Date

Introduction to Python (week 2)

⇒ Print statement-

- Helps in printing the output to the screen
- Can be used in various ways

① Printing a group of character (String)

Syntax → print ("sarthak")

② Printing a variable

Syntax → print (Name of variable)

eg
```
a = 10
b = 20
c = a+b
print (c)
```

output
30

★ Variable can be a string as well.

③ Both together

Syntax → Print ("string", variable)

eg print ("Myname is", n) n = "sarthak")

output
Myname is sarthak

Date.....

print (n, " is my name")

output
Sarthak is my name

* many variable can be printed using comma b/w them

⇒ Variables

- Variable are letters/Alphabet or a string of char which are not keywords and hold data.

eg n = 10
↑ ← constant
variable

- if a variable is ever overwritten the old value is lost

eg n = 10
print ("n is", n)
n = 5
print ("n now is", n)

output
n is 10
n now is 5

- A variable can be inc, dec by using following

Syntax — Variable_Name = Variable_Name + step
↑ inc by how much

⇒ ; command

- ; is used to execute no. of steps/command simultaneously

eg a=1; print(a); a=a+1; print(a)

Spiral

Date.....

⇒ Taking inputs in python

- Input from the user for any variable is done using the syntax → `variable_name = input( )`

eg:→
```
n = input ( )
b = input ("Enter b")
print(n)
print (b)
```

This program will ask for input n and b and print these values accordingly.

Q: WAP for discount calculation on a bill

Sol
```
c = input ("enter bill amount");
d = int (c)
bill_amount = d * 0.9
discount = d * 0.1
print (c); print ("discount=", discount); print ("bill", b
```

output

let c = 100,

100
10
90

⇒ Converting to int

- When we take input of any variable it takes it as a string, so we need to convert it to int to apply the mathematical formulas. So we do the following.

Spiral

Date

Syntax → `New-variable = int(old-variable)`

* Strings can use mathematical statements but Not the same as an integer. eg ABC x 2 = ABCABC
  10 x 2 = 20

⇒ if loop

Syntax →
```
if (condi^n):
    body.
```

eg
```
c = input("Enter your choise")
d = int(c)
if (d == 1):
    print("you entered 1")
if (d == 2):
    print("you entered 2")
if (d == 3):
    print("Enough").
```
(== for comp always)

~~for loops~~ ⇒ if-else loop

Syntax →
```
if (cond^n):
    body
else:
    body.
```

eg
```
num = int(raw_input("Enter no."))
if (num >= 0):
    print num, "is 0 or +ve"
else:
    print num, "is -ve"
```

Spiral

Date

→ if elif loops

Syntax → if (cond^n):
body
elif (cond^4):
body
else:
body.

eg
```
num = int("Enter no.")
if (num > 0):
    print "+ve"
elif (num == 0):
    print "0"
else:
    print "-ve"
```

⇒ for loop

Syntax :→
```
for variable-name in range (n):
    body
```
counter

eg :-
```
for i in range (5):
    print ("Hellow")
    print (i)
```

Output
Hellow 0
Hellow 1
Hellow 2
Hellow 3
Hellow 4

* counter starts from 0 and ends at (n-1) always

Q WAP to find sum of n numbers

Sol
```
n = int (input("enter last digit"));  sum = 0
for i in range (n+1):
    sum = sum + i
print (sum)
```

Spiral

Date.....

⇒ While loops

Syntax :→
```
while (cond^n) :
    body
    counter = counter + 1.
```

eg :-
```
n = 1
c = 1
while (c == 1)
    print (" n = ", n)
    c = input (" continues (0/1) ")
    c = int (c)
print (" n = ! ", n)
```

⇒ lists

- It is a flexible data structure. which is implemented as

Syntax → list_name = [] or list_name = ["items"]

eg shopping = ["Coffee", "Tea", "Bread"]
0 1 2 ← Indexing.

→ Traversing a List

Syntax →
```
for i in list_name :
    print (i).
```

eg
```
for i in shopping :
    print (i)
```

output
Coffee
Tea
Bread. Spiral

Date

→ Appending an item

• We can append items to list using following.

Syntax :→ list_name.append("item")

→ Inserting an item

• Inserting an item at given pos^n,

Syntax → list_name.insert(pos^n, "item")

eg shopping.insert(1, "oil")

→ Counting an item in list

• This method is used to count the no. of occurance of an item in a list.

Syntax → list_name.count("item")

→ length of list

• It counts the total no. of items in the list an gives us the length

Syntax :→ len(list_name)

eg len(shopping)

Spiral

Date

→ Sort operation

- helps sort the list in ascending order.

Syntax :- list_name.sort()

* for numeric value acc to the number, for char done on basis of ~~ascii code~~ Alphabets.

→ Reversing a list

- Helps reversing a given list

Syntax :→ list_name.reverse()

→ list slicing

- It cuts the part of a list which we need.

Syntax :- list_name[start:end+1]

eg to slice 4 things from list students,

students[0:5]

* Default value for start is the min value [0].
* Default end value is the ~~end~~ len(list.)

- list_name[:] will print whole list

* if values not specified default value is taken

Spiral

Date

→ Sort operation

- helps sort the list in ascending order

Syntax :- list_name.sort()

** for numeric value acc to the number, for char done on basic of ~~ascii code~~ Alphabets.

→ Reversing a list

- Helps reversing a given list

Syntax :→ list_name.reverse()

→ list slicing

- It acts the part of a list which we need.

Syntax :- list_name[start:end+1]

eg to slice 4 things from list students,

students[0:5]

* Default value for start is the min value [0].
* Default end value is the ~~index~~ len(list.)

list_name[:] will print whole list

if values not specified default value is taken

Spiral

Q WAP to print fizz when multiple of 3, buzz if multiple of 5 and fizzbuzz if multiple of both.

Sol for i in range (1, 51):

```
    if (i % 3 == 0 and i % 5 == 0)
        print (str(i) + " = fizz buzz")

    else:
        if (i % 5 == 0):
            print (str(i) + " = Fizz")
        else:
            if (i % 5 == 0):
                print (str(i) + " = Buzz")

            else:
                print (str(i))
```

Q What is crowd computing?

Sol^n Outsourcing your problem to the crowd to find a sol^n is known as crowd computing. This way we can solve complicated problems, which even super intelligent powers cant.

Q WAP to show Crowd computing 10% trimmed mean concept.

Q To explain this concept first data is collected from the crowd and stored in a list (75 values for estimation)

Spiral

Date

```
From Statistics import mean.
Estimates = [ 75 values ]
Estimate.sort()
trim_val = int(0.1 * len(Estimates))
Estimate = Estimates[trimval : ]   ← Removes small val
Estimates = Estimates[ : len(estimate)-trim_val ]
                                   ← Removes last values
print (mean (Estimates))
```

Ok) easier way.

```
from states import mean
from scipy import stats
Estimate = [ 75 values ]
Estimates.sort()
m = stats.trim_mean (Estimates, 0.10)   ← direct calc of trim. mean
print (m)
```

⇒ Plotting data on a curve

To plot we need to import a library as follows,

```
import matplotlib.pyplot as plt
```

Now to plot

```
plt.plot ([1,2,3])   ← y values
```

* x values are passed automatically if not passed

Passing x values,

```
plt.plot ([1,2,3,4], [1,4,9,16])
          x val      y val.
```

Scanned by CamScanner

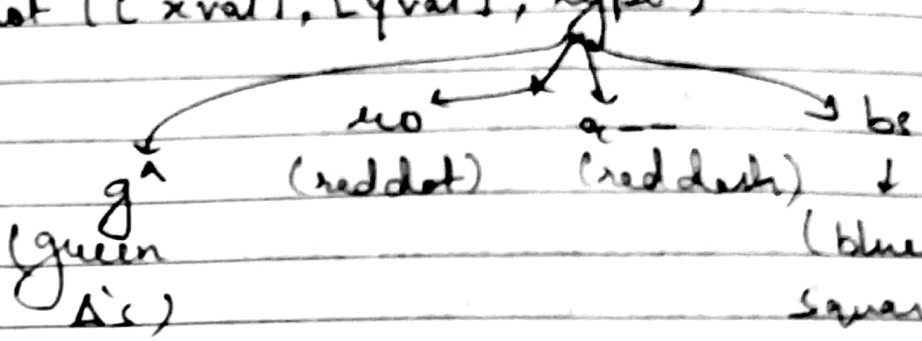
Date

· Labelling axis:

Syntax → libname.ylabel("Name")
libname.xlabel("Name").

· Adding Additional types

Syntax → libname.plot([xval], [yval], type)



Date

Q What are the advantages and disadvantages of python?

Sol^{n}. → Advantages

(i) Easy to use.
(ii) Expressive lang.
(iii) Interpreted lang.
(iv) It is complete
(v) Cross platform lang.
(vi) Free and open source

Disadvantages

(i) Not the fastest
(ii) Less libraries than C, Java, perl.
(iii) Not strong on type binding.
(iv) Not easily convertible

⇒ Data types

1.) Numbers

(i) integer's

(a) Plain Integer :— Uses only 32 bits (4 byte) generally to store a value

(b) long Integer :- No size limit. To stor plane int as long we add L to name. Special
eg:- 254, 36L

(c) Boolean :- True and false ie 0 and 1

eg
```
bool(0) = False
bool(1) = True
```

(ii) Floating point

(a) Fractional form :- 32.1, 36.4 etc

(b) exponential form :- 3.50E03, 0.5E-04

(iii) Complex Number

eg 0.5 + 1.2j
to print real part we use z.real
to print imaginary part we use z.imag

2.) Strings

(i) ASCII strings :- eg, 'boy', 'abc', etc

(ii) Unicode string :- eg, u'abc', u'βdγ', etc

3.) Lists and Tuples

(i) Lists :- It is a collection of data separated by comma that can be manipulated or changed

eg a = ['a', 'b', 1, 2]

Spiral

Date.....

(ii) Tuples :- It is same as lists but cannot be manipulated

eg z = ('a', 'b', 'c', 'd')
q = (1, 2, 3, 4)

4.) Dictionaries

- Unordered set of comma separated 'key: value' pairs within { }.

- In Dictionary no two keys can be the same ever.

eg vowels = {'a': 1, 'e': 2, 'i': 3, 'o': 4, 'u': 5}

key value

vowels ['a']
→ 1
vowels ['u']
→ 5

⇒ Functions

- functions are a block of code which take an input perform an operation and return

eg Syntax → def funct_name (argument) :
body.

Spiral

Date

eg
```
def calc(x):
    r = 2 * x ** 2      ← 2x²
    return r
```

- functions needs to be called by a function call for operating.

Syntax :- funct_name (argument)

eg
```
a = int(raw_input("Enter value of x"))
print calc(a)
```
this calls that func^n

→ Types of functions

1.) Built in function

- pre defined by python, eg len(), int()

2.) Function in module

- pre defined in particular module eg, sin(), cos() from math library.

3.) User defined functions

- functions made up by a user eg calc() in above example.

Date.....

eg
```
def calc_sum(x):
    r = 2*x**2      ← 2x²
    return r
```

- functions needs to be called by a function call for operating.

Syntax :- funct_name (argument)

eg
```
a=int(raw_input("Enter value of x"))
print calc(a)
```
this calls that func^n

→ Types of functions

1.) Built in function

- pre defined by python, eg len(), int()

2.) Function in module

- pre defined in particular module eg, sin(), cos() from math library.

3.) User defined functions

- functions made up by a user eg calc() in above example.

Scanned by CamScanner

Date.....

⇒ Scope of variables

→ Global scope :- Variable declared in _main_ has a global scope and can be used all over the program, even in func"s.

→ Local scope :- Variables declard in a func" body has a local scope and can be used in only that function and their sub blocks.

eg
```
def calsum(x,y):
    z = x+y        <- x, y, z local variables
    return z
num1 = int(raw_input("No. 1"))
num2 = int(raw_input("No. 2"))
sum = calsum(num1, num2)
print "sum of given no is", sum.
```
global variables

Q1 WAP to find largest of 3 given numbers.

Sol^n
```
x = float(raw_input("Enter No"))
y = "  "  ("Enter No")
z = "  "  "  "
max = x
if (y > max):
    max = y

if (z > max):
    max = z
print "largest No is", max
```

Spiral

Date.....

WAP that returns the sum of unequal no out of 3 no's.

```
def sum (a, b, c):
    if (a != b and a != c):
        sum = sum + a
    if (b != a and b != c):
        sum = sum + b
    if ((c != a and c != b):
        sum = sum + c
    return sum.

x = int (raw_input ("No 1")
y = "  "  "  ("  2")
z = "  "  "  ("  3")
sum = sum (x, y, z)
print "sum of unequal no is", sum.
```

★ Read inf Questions from Pg 342 to 346.

Date

⇒ String Manipulation

- String are a set of characters.
- Strings are immutable.
- Indexing is done like

0	1	2	3	4	5
P	Y	T	H	O	N
-6	-5	-4	-3	-2	-1

Syntax → string_name = 'string_val'

eg subject = 'computer'

- We can access string like

string_name [Index]

↳ both types.

→ length of String

Syntax → len (string_name)

eg: len (subject) = 9.

- Max element index posible is len-1.

∴ subject [9] ← error.

↳ len (Not possible.

Spiral

→ Traversing a string

- Accessing each element individually is known as traversing.
- Program to traverse any string.

eg

```
name = 'superb
for ch in name
    print ch, '-',
```

output → S-U-P-U-R-B.

WAP to display a string in Reverse without making new string.

```
str1 = raw_input("Enter string")
length = len(str1)
for a in range(-1, (-length-1), -1)        ← To include last index
    print str1[a]
```

Date

→ String operations

(i) Contatination (+)

- It joins 2 strings to form a new string.

  eg 'sar' + 'thak' = sarthak

(ii) Replication (*)

- The operator multiplies or replicates the string 'n' no of times

  eg 3*'GO' => GOGOGO

- Needs 1 number and 1 string

  'A' 's' not Valid

iii) Membership operator

- Two types → 'in' and 'not in'
- They check if given string contains a particular substring.

  eg '12' in 'x42' ← false
  '12' not in 'x42' ← True

A Case Sensitive operations H and h not same.

Spiral

Date

(iv) Comparision Operators

- these are, ==, !=, <, >, <=, >=.
- Ascii and Unicode's are compared

eg 'a' == 'a' ← True
'abc' == 'a' ← false

- Ascii codes → 0 to 9 = 48–57
A to Z = 65 to 90
a to z = 97 to 122

→ String slicing

- Cutting out a part of a string is called string slicing.

Syntax → String_name [start : stop]

* Start by default 0th index
* Stop by default (n-1)th index, (n is length of string)

eg word = 'AMAZING'

word [0:7] = Amazing
word [0:3] = Ama

Spiral

→ String functions and methods

1.) Capitalize() :- Capitalizes 1st character.

Syntax → string_name.capitalize().

eg str = "Ilove India"
str.capitalize()
=> I love India.

2.) find(sub[, start[, end]])

- Returns lowest index where sub is found in string

Syntax → string_name.find(sub, start, end)

eg str = Hello
str.find(l)
=> 2.

3.) isalnum()

- Checks if string has all number or alphabet with at least 1 alphabet/character.

Syntax → string_name.isalnum()

Date.....

4.) isalpha()

· checks if all alphabets

syntax → string_name.isalpha().

5.) isdigit()

· checks if all are digits

syntax → string_name.isdigit.

6.) islower() and isupper()

if all in lower case,

Syntax → string_name.islower()

if all in upper case.

Syntax → string_name.isupper

7.) isspace()

· checks if string has only white spaces.

syntax → string_name.isspace().

Date

8) lower() and upper()

returns a copy of list in lower case

Syntax → String-name.lower()

Returns a copy in Upper case

Syntax → String-name.upper()

9) lstrip([chars])

- Removes the leading characters

Syntax → string-name.lstrip([chars])

eg str = 'Hello'
str.lstrip('Hel')
⇒ lo

10) Rstrip([chars])

- Removes trailing characters

Syntax → String-name.rstrip[[chars])

eg str = Hello
str.rstrip(lo)

⇒ Hel

★ Read Programs at Page No 372 - 373.

Spiral

Date

→ Lists (continued)

→ list operations

(i) Joining lists (+)

. It joins 2 lists

eg
```
list1 = [10, 11, 12]
list2 = [14, 16, 18]
lst = list1 + list2
print lst
```

∴ [10, 11, 12, 14, 16, 18]

(ii) Replicating lists

. It replicates a list by n no. of types

```
lst1 = [10, 11, 12]
lst2 = lst1 * 2
print lst2
```

= [10, 11, 12, 10, 11, 12]

* Rest Done before.

Spiral

Date.....

→ List functions

(i) Extend ( ).

· It helps add a list or multiple elements to the end of a given list

Syntax → list_name. extend (list/items)

eg
```
t1 = [1, 2, 3]
t2 = [4, 5]
t1. extend (t2)
```

(ii) Remove ( )

· It helps remove an element not on the basis of posⁿ but on the basis of value

Syntax → list_name. remove (item – value)

eg
```
t1 = [1, 2, 3]
t1. remove(2)
print t1
```

[1, 3]

* removes the 1st occurrence and not all of them

Date.....

(iii) pop()

. It deletes and returns the element on a given posⁿ

Syntax → variable = list_name.pop(posⁿ).

eg
```
t1 = [1, 2, 3, 4]
item = t1.pop(3)
print Item
```

⇒ 4

(iv) Delete

. It can delete 1 or a group of values from list on basis of posⁿ.

Syntax →
```
del list_name (Index)
del list-name (start : stop)
```

Q WAP to implement list operations with menu.

```
Ch = 0
list = []
While True:
    print "List Mani"
    print "1. add element"
    print "2. Modify"
    print "3. Delete"
    print "4. Sort"
    print "5. Display"
    print "6. EXIT"
```

Spiral

Date

```
ch = int (raw input ("Enter choise 1 --- 6 ")
if ch == 1
    print "1. add one element"
    print "2 addr a list"
  if ch1 == 1:
      item = int (raw_input ("element")
      pos = int (raw_input ("position")
      list. insert (item, pos)
  elif ch1 == 2:

    list2 = eval (raw_input ("enter list")
    list1.extend (list2)
  else :
      print "Wrong input"
  print "Added"
el if ch == 2 :
    pos = int (raw_input ("position to modify")
    item = int (raw_input ("Enter new value")
    old = list[pos]
    list[pos] = item
    print old, "modified to", new
elif ch == 3 :
    print "1. by posn"
    print "2. Del by value"
    ch2 = int (raw_input ("Enter 1 or 2")
    if ch2 == 1
        pos = int (raw input ("Enter pos")
        item = list.pop (pos)
        print item, "Deleted"
    elif ch == 2
        item = int (raw_input ("Enter Item")
        pos = list.remove (item)
        print "succesfully Deleted"
```

```
        else:
            print "Wrong choice"
    elif ch==4:
        print "1. Ascending"
        print "2. descending"
        ch3= int (raw_input ("Enter choise 1 or 2")
        if ch3==1:
            list.sort()
        elif ch==2:
            list.sort (reverse=true)
        else:
            print "Invalid choise"
    elif ch==5:
        print list

    elif ch==6:
        Break

    else:
        print "Valid choise are 1---6"
```

Date

⇒ Dictionaries

- It is a data structure which is saved similar to any data structure but instead of indexing it contains key: value pairs

Syntax → dict_name = { key: value, key: value, ... }

eg teachers = { Dimple: Comp Sc., Karen: Sociology
Harmeet: Maths, Sabah: chem }

→ Accessing elements of a dictionary

- To find an element, we must know the key.

Syntax :- dictionary_name [key]

eg teachers ['karen']

⇒ Sociology

→ Traversing a dictionary

Syntax → for items in dictionary_name :
process item

eg for key in teachers :
print key, &teachers [key],

Spiral

Date

→ Accessing keys and values

(i) Accessing keys

Syntax → dictionary_name.keys()

(ii) Accessing Values

Syntax → dictionary_name.values()

→ Characteristics of dictionary

1.) Unordered Set
2.) Not a sequence
3.) Indexed by keys, not no..
4.) keys must be unique
5.) Mutable
6.) Internally stored as mapping.

→ Working with dictionary

1.) Creating dictionary

(a) Initializing a dictionary ie making key:value pares

eg Employee = {'name': 'John', 'Salary': 1000, 'age': 14}

(b) Adding key value paire to empty dictionary.

Spiral

Date.....

employee = {} or employee = dict()

employee['name'] = 'John'
employee['salary'] = 10000
" ['age'] = 24

(c) Creating dict from name and value pairs

(i) Specify key:value to dict() constructor

eg Employee = dict('name'='John', 'salary'=10000)

Employee = {'name':'John', 'salary':10000}

(ii) Specify comma separated key:value pairs

eg Employee = dict("name":'John', 'salary':10000)

(iii) Specify key separately and values separately

eg Employee = dict(zip('name', 'salary'), ('John', 1000

2) Adding Elements to Dictionary

Syntax → dictionary_name[key] = value

* Similar key should not exist.

3) Updating elements in Dictionary

Syntax → dict-name [key] = new value

4) Deleting element in dictionary

By Del command

(i) Syntax → del dictionary-name [key-name]

eg del employee ['salary']

employee = {'name': 'John'}

(ii) By pop ()

Syntax → item = dictionary-name.pop (key)

5) Checking existance of key

Syntax → key in dict-name
key not in dict-name

→ Dictionary functions

1.) len ()

- Gives count of key : value pairs

Syntax → len (dict-name)

2.) clear ()

- Removes all items -

Syntax → dictionary-name . clear ()

Speral

Date

3) Get()

• Gives value of the given key.

Syntax → dict-name.get(key,[default])
↳ if key not found

4) has_key()

• Checks presence of a key.

syntax → dict-name.has_key(key-name)

5) items()

• Returns all key:value pair in a sequence

Syntax → dict-name.items()

6) Update()

• Merges 2 dictionaries by adding or replacing values

Syntax → dict1-name.update(dict2-name).

7) cmp()

• Compares 2 dict on basis of key and values

Syntax → cmp(dict1, dict2)

Date.....

⇒ Tuples

- Tuples are used to store sequence of data and are immutable.

Syntax →. T = tuple ( ) or t = ( )
T = tuple (4,) or t = (4,)
T = tuple (Hello) or t = (H, e, l, l, o)

→ Accessing elements of tuple

Syntax → tuple-name (index)

eg vowels = ('a', 'e', 'i', 'o', 'u')

vowels [3] = 'o'

→ Traversing tuples

Syntax → for item in tuple :
process item.

eg for i in vowels :
print vowels [i]

→ Tuple operations

(i) Concatination +

Joins two tuples.

Spiral

Date.....

Syntax → tup1 + tup2 + tup3

2.) Replication

Syntax → tup1 * Numeric-value

3.) Slicing

Syntax → Tuple name (start : stop : step)

4.) Unpacking

we assign variables for each tuple value.

Syntax = variable1, var 2 . . . Variable n = tuple-name

→ Tuple functions

1.) len ( ) :-

Syntax → len (tuple-name)

2.) max ( )

Syntax → max (tuple-name)

3.) min ( )

Syntax → min (tuple-name)

Date.....

4.) cmp()

Syntax → cmp(tup1, tup2).

→ 1 tup1 > tup2

0 equal

-1, tup1 < tup2

Date

WAP for Addition and sub of 2 matrices.

Sol

```
def make_matrix(n):
    return [[random.randint(1,9) for i in range(n)]
            for j in range(n)]

def print_matrix(matrix, n):
    for i in range(n):
        for j in range(n):
            print(matrix[i][j], end=" ")
        print()
    print()

def operate_matrix(matrix1, matrix2, n, func):
    result_matrix = []
    for i in range(n):
        temp_matrix = []
        for j in range(n):
            temp_matrix.append(func(matrix1[i][j], matrix2[i][j]))
        resultant_matrix.append(temp_matrix)
    return resultant_matrix
add_element = lambda a,b : a+b
sub_element = lambda a,b : a-b
n=3
matrix1 = make_matrix(3)
matrix2 = make_matrix(3)
operate_matrix(matrix1, matrix2, n, add_elements)
operate_matrix(matrix1, matrix2, n, sub_elements)
```

Spiral

Date.....

```
print ("Matrix 1 :")
print_matrix (matrix1, n)
print ("Matrix 2 :")
print_matrix (Matrix 2, n)
print("matrix 1 + matrix 2")
print_matrix (add_matrix, n)
print ("matrix 1 - matrix 2")
print_matrix(sub_matrix, n)
```