Three”)
    case_:
       print(“default”)
```

match

- match and case are soft keywords
- They are not reserved words in other grammatical contexts.
- They are recognized as keywords when part of a match statement or case block only, and are allowed to be used in all other contexts as variables or argument names.

duplicate case

duplicate case is not an error but first match will only work.

Conditional casing

```
match expression:
    case data if condition:
        ---------------
        ---------------
    case data if condition:
        ------------------
        ------------------
```

While loop

Objectives:-

1. Iterative control
2. while
3. practice problem

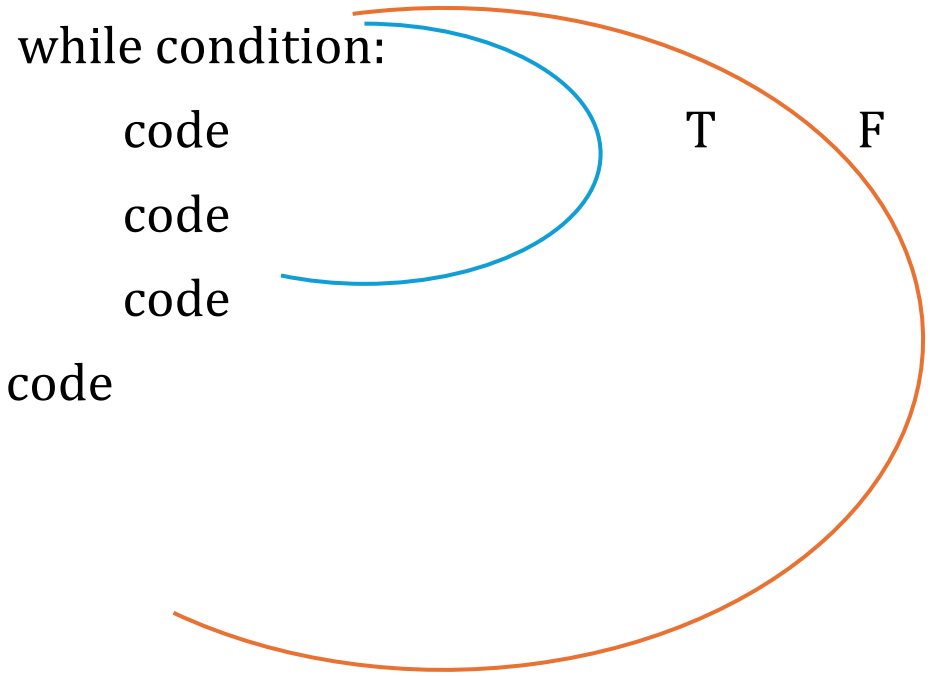
Iterative control(Loop)

while

for

X do while

while



- Write a program to print Hello Nature five times.

```
i=1  initialization
while i <=5:  condition
        print("Hello Nature")
        i + = 1  flow
```

- Write a program to print first 10 natural numbers.

1 Read/understand

2. Test case

3. Dry run

- Write a program to print first 10 natural numbers in reverse order.

- Write a program to print first n even natural numbers.

Transfer control statement

Objectives:-

1. Break
2. continue
3. else with loop
4. pass

break

- break is a keyword
- break is used to transfer control outside the loop body.
- break can only be used in the body of loop.
- break terminates the execution of loop.

- Write a program to ask user to enter an even number at most 3 times.

  It user failed to enter an even number in all the three chances then he has lost the

  game. It user enter an even number, then no more chances will be given and annonnced him a winner.

continue

- Continue is a keyword
- It can only be used in the body of loop
- Continue transfer the control immediately to the next iteration.

Example

- i = 1

```
while i <= 10:
        x = int (input(“Enter a number”))
        if x % 2 == 0:
                continue
        print(i ,”x = “, x)
        I +=1
```

else with while loop

```
while condition:
    code
    code
else:
    code
```

else block executes when while loop terminates normally.

else block won’t be execute when loop terminater due to break statement.

pass

pass keyword is used to create empty block.

```
if condition:
    pass
```

```
while condition:
    pass
```

```
while(--)
{
}
```

```
if(---)
{
 }
```

For loop

Objectives:-

1. for loop
2. practice program
3. for vs while
4. break in for
5. for else

for loop

for loop only works on iterables.

```
for  variable in Iterable:
    code
    code
```

Example

```
for x in “Hello”:
    print(x)
```

output

H

e

l

l

o

Write a program to count ‘a’ in a given string, using for loop.

```
s = input(“Enter a string”)
        count = 0
for e in s:
     if e ==‘a’:
         count+= 1
print(“Total occurrence of ‘a’ is” , count)
```

while vs for

While loop iterates a block of code till the specified condition is true.

for loop iterates a block of code number of times same as number of elements in the iterable.

It iterates for elements of iterable from first element to the last element

Iterable object

- Group of values
- Ability to access elements of the object (from first element to the last element) is applicable on certain types.

break in for

break keyword can be used only in the body of loop (while or for) break transfers the control outside the loop body and loop terminates

for else

```
for variable in iterable:
        code
        code
        if condition:
            break
         code
else:
         code
         code
```

print all the character of a string, but stop printing it ‘r’ appeared in the sequence It all the character successfully prmted the print message “All the characters are processed”

```
x  = input(“Enter a string”)
for i in x:
    if i ==‘r’:
        break
    print(i , end = “”)
else:
    print(“All the characters are processed”)
```

Range

What are iterables?

- An iterable object is something that contains a countable number of values.
- You can traverse through all the values from beginning to the end.
- Technically , in Python , an iterator is an object , which implements the iterator protocol , which consist of methos __iter__() and __next__()

Various iterables

- range
- list
- tuple
- str
- bytes
- bytearray
- set
- frozenset
- dict

range

- range is a class
- range is immutable sequence
- range can contain only int type values
- range contains sequence of integers with common difference (Arithmetic progression)
- range elements are indexed

How to create range object?

- r = range (beg, end, step)
- Example:-

range (1 , 10 , 1)

range (2 , 8 , 2) 2 4 6

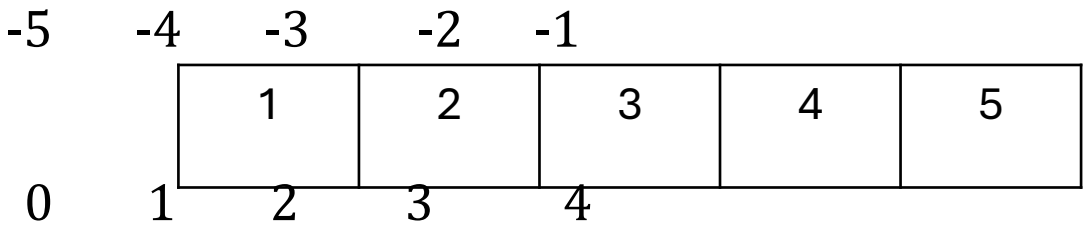
range (10 , 3 , -2) 10 8 6 4

Other ways to create range object

- r = range (beg , end , syep)
- 1 range(end) beg = 0   step = 1
- 2 range(beg , end)   step = 1

indexing

- r = range(1 , 6 , 1)
-



r[0] = 1

r[1] = 2

r[4] = 5

positive and negative indexing r[-1] = 5 , r[-2] = 4 , r[-4] = 2

Accessing range elements

```
for e in r:
    code
     # e is an element of range
```

- Write a program to print first n natural numbers.

[use range and for]

n = 6

1 2 3 4 5 6

range(1 , n+1, 1)

- Write a program to print squares of first n natural numbers.

[use range and for]

- Write a program to print first n even natural numbers in reverse order. [use range and for]

- Write a program to calculate sum of first n multiples of x.

[use range and for]

List

list

list is a class

list is an iterable sequence

list is mutable

list is growable

list can store heterogeneous data

list elements are indexed

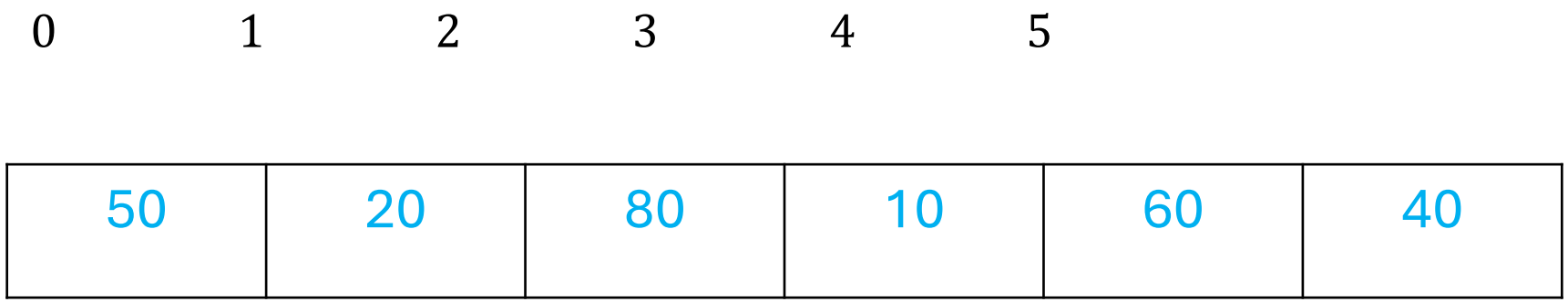
How to create list object?

```
l1 = [10 , 20 , 30]              # Square brackets are used
                                     to denote a list

l2 = []            #empty list object
l3 = [50 , 3.5 , “abc”]   # Hetrogeneous elements
```

How to access list elements?

```
l1 = [50 , 20 , 80 , 10 , 60 , 40]
print(l1)    # [ 50 , 20 , 80 , 10 , 60 , 40]
print(l1[0])   #50
print(l1[1] , l1[2])   #20 80
```



Concept of negative indexing

l1 = [50 , 20 , 80 , 10 , 60 , 40]

-6	-5	-4	-3	-2	-1
0	1	2	3	4	5


```
print(l1[-1])   # 40
print(l1[-2])   # 60
```

Accessing list elements via for loop

```
l1 = [50 , 20 , 80 , 10 , 60 , 40]
 for  x  in  l1:
        print(x , end = ‘ ‘)
```

```
i = 0
 while i<=5:
        print(l1[i],end = ‘ ‘)
        i+=1
```

How to delete an element from the list?

L1 = [50 , 20 , 80 , 10 , 60 , 40]

del l1[2]

How to edit an element of the list?

l1 = [50 , 20 , 80 , 10 , 60 , 40]

l1[2] = 45

How to add more elements in the list?

l1 = [50 , 20 , 80 , 10 , 60 , 40]

l1[6] = 70 # index error

list object[invalid Index] is an error

l1[5] = 70

l1[4] = 90

Not adding more values in the list but only updating values at index 5 and 4

There are two standard ways to add elements in the list:

1 append()

2 insert()

attributes of list class

Class is a group of variables and functions

There variables and function are called attributes.

- listObject.append(value)
- listObject.insert(index , value)

append method adds value at the end of the list , that is new element will become the last element of the list

l1 = [50 , 20 , 80 , 10 , 60 , 40]

l1.append(70)

l1 = [50 , 20 , 80 , 10 , 60 , 40]

l1.insert(-5 , 70)

l1.insert(index , value)

If index > last index then value will store at last index+1 only(append)

Packing and Unpacking

```
l1 = [20 , 50 , 30]
a , b , c = l1              # unpacking
```

number of variables in the left hand side must be same as the number of elements in the list.

```
a = 5
b = 6
c = 10
l2 = [a , b , c]            # packing
```

Built in methods

len() → return length of specified iterable

min() → return min value element

max() → returns max value element

sum() → returns sum of elements.

sorted() → returns a sorted list of elements

```
l = [24, 13 , 5]
l1 = sorted(l)
      [5 , 13 , 24]
```

list() method

```
l1 = list()        # l1 = []
l1 = list(10)      # l1 = [10]     error
l1 = list(10 , 20 , 30)
l1 = list("MySirG")
l1 = list(range(5))
l1 = list([10 , 20])
```

list() method can take at most one argument.

one argument iterable

Comparison Operator on list

l1 = [1 , 2 , 3]

l2 = [2 , 3 , 1]

l3 = [1 , 2 , 3 , 4 , 5]

l4 = [1 , 2 , 3]

l1 == l2 False

l1 == l3 False

l1 == l4 True

l1 > l2 False

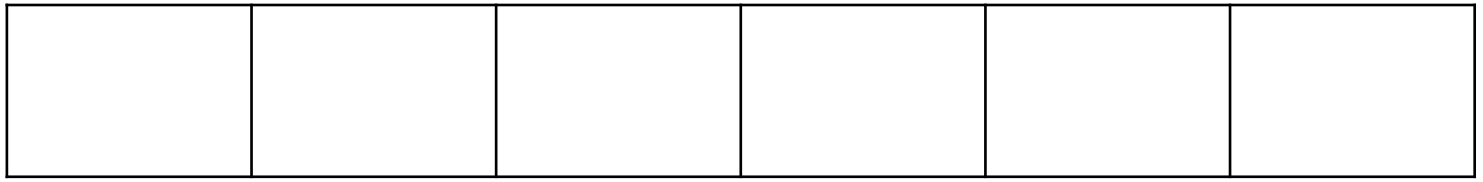
Concatenation Operator

l1 = [1 , 5 , 9]

l2 = [2 , 3 , 1]

l3 = l1 + l2

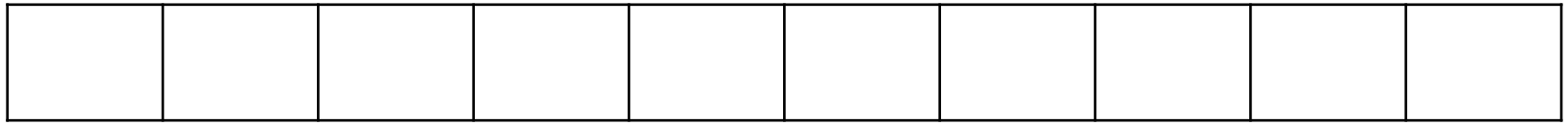
l1 + = l2 ⍰ l1 = l1 + l2



Repetition Operator

l1 = [2 , 5]

l1 * 5



Slicing Operator

listObject[beg : end : step]

l1 = [20 , 40 , 10 , 30 , 60 , 50]

By default step = 1

By default beg ,and = extreme end

l1[2 :6 :2]

l1[4 : 0 :-1]

Str , sets , tuples and dictionaries

Str

str is a class

str is immutable

str is iterable

str is hashable

str is a sequence

How to create str object?

s1 = “college”

s2 = ‘college’

s3 = “””Hello”””

s4 = ‘’’Hello’’’

s5 = str()

s6 = str(125)

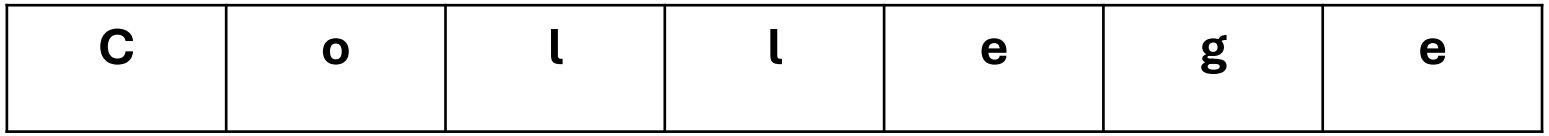
s7 = str(3.45)

indexing

s1 = “College”

s1[0] C

s1[2] l



Accessing str elements

1. s1[index]
2. print(s1)
3. for loop
4. slicing operator strobject[: :]

Built in methods

len()

min()

max()

sum()

sorted()

Concatenation and repetition operator

s1 + s2 s1 = “ABC” s1 + s2

s2 = “DE” “ABCDE”

S1 * 3 s2 * 3 “DEDEDE”

Comparison Operator

s1 > s2 → true if s1 comes after s2 in dictionary order

Str object methods

index()
count()
startswith()
endswith()
split()
join()
format()
isdigit()
islower()
isupper()
lower()
upper()
replace()

split

s1 = “ education services”

l1 = s1.split(‘ ‘)

split function returns a list of splitted strings

l1 = [“education” , “services”]

join()

l1 = [“education” , “services”]

space = “ “

s2 = space.join(l1)

join will return a string which is formed by joining elements of l1 with the help of spaces.

format

String.format(var1 , var2 , ….)

print(“ { } , how are you?” .format(“Mahavir”))

print(“ {] , { } , { } “ .format(one” , 25 , 3.5))

print(“ {2} , {0} , {1}” .format(10 , 20 , 30))

Split and join

s1 = “ education services”

l1 = s1.split(“ “)

split function returns a list of splitted strings

l1 = [“college” , “education” , “services”]

Taking input list elements

```
s1 = “35 , 40 , 20 , 10 , 55 , 80 ,17 , 60”
l1 = s1.split(‘ ,’)
l1 = [“35” , “40” , “20” , “55” , “80”  , “17” , “60”]
mylist = [int(e)  for  e in l1]
       = [35 , 40 , 20 , 10 , 55 , 80 ,17 ,60]
```

join()

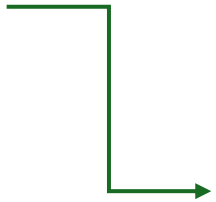
l1 = [“25” , “10” , “2025”] list of str

list of str str

“25102025”

“25/10/2025”

“25-10-2025



s1 = “-”.join(l1)

join returns a str type value

split → str →list of str base string

join → list of str → str base string

S1 = ‘Education Services’

base string = ‘ ‘ (space)

s1.split(‘ ‘) → [‘Education’ , ‘Services’]

l1 = [‘Education’ , ‘Services’]

base string = ‘ ‘ (space)

baseString.join(list of str)

‘ ‘.join(l1) →’Education Services’

set

set is ate a class

set is mutable

set is not hashable

set is iterable

set is not a sequence

set cannot have duplicate values

indexing is not applicable to set object

Slicing operator is not applicable

set does not guarantee to store values in the order of insertion

How to create set object?

s1 = { 1 , 5 , 8}

s2 = { 10 , 2 , 8 , 10 , 10 , 8}

s3 = { }

s4 = set()

s5 = set([1 , 5 , 3])

Elements of set object must be hashable

Accessing set elements

```
for  e  in setobject:
        code
```

built – in methods

len()

min()

max()

sum()

sorted()

Concatenation and Repetition Operator

set + set	not supported
set * int	not supported

Comparison operator

s1 > s2

s1 > s2

s1 >= s2

s1 <= s2

s1 == s2

s1 != s2

Two set objects are equal if their elements are same , doesn’t matter the order of elements.

set object methods

add() add specified item in a set

update() add iterable(s) elements

discard() remove item

remove()

intersection()

union()

clear()

is subset()

Is superset()

pop()

set comprehension

s1 = { expression for e in object }

s = input (“Enter a string”)

s1 = { e for e in s if e in “aeiou”}

```
for e in s:
    if e in “aeiou”:
        s1.add(e)
```

user input

```
s = input("Enter elements separated by comma")
s1 = { evale   for  e in s.split(' , ')}
```

forzenset

A set is a mutable object while frozenset provides an immutable implementation.

tuple

tuple is a class

tuple is iterable

tuple is immutable

tuple is a sequence

How to create tuple object?

t1 = (1 , 2 , 5 , 7)

t2 = ()

t3 = (10) not a tuple

t3 = (10 ,)

t4 = 10 , 20 , 30

indexing

t1 = (10 , 20 ,30)

Accessing tuple elements

t1 = (10 , 5 , 20 , 15)

1. t1[0]

   printf(t1[1]) 5

2. i = 0

   while(i<len(t1))

   print(t1[i])

   i+=1

3. For a in t1:

   print(a)

built – in methods

len()

min()

max()

sum()

sorted()

Concatenation and Repetition Operator

t1 = (10 , 20)

t2 = (11 , 22 , 33)

t1 + t2 (10 , 20 , 11 , 22 , 33)

t1 * 3

Comparison Operator

t1 = (10 , 20)

t2 = (11 , 33 , 55)

t1 > t2 false

t1 == t2 false

t2 > t1 True

tuple object methods

index()

count()

Slicing operator

t1[beg : end : step]

t1[: : -1] reverse

user input

```
t1 = tuple([int(e)   for  e   in input() .split(‘ ,’)])
t1 = tuple()
t2 = tuple([1 , 2 ,  3])
t3 = tuple( range( range(5))
```

dict

dict is a class

dict is mutable

dict is not hashable

dict is iterable

dict is not a sequence

dict cannot have duplicate keys (not data values)

indexing is not applicable to dict object

slicing operator is not applicable

dict elements are pair of key-value and data-value

How to create dict object?

Rollno	Student name
102	“Rahul”
105	“Payal”
106	“Prachi”

d1 = { 102: ‘Rahul’ , 105 : ‘Payal’ , 106 : ‘Prachi’}

d1 = {} #empty object

d1 = dict (a = 10 , b = 20 , c = 30)

key	data
‘a’	10
‘b’	20
‘c’	30

Accessing dict elements

d1 = {102 : ‘Rahul’ , 105 : ‘Payal’ 106 : ‘Prachi’}

1. print(d1)
2. print(d1[102] , d1[105] , d1[106])
3. for k in d1:
   print(k) only keys
4. For k in d1:
   print(k , d1[k])

How to edit dict element?

Editing dict element means you want to change data – value of the element and not the key – value

dict Object [key – vaule] = new data value

del d1[102]

How to add new element in the dict?

dictObject [new – key – value] = data – value

methods

items() → collection dict elements

keys() → collection of keys only of the elements

values() → collection of data-values only of the dict elements

All these methods are dict class attributes

built – in methods

len()

min()

max()

sum()

sorted()

Concatenation and Repetition Operator

dict + dict	not supported
dict * int	not supported

Comparison Operator

d1 > d2

d1 > = d2

d1 < d2

d1 <= d2

d1 == d2

d1 !=d2

Two dict object are equal iftheir items are equal.

Elements can be stored in any order.

dict object methods

pop(key)

popitem()

clear()

dict comprehension

d1 = { key- expression: data-expression for v in seq}

Thank you!