GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

Unit – III

Control Flow Functions

PART-B

Boolean Expressions:

It is an expression that is either true or false

Eg, >>> 5 = = 5	Print (5 = = 5)
True	True
>>> 5 = = 6	Print (5 = = 6)
False	False

The examples use the operator = = which comparies two operands and produces True if they are equal, and False otherwise. True and False are special values the belong to type bool;

Eg >>> type (True) >>> type (False)

O/P < type 'bool' > O/P < type 'bool' >

The condition in a selection statement often takes the form of a comparison. The result of comparison is the Boolean value (either True or False)

Comparison operators (relational operators)

= = Equals, = not equals

< Less than, > Greater than

< = less than or equal, > = Greater than or equal

note: '= = 'means equal, '= 'means Assignment

eg >>> 4! = 4 False

>>> 4 > = 3 True

Conditional Execution (Statements)

If : - (one way selection statements)

In order to write useful programs, we almost need the ability to check conditions and change the behaviour af the program accordingly conditional statements give us this ability.

Syntax af if statement:

if x > 0 :

Print 'x is positive '

Or

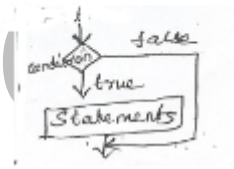
Print ("x is positive ")

Syn:

if < condition > :

< Sequence of statements >

semantics:



in the syntax

If x > 0:

Condition (Boolean expression)

The Boolean expression after the if is condition if it is true then the indented statement gets executed if not nothing happens

Eg, pgm to check the given no is positive

```
Print ("Enter a number " )
number = int (input ())
```

O/P

```
Enter a number 10
The Number is positive
```

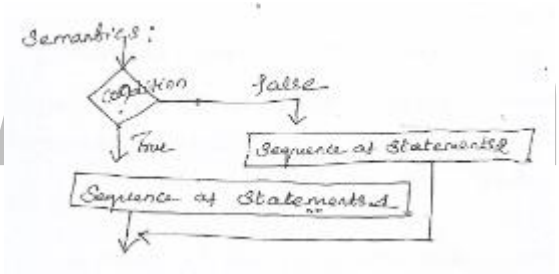
```
if num >0:
        Print ("The Number is positive " )
```

Alternative Execution (if else): (two way selection statements)

The second form of the if statement is alternative execution in which there are two possibilities and the condition determines which one gets executed. The syntax is

```
if < condition >:
    < Sequence of statements – 1 >
Else:
    < Sequence of statements – 2 >
```

Semantics:



eg

```
∩ = int (input ("Enter the value " ))
if ∩ % 2 = = 0:
        Print ('x is even ' )
else:
        Print ('x is odd ' )
```

O/P

```
Enter the value 4
x is even
```

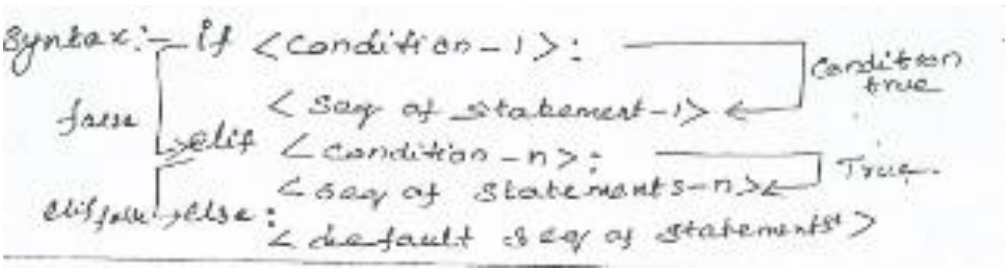
Since the condition must be true or false exactly one of the alternatives will be executed. The alternatives are called branches.

Chained Conditionals (if – elf – else statements)

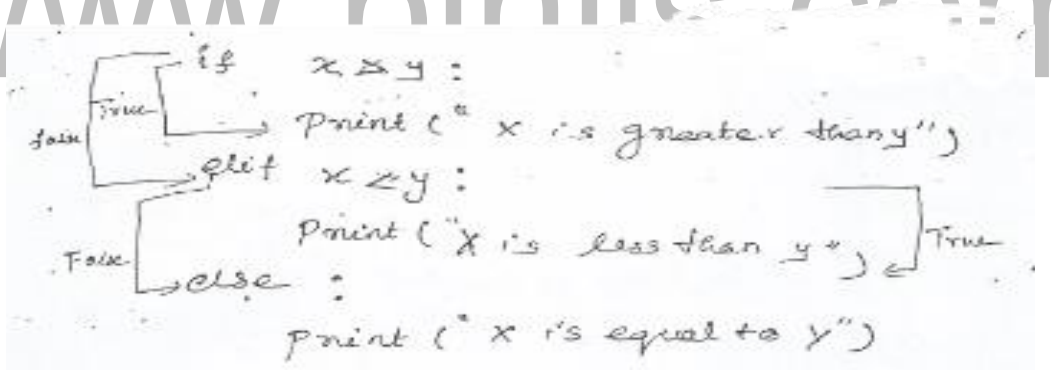
Or

Multiway if statements:

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is chained conditional



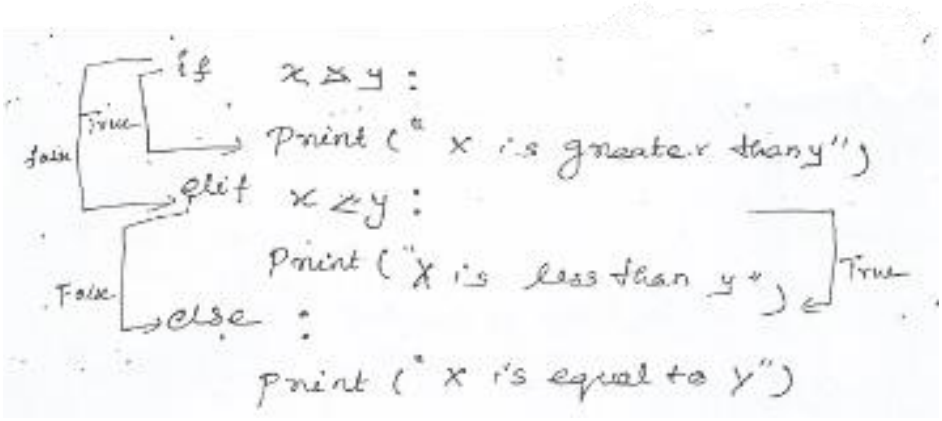
	O/P
eg: x = int (input (“Enter x value “)	Enter x value 10
y = int (input (“Enter y value “)	Enter y value 5
	X is greater than y



Elif is an abbnevation of “else if “– exactly one branch will be executed. There is no limit on the number of elif statements if there is an else clause it has to be at the end.

Nested Conditionals:

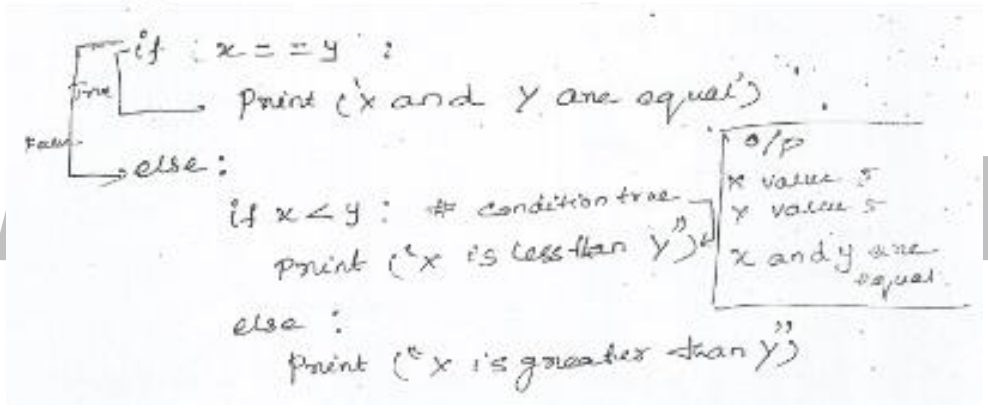
One conditional can also be nested with in another. We could have conitten the syntax like this.



Greatest among 2 numbers pgm to check two number is equal or greater or lesser.

x = int (input (“x value “)

y = int (input (“y value “)

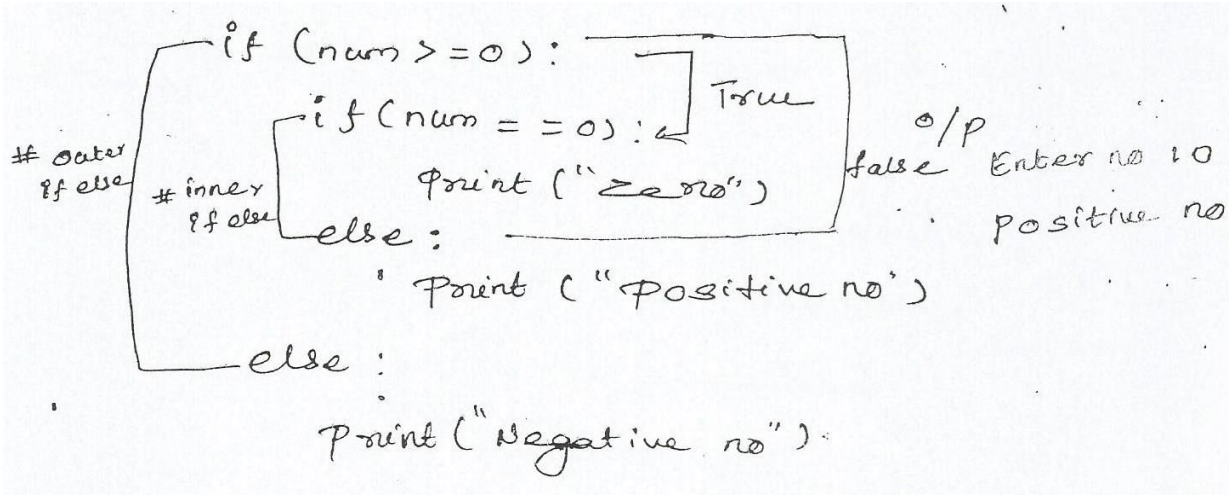


The outer conditional contains 2 branches. The first branch contains simple statements and second branch contains another if statements. Which has two branches of its own.

Eg – for vested conditionals :

Eg, pgm to find the number is positive or zero or negative

Num = int (input (“Enter no “))



Iterators (Looping)

In some situation there is a need to repeat a set of instructions in specified numbers of times or until a particular condition is satisfied. The repetitive operations are done through a loop control structure.

Loop: is defined as the block of statements which are repeatedly executed for certain number of times.

Loop consists of 2 parts

(i) Body of the loop and
(ii) Control statement (test condition)

Steps:

(i) Initialization condition variable
(ii) Test the control statement
(iii) Executing the body of the loop depending on the condition
(iv) Update the condition variables.

Types:

While loop

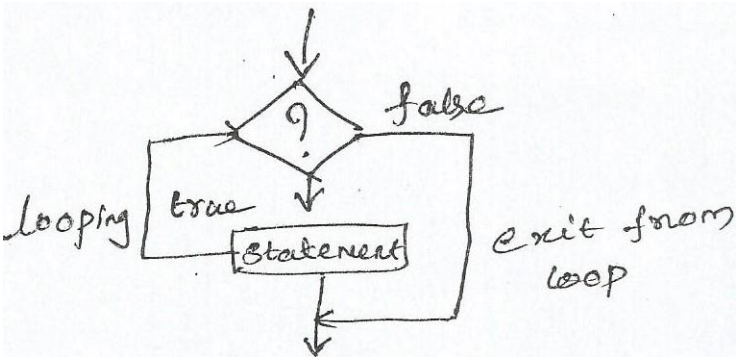
For loop

The while loop (Conditional iteration):

Conditions iteration requires that a condition be tested with in loop to determine whether the loop should continue. Such a condition is called loop's continuation condition if the condition is false the loop ends if the condition is true. The statements in the loop are executed.

Syn: While < Condition >:

< Sequence of statements >



Eg pgm to find the sum of n numbers using while – loop

>>> i = 1

>>> Sum = 0

>>> n = 10

>>> While (i < = n): ≠ i value starts from 1 ton = 10

Sum = Sum + i ≠ i value added O/P with sum

i = i + 1 ≠ incrementing i sum of n number = 55

>>> Print ("Sum of n number = ", Sum)

The for Loop (Definite interation):

Each repetitation of the action is known as pass or an interation. There are two types of loops.

- definite → Repeat an action a pre defined no. of times

Eg for loop (we know the number of iteration)

(Condition required) - We don't know the no. of iteration

Indefinite: that perform the action until the program determines that it needs to stop.

Eg, while loop eg while (1) → always True

For loop: is another repetitive control structure and is used to execute the set of instructions repeatedly until the condition becomes false.

The assignment incrementation / decrementation and condition checking is done in for statement only.

Syntax:

For < variable > in range (< an integer expression >):

< Statement 1 >

:

< Statement n >

Or

For < variable > in sequence

< Statements >

Explanation: The first line of code in a loop is called as loop header. The colon (:) ends the loop header. The loop body comprises the statements in the remaining lines of code.

Range (): → It is used to generate sequence of numbers.

Eg range (10) → generate 0 to 9

Syn: range (Start, Stop, Step size)

- Step size default is 1

Eg, O/P

Print range (10) → range (0, 10)

Print (list (range (10)) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Print (list (range (2, 20, 3)) → [2, 5, 8, 11, 14, 17]

Eg >>> for word in “python “:

Print ("Char in word ", word)

O/P Char in word p

Char in word y

Char in word t

Char in word h

Char in word o

Char in word N

Eg, executing a statement a given number of times

>>> For each pass in range (4):

Print ("It's alive! " End = ")

O/P

It's alive! It's alive! It's alive! It's alive

Eg, >>> number = 2

>>> Exponent = 3

>>> Product = 1 ≠ for each iteration each pass variable holds one value In range.

>>> For each pass in range (exponent):

Product = Product * number

O/P Print (product, end = "")

2 4 8

Eg, pgm to find sum at 1 – 10 number using for

>>> Sum = 0

>>> For i in range (1, 11): ≠ start = 1

≠ End = 11 – 1 = 10

Sum = Sum + i

>>> Print ("Sum = ", Sum) O/P 55

Traversing the contents of a Data sequence

Syn:-

For < variable > in < sequence >:

Eg >>> for number in [1, 2, 3]:

Print (number, end = "")

O/P 1, 2, 3

≠ Just printing each element in list and "" → gives space after the element

Eg

>>> For char in "Hi there ":

Print (char , end = "")

No space

O/P

Hi there

≠ printing each char in the string

≠ ""gives space at the end of each character

↓

Space

Nested Loops:

The loop with in the loop is called nested loop.

Syn:-

For < Var > in < Sequence >:

For < var > in < Sequence >:

White < exp >:

While < exp >:

Statements

Statements

Statements

Statements

Eg for i in range (1, 5):

For j in range (1, 3):

≠ j = 1, 2

Print (“x “)

O/P x

X

X

i = 1	i = 3
j = 1	j = 1
j = 2	j = 2
i = 2	i = 4
j = 1	j = 1
j = 2	j = 2

While loop with else:

>>> i = 0

>>> while i < 5: ≠ First i value is 0 & in cremented to 5 in each step

Print (i, “less than 5 “)

i = i + 1

Else:

Print (i, “is not less than 5 “)

Explanation: Here initially i value is zero and incremented by 1 by checking the condition once the condition fails else part will be executed

O/P

0 less than 5
1 less than 5
2 less than 5
3 less than 5
4 less than 5
5 is not less than 5

Loop control statements (un conditional statements)

Loop control statements change execution from the normal sequence. When execution leaves

Break → Terminates the loop statement and

Syn: break — transfers execution to the statement Immediately following the loop

Continue → Causes the loop to skip the remainder

Syn: continue — af its body and immediately Its condition prior to reiterating

Pass: → The pass statement in python is used When a statement is required.

Syn : pass — syntactically but you do not want any code to execute.

break	→	Terminates loop & transfer to next immediate statement
Continue	→	Continues the working of loop & skip the other statements below the Continue.
Pass	→	∩∪ *cc* operation, used with in loop or function that is not yet implemented but want to implement it in future.

Sum = 0.0

Eg

```
While True
    Data = input ("Enter a number or To Quit press space):
    If (data = = "" ):
        Break
    Number = float (data) ≠ float (10)
                ⇓
              10.0
    Sum + = number ≠ 0.0 + 10.0
```

```
Enter a number or
To Quit press space:
        10
The sum is 10.0
```

Print (“The sum is “, sum)

Eg, for i in “Computer “:

If (i = = p):

Break

Print i current letter is “, i)

O/P

Current letter is C

O

M

Continue:

For i in “Computer “:

If (i = = p):

Continue

Print “i current letter is “i)

O/P

Current letter is C

O

M

U

t

e

r

Pass:

For i in range (1, 6):

If (i = = 3): ≠ when the i value increments as 3 1

Pass it will skip the if condition

Print (i) ≠ pass = null a relation

O/P

2

3

4

5

Fruit full functions:

- Function with return values are called as fruit full function.
- Function that does not have a return value is called a procedure.

Return values:

Some of the built in functions we have used, such as the math functions, produce results. Calling the function generates a value. Which we usually assign to a variable.

Import Math

Eg e = math.pi to find exponential

Print ("The value of sine of pi is ", math sin (a))

≠ returning the value of sine of pi

The return statement may or may not send back any values to the main program (calling program) it can done using the return statement.

Eg def sum (arg 1, arg 2): ≠ fun definition ≠ arg 1 = 100

Total = arg 1 + arg 2 ≠ arg 2 = 200

Print ("Inside fun: ", total) ≠ Top to bottom

Return total approach

Total = Sum (100, 200) ≠ fun cau

Print ("Outside function: ", total)

O/P

Inside the function : 300

Outside the function : 300

Eg, pgm to find distance b/w two points, given by the co-ordinates (x_1 , y_1) and (x_2 , y_2)

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Eg def distance (x_1, y_1, x_2, y_2):

dx = $x_2 - x_1$

dy = $y_2 - y_1$

Squared = dx ** 2 +dy ** 2

Result = math: sqrt (Squared)

Return = result

Ans = distance u (1, 2, 4, 6)

Print ("Distance b/w 2 points = ", ans)

O/p Distance b/w 2 points = s.o

Scope of variables:-

All variables in a program may not be accessible at all locations in that program. This depends on where you have declaned a variable.

1. The scope of variable determines the portion of the program where you can acess a particular Identifier Two basic scopes :-
    1. Local scope
    2. Global scope

Global & local variables:-

Variables that are defined inside a function body have a local scope and those defined outside have a global scope.

2. Local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed through out the program body by all functions.

a = 5 ≠ global variable

def fun ():

x = 3 ≠ local variable

Print ("value of x inside fun ", x)

Print ("value of a is: ", a)

Fun ()

Print ("value of a is ", a)

Print ("value of x is ", x)

↓

Error

O/P

value of x inside fun 3

value of a is = 5

Value of a is 5

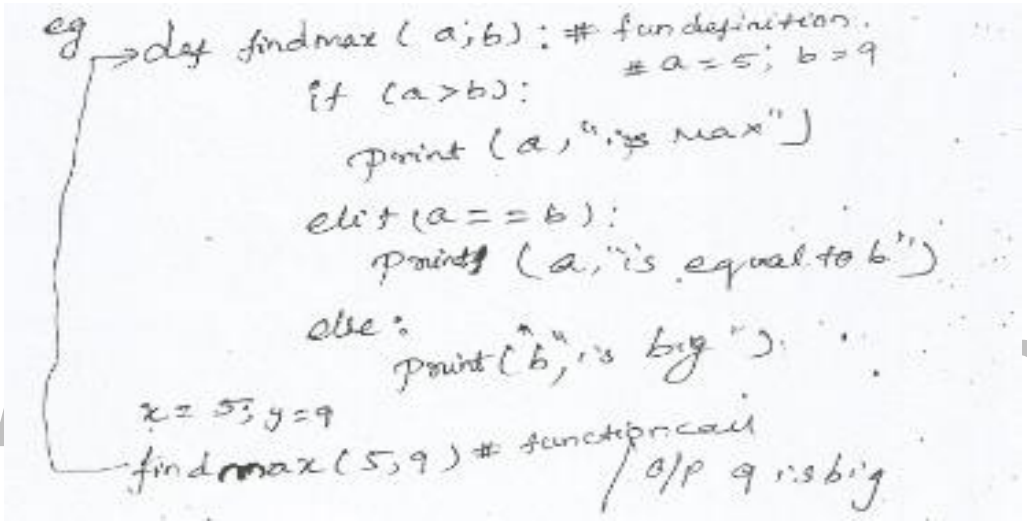
Name Error: name 'x'

is not defined

Function Parameters:-

Parameters are just like variables except the values of the variables are defined when we call the function and are already assigned values when the functions runs.

Parameters are specified within the pair of parentheses in the function definition separated by commas. When we call the function. We supply the values in the save way. The names given in the function definition are called parameters whereas the values you supply in the function call one called arguments.



Function Composition or Nested function

You can call one function from within another This ability is known as composition.

```
Eg   def   square (s):
         Return (SA S)
     def   quad (n):
           Nes = Square (n) * square (n)
           Return nes
a = int (input (“Enter a value “ )
Ans = quad (a)
Print (ans)                          O/P Enter a value
```

2

16

Eg, def outer ();

def inner (0 :

Print (“Inside fun “)

Print (“Welcome “)

Print (“Outside function “)

Print (“In calling fun “)

Inner ()

Outer ()

O/P

Outside function

In calling fun

Inside fun

Welcome

Recur sine Function: - Direct

In Direct

A recur sine function is a function that cause itself. To present a function from repeating itself indefinitely, it must contain at least one selection statement.

- This statement examines a condition called a base case to determine whether to stop or to continue with another recursive step.

Direct: - fun call is present inside a fun definition

Eg def factorial (n):

If n = = 0:

Return n * fact (n – 1)

n = int (input (“Enter the number “)

Fact = factorial (n)

Print (“factorial = “, fact)

Eg pgm to find greatest common Divisor

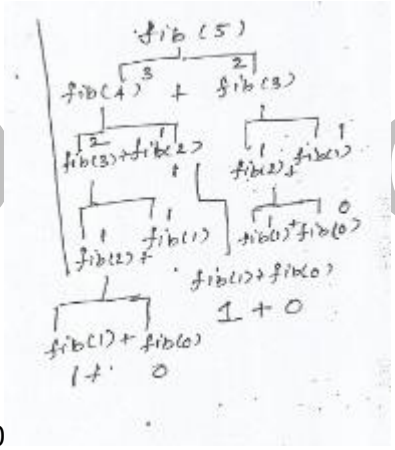
```
def gcd (x, y):                                  hcf or GCD

                                                 12, 14 = 2
    If (y = = 0):                                12 → 1,2,3,4,6,12
                                                            Rem = 0
        Return x                                 14 → 1, 2,7,14
    Else:
        Return     gcd (y , x % y )

n1 = int (input (“Enter a value “ ) ;
n2 = int (input (“Enter a value “ ) ;
Ans = gcd (n1, n2)                               O/P  12, 14
Print (“Gcd = “, ans )                           Gcd = 2
```


Exponential, or power of numbers

```
def  power  (base, exp):
    If (exp = = 1):
        Return (base)
    Else:              2    *          2      1
        Return (base * power (base, exp – 1)
b = int (input (“value “ ))
e = int (input (“value “ ))
Ans = power (b, e)
Print (“Exponential = “, ans )
```

Leap of faith: - (follow the flow of execution and checking the recursive cause works Correctly)

```
def   Fibonacci (n):
    If   n = = 0
        Return 0
    elif   n = = 1:
        Return 1
    Else:
        Return Fibonacci (n – 1) + Fibonacci (n – 2)
n = int (input (“Enter a value “ )
For i in range (n):
    Print (fib (i))
```



O/P 0

1

1

2

3

5

In Direct – calling from outside of the function definition

```
def   fact   (x):
    F = 1
```

```
For i in range (1, n + 1)
        f = f * i
        Return f
n = int (input ())
r = int (input ())
Ncr = fact (n) / fact (r) * fact (n – r)
Print (“NCR = “, ncr )
```

Pgm to find square root of a number

```
n = 8
Sqnt num = n ** 0.5
Print (“The square root is “, sqrt num )
```

Or

Using function find square root

```
def     square (n)
        num = ∩ * * 0.5
        Return num
n = int (input (“value “ ))
Nes = square (n)
Print (nes)
```


Strings

A String is a sequence:-

String is a sequence of characters you can access the characters one at a time with bracket operator or index

```
>>> Fruit   = ‘banana ‘
>>> Letter = fruit [1]
```

>>> Print (Letter) O/P a

The 2nd statement selects character number 1 from fruit and assign it to letter. The expression in bracket is called an index. The index indicates which character in the sequence you want to access.

Explanation: String is a sequence of characters enclosed

& Def in single or double quotes

Syn Variable name = “string “

Variable name = ‘string ‘

Eg Fruit = ‘banana ‘

↓ ↓

Variable String sequence of character

Fruit

b	a	n	a	n	a
0	1	2	3	4	5

Letter = fruit [1]

↓ Index → used to a

Index access a char in string

Def: String is a group of character or special characters enclosed in single or double Quotes.

Accessing values in strings :-

- [ ] → index bracket is used to access character in string. Access one at a time.
- [:] → Range of characters

2 types of index

Positive index → counting forwards

Negative index → counting backwards

Eg >>> Str = “hello “

>>> Print (Str [0]) → O/P h

>>> Print (Str [- 1]) → 0

String slices :-

Syn:- variable name [start : end : step size]

A segment of a string is called a slicesis similer to selecting a character

>>> s = " Monty python "

>>> Print s [0 : 5]

Range of characters

O/P Monty

>>> Fruit = ' banana ' ""

>>> Fruit [: 3]

O/P ban (if you omit the first index before colon the slice starts at the beginning

at string, if you omit the second index, The slice goes to the end of the

string)

eg

>>> Fruit = " banana "

>>> Fruit [3:3] → OP ' '

if the first is greater than or equal to the second the result is Empty string it is represented by two Quotation mark.

eg, Str = " python "

Print (str [0]) → p

Print (str [- 1]) → n

Print (str [0 :]) → python

Print (str [0 : 3]) → pyt

Print (str [0 ; 6 : 2]) → p t o

Print (str [3 :]) → hon

Print (str [: 3] → pyt

Print (str [:]) → python

Strings are Immutable

immutable → not possible to change or delete the existing string the values or characters in the string cannot be changed.

eg str = " Hello python "

str [ ] = M ⟹ error : object does not support item assignment .

But you can create a new string that is a variation on the orginal.

>>> greating = " Hello world ! "

>>> new – greating = ' J '+ greeting [1 :]

>>> Print (new – greeting)

→ O/P Jello world !

String Functions:-

- Len ( ) → returns the length of the string ie how many character present in the string

Syn : len (str)

eg >>> word = ' python "

>>> Print (" The length of string 1's " len (word))

O/P the length of string is 6

- Max ( ) → returns the maximum alphabetic character from the string.

Syn : Max (str)

>>> S = " python "

>>> Print (Maxcs) O/P Y

- Min ( ) → returns Minimum alphabeticas character from the string

Syn min (str)

>>> s = " python "

>>> print (min (s)

Count ( ) :- Count how many times the particular char is present in the string.

Syn :- Variable

String Methods () :-

1. Upper () :- Conuerts lower case letter to upper case letter.
2. Lower () :- Conuerts upper case letter to lower case letter.

Syn:

>>> Str = " Hello world "

>>> Print (Str. lower ()) O/P

>>> Print (Str. upper ()) hello world

HELLO WORLD

3. Count () :- Count how many times the particular character is present in the string.

Syn :- Variable count (' Character ')

eg, >>> Str 1 = " Welcome "

>>> Print (Str 1. Count (" e ") O/P 2

4. Find () :- This method is used to find the location of the particular character in the string.

Syn : Variable name . find ()

>>> Str = " First year "

>>> Print (Str. find (" i ") O/P 1

if character not found → it returns - 1

5. Join () :- (Join the string with the given char)

This method adds the specified characters in between every character in the string.

eg, Str = " hello world "

Print (" , " . join (str)) O/P h,e,l,l,o, w,o,r,l,d

Used to combine 2 string also

eg, Str 1 = [" hello " " world "] O/P hello world

Print ("" . Join (Strl)

6. String reversed () :-

To reverse the given string used reversed () function is used along with Join ()

eg, >>> book = ' problem solving '

>>> Print (" " Join (reversed (book))

O/P given me / borp

7. Split () :-

Splits a single multi word string in to list of individual words and that are seperated by the given character or spaces.

eg, Str = " Hi How are you "

Print (Str. Split ()) → O/P [' Hi ' , ' How ' , ' are ' , ' you ']

Print (Str. Split (H)) → [ ]

8. Replace () :-

It can taken an orginal string and returnan updated string with some replacement.

Syn : variable = String. replace (" old " , " new ")

eg Str = " Problem Solving Python "

a = (str . replace (" python " , " book ")

Print (a)

O/P problem solving book

String Swap case () method () :-

⟹ returns a copy of the string in which all the case base characters have had their case swapped.

Str1 = " Hello "

Print (Str1 . swapcase ()) O/P hELLO

- Conuerts the upper case character to lower case and lower case character to upper case.

String Operaters:-

1. Concatenation :- (+)

It is used to concatnenate two strings

eg Str1 = " Hai "

Str 2 = " Welcome "

Str 3 = Str 1 + Str 2 O/P Hai welcome

Point (str 3)

- Repetition :-

Operator is used for repetition (Repeat n no. of times)

eg, Str 1 = " Hello world "

Str 3 = Str 1 * 3

Print (Str 3) O/P Hello world, Hello world, Hello world

- Boolean methods :-

String methods that will evaluate to a Boolean value. The methods are usefull when we creating forms for the user to fill. eg, for postal code it will accept only numeric string.

Methods :-

Str . isalnum ( ) → Check whether the string contain alpha numeric characters.

Str. isalpha ( ) → String consists only alphabetic char

Str. is lower ( ) → Check whether the string characters are in lower case.

Str. is upper ( ) → Check thestring characters are in upper case.

Str. is numeric ( ) → Checks the string contains only numeric character.

Str. is space ( ) → Checks all the characters are white space character.

Str. is title ( ) → String is in title case. (Python)

↓

title case

- String Modules :-

The modules contains no . of methods to process std python string

eg import String

text = " Programming Subject "

Print (" upper ( ) = " , text.upper ( )

Print (" lower ( ) = " , text.lower ( )

Print (" Split ( ) = " , text . split (i)

Print (" replace ( ) = " , text . replace (" subject " , " book ")

Print (" find 11 = " , text . find (p)

Print (" Count 11 = text . count (m)

O/P upper ( ) = PROGRAMMING SUBJECT

lower ( ) = programming subject

Split ( ) = [' program ' , ' ng ' Subject]

replace ( ) = programming book

find ( ) = 0

count ( ) = 2

List as array

Instead of array we can use list and list functions & Methods.

List in python is a collection of items which can be of any type array is a collection of items of single type. When comparing list and array, list is more flexibility.

List is also a dynamic mutable type and this means you can add and delete elements from the list at any time

To define the list :-

mylist = [1, 2, 3, 4, 5, 6, 7]

Acessing using index & slicing :-

Print (mylist [2]) → 3 (O/P

my list [2] = 100

Print (mylist) ⟹ [1, 2, 100, 4, 5, 6, 7]

Print (mylist [2 : 5]) ⟹ [100, 4, 5]

Print ([0 :]) ⟹ [1, 2, 100, 4, 5, 6, 7]

Print ([:]) ⟹ [1, 2, 100, 4, 5, 6, 7]

Print ([: 5]) ⟹ [1, 2, 100, 4, 5]

Basic List Operators :-

- Concentration : (+ operator) → used to join two or more list

eg , a > [1 , 2]

```
b = [ 4 , 5 ]

c = a + b

print ( c ) →   O/P   [ 1, 2, 4, 5 ]
```

- Repetition : - used to repeat the list n times

```
eg   a = [ 1, 2 ]

     c = a * 2

     print ( c ) ⟹ [ 1, 2, 1, 2 ]
```

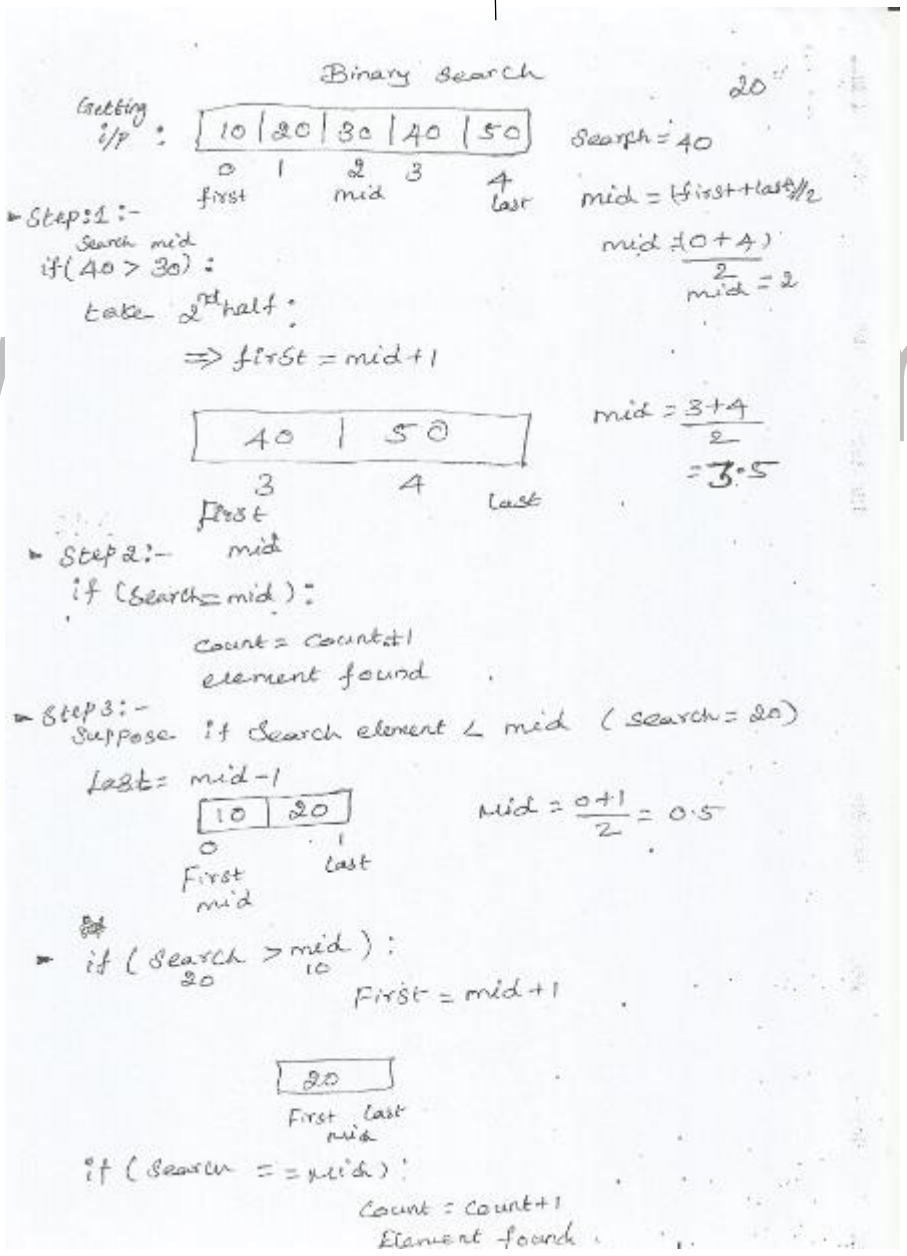
Binary Search using list ;-

```
Print ( " Binary search pgm " )

Count = 1

List 1 = [ ]

n = int ( input ( " Enter the no. of element " )    ≠ 5

Print ( " Enter elements one by one " )

for  i in range   ( 0 , n ) :

     a = int ( input () )

     list 1 . append ( a )

S = int ( input ( " Enter the element to be search ) )

First = 0

Last = n – i

While ( first < = last ) and ( count = = 0 ) :

     mid = ( first + last ) // 2

     if ( list 1 [ mid ] < s ) :

               First = mid + 1

     elif ( list 1 [ mid ] > s ) :
```

```
            Last = mid – 1
        else :
            Count = 1
    if ( count = = 1 ) :
        Print ( " Element found " )
    else :
        Print ( " Element not found " )
```

O/P

```
Enter the no. of elements : 5
Enter elements one by one
10, 20, 30, 40, 50
Enter the element
to be search = 40
Element found
```



Linear Search Using List [ ] : -

```
Print ( " Linear Search " )

Count = 0

List 1 = [ ]

n = int ( input ( " Enter the element size " )

Print ( " Enter elements one by one " )

for i in range ( 0, n ) :

        a = int ( input ( ) )

        List 1 . append ( a )

S = int ( input ( " Enter a search Element " ) )

for i in range ( 0, n ) :

    if ( list 1 [ i ] = = s ) :

        Count = count + 1

        Print ( " The element is present in index " , i )

if ( Count > 1 ) :

        Print ( " Element found " )

else :

    Print ( " Element not found " )
```

O/P

```
    Enter the element size 5

    Enter elements one by one

    10, 20, 30, 40, 50

    Enter a search Element 40

    The element is present in index : 3
```

Element found

Find Minimum element in a list :-

```
list 1 = [ ]
n = int ( input ( " n value = " )
for i in range ( 0 , n ) :
    a = int ( input ( " Enter a elements " )
    list 1 . append ( a )
    minno = list 1 [ 0 ]
for i in range ( 0, n ) :
    List 1 [ i ] < minno
    minno = list 1 [ i ]
Print ( " The minimum No is " , minno )
```

O/P n value = 5

Enter a element

1, 2, 3, 4, 5

The minimum No is 1

Pgm to find the Sum of n elements or numbers in the list

```
Sum = 0
list 1 = [ ]
n = int ( input ( " Enter the Range " )
for i  in range ( 0, n ) :
        a = int ( input ( " Enter a elements " ) )
        list 1 . append ( a )
for i in range ( 0 , len ( a ) ) :
    Sum = sum + a [ i ]
Print ( " Sum = " , Sum )
```

O/P Enter the Range 5

Enter a element ; 10

Enter a element : 20

Enter a element : 30

Enter a eleement : 40

Enter a erlement : 50

Sum =

Multidimensional List ;-

Syn : variable = [[ ], [ ]]

00 01 10 11

eg : a = [[1, 2] [3, 4]]

```
for i in range ( 0, len ( a ) ) :
    for j in range ( 0, len ( a ) :
        Print ( a [ i ] [ i ] )
```

O/P 1

2

3

4

Python pgm to add 2 Matrices :-

```
a = [ [ 2, 6 ], [ 7, 8 ] ]
b = [ [ 1, 3 ], [ 2, 4 ] ]
for i in range ( 0 , len ( a 0 0 :
    for j in range ( 0 , len ( a ) ) :
        c [ i ] [ j ] = a [ i ] [ j ] + b [ i ] [ j ]
for i in range ( 0, len ( a ) ) :
    for j in range (0, len ( a ) ) :
        Print ( c [ i ] [ j ] , end = )
Print ( 1 n )
```

O/P

3 9

9 12

Getting i/p values in Runtime :

```
a = [ [ ], [ ] ]
for i in range ( 0, n ) :
```

```
n = int ( input ( ) )
```

```
For j in range ( 0, n ) :
    a [ i ] [ j ] = int ( input ( ) )
```

Matrix Multiplication :-

```
a = [  [ 1, 1, 1 ] , [ 1, 1, 1 ] , [ 1, 1, 1 ]  ]

b = [  [ 1, 1, 1 ] , [ 1, 1, 1 ] , [ 1, 1, 1 ]  ]

c = [  [ 0, 0, 0 ] , [ 0, 0, 0 ] , [ 0, 0, 0 ]  ]

for i in range ( 0, len ( a ) ) :
    for j in range ( 0, len ( a ) ) :
        for k in range ( 0, len ( a ) ) :
            c [ i ] [ j ] = c [ i ] [ j ] + a [ i ] [ k ] + b [ k ] [ j ]

for i in range ( 0, len ( a ) ) :
    for j in range 9 o, len ( a ) ) :
        Print ( c [ i ] [ j ] , end = ‘ 1t ‘ )
Print ( “ In “ )
```


O/P

```
3   3   3
3   3   3
3   3   3
```

i/p 1 1

$$a = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \qquad b = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$