° Python Programming °

Data
Page

° Introduction - python is a widely used general-purpose, high level pl.
- Guido van Rossum in 1991. designed.

(1) Finding an Interpreter - Before we start python programming, we need to have an interpreter to interpret & run our programs.

° Running python in interactive mode -

- >>> print ("hello world")
hello world.

- [0, 1, 2]
>>> 2 + 3
5

° Data Types °

(1) Int - Int, ort integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
>>> print (246534)
246534
>>> print (20)
20
```

- to verify type
```
>>> type(10)
<class 'int'>
```

Date
Page

- **Float** - Float, or "floating point number" is no, (+ or -) containing one or more decimals.

```
>>> y= 2.8
>>> y
2.8

>>> print(type(y))
<class 'float'>
```

- **Boolean** - Objects of Boolean type may have one of two values, True or false.

```
>>> type(True)
<class 'bool'>
```

- **String** - strings is python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of the single or double quotes.

```
>>> print("mrcet college")
mercet college
>>> type("mrcet college")
<class 'str'>
```

- **Supperessing special Character** - Specifying a backslash (\) in front of the quote in a string "escapes" it and causes Python to suppress its usual special meaning.

```
>>> print("mrcet is an autonomous (\') college")
mrcet is can autonomum(') college
```

```
>>> print('Manoj (\'') Avni')
Manog (") Avni

>>> print('a\....b')
a...b

>> print(mrcet \n college")
mrcet
college
```

escape sequence

\' ⟶ terminates opening sigle quote delimiter.

\" ⟶ string with double quote opening delimiter

\newline ⟶ input line

\\ ⟶ escape sequence.

```
>>> print("a\tb")
a    b
```

- List ⟶

- it is a general purpose most widely used in data structures.
- list is a collectn which is ordered & changeble & allows duplicate members.
- we construct/create list in many ways.

Ex-

```
>>> list = [1, 2, 3, 'A', 'B', 7, 8, [10, 11]]
>>> print(list)
[1, 2, 3, 'A', 'B', 7, 8, [10, 11]]

>>> x = list()
>>> x
[]
```

```
>>> tuple = (1,2,3,4)
>>> x = list(tuple)
>>> x
[1,2,3,4]
```

- **Variables** - Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Ex-
```
a = 100
b = 1000.0
c = "John"
print(a)
print(b)
print(c)
```
Output - 100
1000.0
John

- **Multiple Assignment** - Python allows you to assign a single value to several variables simultaneously

Ex- `a=b=c=1`

- `a,b,c = 1,2,"mrcet"`

- **Output Variables** - The Python print statement is often used to output variables.

- Variables do not need to be declared with any particular type & can even change type after they have been set.

Ex -
```
x=5
x = "mrcet"
print(x)
```
Output : mrcet.

° Expressions °

An expression is a combination of values, variables and operators. An expression is evaluated using assignment operator.

Ex -
```
Y = X + 17
>>> X = 10
>>> Z = X + 20
>>> Z
30
```

° Identifiers - is used to define a class, function, variable, variable module, or object is an identifier.

° Literals - there are language-independent terms in Python & should exist independently in any programming language.

° Operators - implement the following operation

Operator	Token
add	+
subtract	-
multiply	*
Integer Division	/
remainder	%
Binary left shift	<<
Binary right shift	>>
add	&
or	|
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=

check equality ==
check not equal !=

- Some of the python expressions are:

- Generator expression:

```
>>> x = (i for i in 'abc')
>>> x
<generator object <genexpr> at 0x033EEC30>
```

- >>> x = "1" if True else "2"

```
>>> x
"1"
```

- Statements ← A statements is an instructn that the python interpreter can execute.

```
>>> x = 10
>>> college = "mrcet"
>>> print("mrcet colege")
mrcet college.
```

- Precedence of Operator - Operator precedence affects how an expression is evaluated.

```
>>> 3+4*2
11
```

- Comments -
A single-line comments beigns with a hash (#) symbol & is useful is mentioning that the whole line should be considered as a comment

until end of line.

- A Multi line comment is useful when we need to comment on many lines.

Ex- # Write a python program to add no.
""" print the value """
''' print the value '''

- Modules - python module can be defined as a python program file which contains a python code including python functions, class, or variables.

```
>>> import sys
>>> print(sys.version)
3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23)
>>> print(sys.version_info)
sys.version_info(major=3, minor=8, micro=0, releaselevel
  ='final', serial=0)
>>> print(calendar.month(2021,5))

>>> print(calender.isleap(2020))
True.
>>> print(calender.isleap(2017))
False
```

Data
Page

- **Functions** :- Function is a group of related statements that perform a special task. Functions help break our program into smaller & modular chunks. As our program grows larger & larger, functions make it more organized & manageble.

1. **Built-in functions** - functions that are built into python.

Ex - abs(), all(), ascii(), boo() ..... so on....

```
integer = -20
print('Absolute value of -20 is:', abs(integer))
```

Output - Absolute value of -20 is: 20

2. **User-defined functions** - functions defined by the users themselves.

```
def add numbers(x, y);
    sum = x + y
    return sum
print("The sum is ", add-numbers(5, 20))
```

Output - The sum is 25

- **Flow of execution** -

1. the order in which statements are executed is called flow of execution.
2. Execution always beigns at the first statement of the program.
3. Statements are executed one at a time, from top to bottom.

Ex-

```
# example for Flow of execution
print(" Welcome ")
for x in range(3);
print(x)
print ("Good morning college")
```

Output :

```
C:Users/MRCET/AppData/Local/Programs/Python/Python38-32/pyyy/Flowof.py
Welcome
0
1
2
Good morning college.
```

Ex-

example for Flow of execution with functions.

```
def hello():
    print(" Good Morning")
    print(" mrcet ")
print (" hi")
print (" Hello ")
hello ()
print( " done ")
```

Output - hi
Hello
Good morning
mrcet
done !

The flow/order of execution is : 2, 5, 6, 7, 2, 3, 4, 7, 8.

- Parameters and arguments - parameters are passed during the defination of function while Arguments are passed during the function call.

Ex -
```
# here a and b are parameters
def add(a,b):
    return a+b
result = add(12,13)
print(result)
```
Output - 25

- Types of Python -

1. Default Arguments
2. Keyword Arguments
3. Variable-length Arguments.

Ex -
```
def hf():
print("hello world")
hf()
```
Output - hello world

-
```
def hello_f():
    return "hellocollege"
print(hello_f().upper())
```
Output - HELLOCOLLEGE

- Passing Arguments.

```
def hello(wish)
  return '{}'.format(wish)
print(hello("mrcet"))
```
Output :- mrcet.

- **Keyword Arguments** - some functions with many parameters & you want to specify only some of them, then you can give values for such parameters by naming them - this is called keyword arguments.
  - python allows functions to be called using keyword arguments.

Ex -
```
def func(a, b=5, c=10):
    print 'a is', a, 'and b is', b, 'and c is', c
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```
Output - a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50.

- **Default Arguments** - function arguments can have default values in python.

Ex -
```
def hello(wish, name='you'):
    return '{}, {}'.format(wish, name)
print(hello("good morning"))
```
Output - good morning, you

- **Variable-length arguments** -
Sometimes you may need more arguments to process function then you mentioned in the definatn. If we don't know in advance about the arguments needed in functn, we can use variable-length arguments also called arbitary arguments.

Ex-

```
def wish (*names):
""" This function greets all
    the person in the names tuple. """
for name in names:
 print("Hello", name)
wish("MRCET", "CSE", "SIR", "MADAM")
```

Output - Hello MRCET
Hello CSE
Hello SIR
Hello MADAM

program to Find biggest of two numbers using functn.

```
def biggest(a,b):
 if a>b:
  return a
 else:
  return b

a = int(input("Enter a value"))
b = int(input("Enter b value"))
big = biggest(a,b)
print("big number =", big)
```

Output - Enter a value 5
Enter b value -2
big number = 5

Write a program to display mrecet cse dept 5 times on the screen.

```
def usingFunctions():
    count = 0
    while count <10:
        print("mrcet cse dept", count)
        count = count +1
usingfunctions()
```

Output - mrcet cse dept 0
mrcet cse dept 1
mrcet cse dept 2
mrcet cse dept 3
mrcet cse dept 4

- CONTROL FLOW, LOOPS

Conditionals - Booleans values and operators, conditional (if), alternative (if -else), chained conditional (if-else-else);

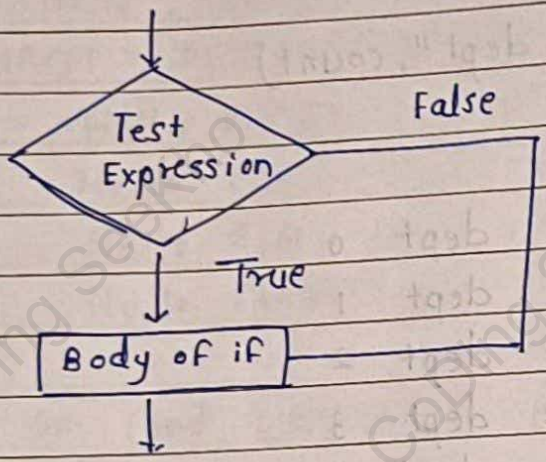
- Iteration : while, for, break, continue.

- A boolean expressions is an expression that is either true or false.

```
>>> 5 == 5
True

>>> type(True)
<class 'bool'>
```

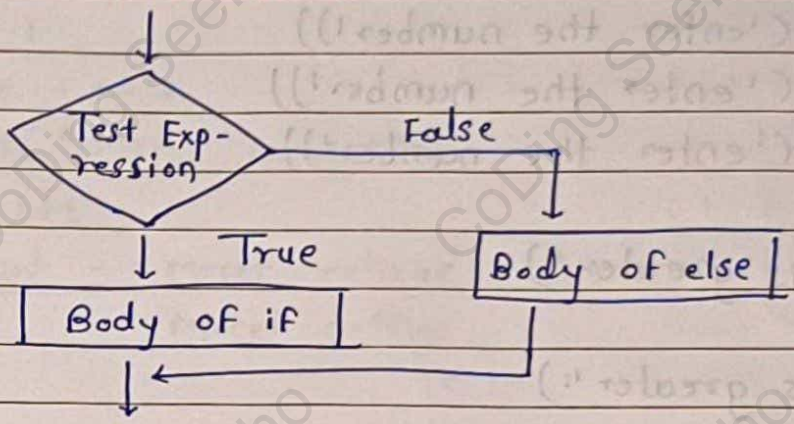
- Conditional (if) :- the if statement a logical expression using which data is compared and a decision is made based on the result of the comparison. is Statement flowchart:



Ex-
```
a = 3
if  a >2 :
print (a," is greater ")
print("done ")
a = -1
if a <0 :
  print(a," a is smaller")
print(" Finish ")
```
Output - A is Greater than 9

- Alternative if (if-Else) - An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

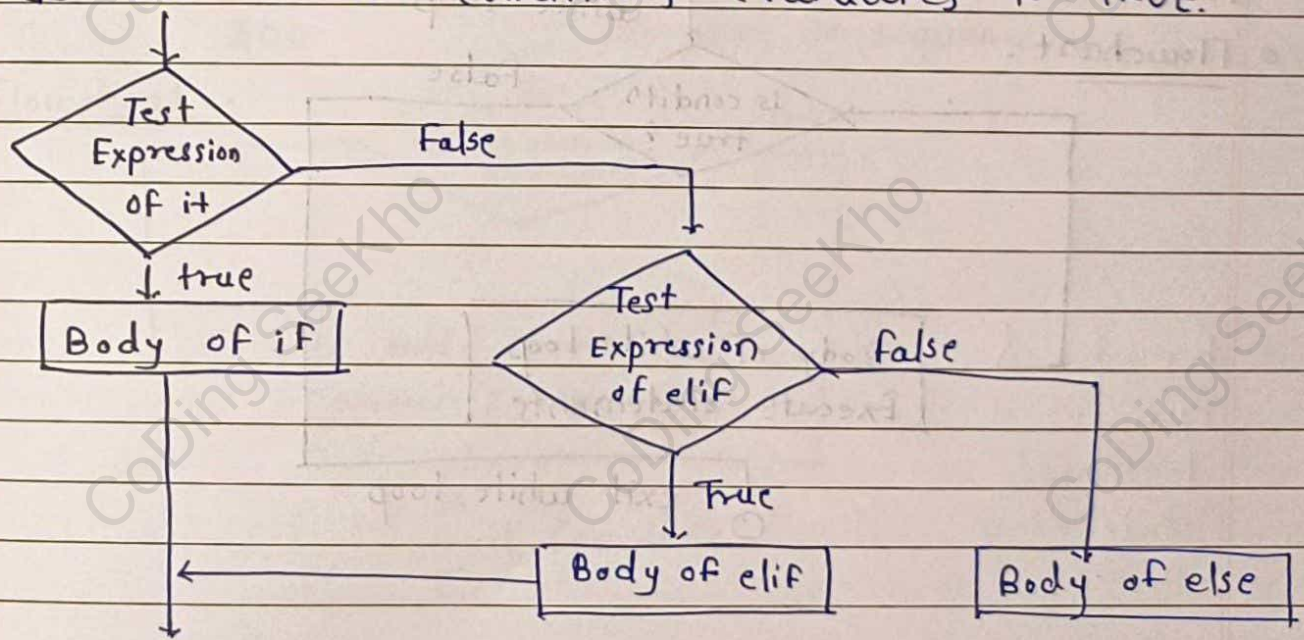
If - else Flowchart :



Ex -
```
a = int (input('enter the number '))
if a>5:
  print("a is greater ")
else :
  print(" a is smaller than the given input")
```
Output - enter the number 2
a is smaller than the input given

- Chained Conditional : (if-elif-else)

the elif statement allows us to check multiple expressi-ons for TRUE & execute a block of code as soon as one of the conditions evaluates to TRUE.

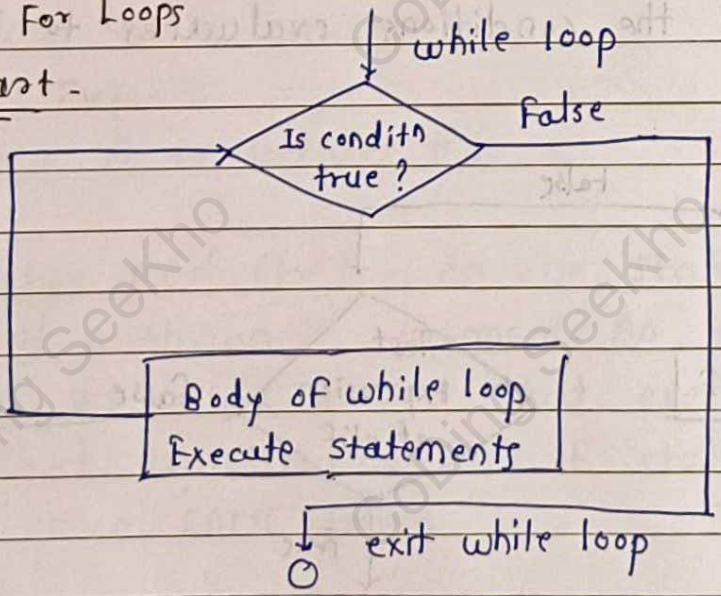


Ex-
```
a = int(input('enter the number'))
b = int(input('enter the number'))
c = int(input('enter the number'))
if a>b:
  print("a is greater")
elif b>c:
  print("b is greater")
else:
  print("c is greater")
```
Output - enter the number 5
enter the number 2
enter the number 9
a is greater

- Iteration :- A loop statement allows us to execute a statement or group of statements multiple time as long as the condition is true.

1. While Loop
2. For Loop
3. Nested For Loops

- Flowchart -

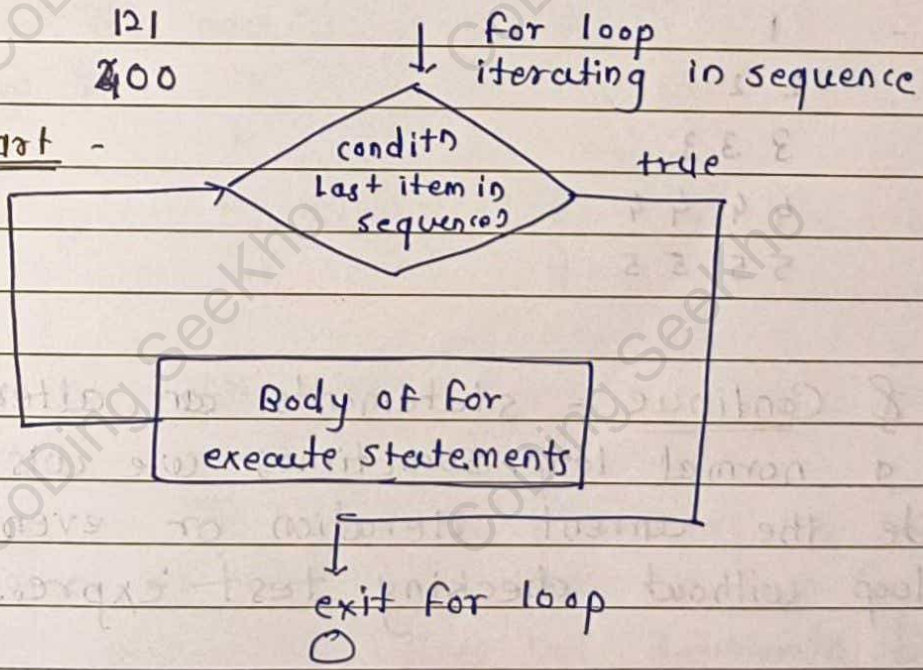


Ex -
```
i = 1
while i <= 2
print(" Mrcet college ")
 i = i+1
```
Output - Mrcet college
Mrcet college

- For Loop - python for loop is used for repeated execution of a group of statements for the desired no of times.

Ex -
```
numbers = [ 1, 2, 4, 6, 11, 20]
Seq = 0
for val in numbers :
  seq = val * val
  print (seq)
```
Output - 1
4
16
36
121
400

- Flowchart -



- Iterating over a list -

```
list = ['M', 'R', 'C', 'E', 'T']
i = 1
for item in list:
print ('college.'i', is', item)
  i = i+1
```

Output - college 1 is M
college 2 is R
college 3 is C
college 4 is E
college 5 is T

- Nested For Loop - When one Loop defined within another Loop is called Nested Loops.

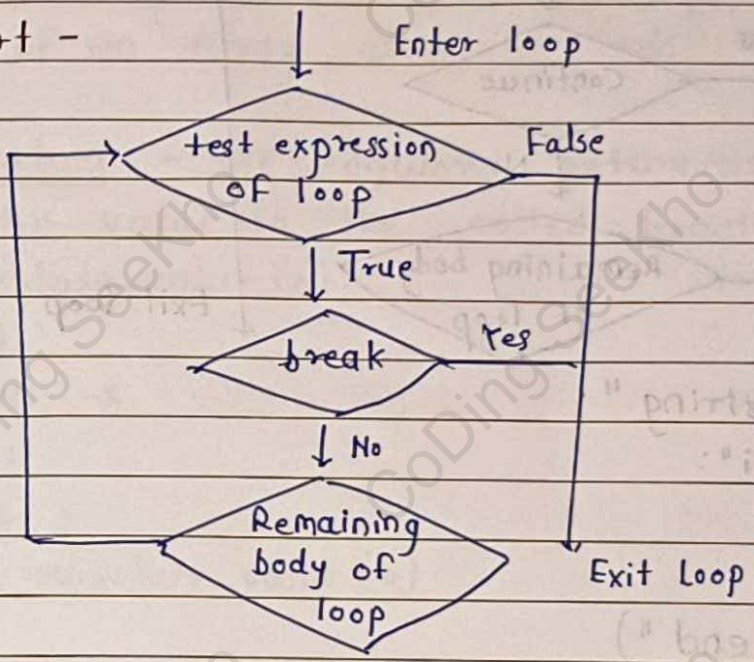
Ex -
```
For i in range(1,6):
  for j in range( 0, i):
    print(i, end= " ")
print(")
```

Output -
1
2 2
3 3 3
4 4 4 4
5 5 5 5

- Break & Continue - statements can alter the flow of a normal loop. Sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

- **Break** - the break statement terimates the loop containing it and control of the program flows to the statement immediately after the body of the loop.

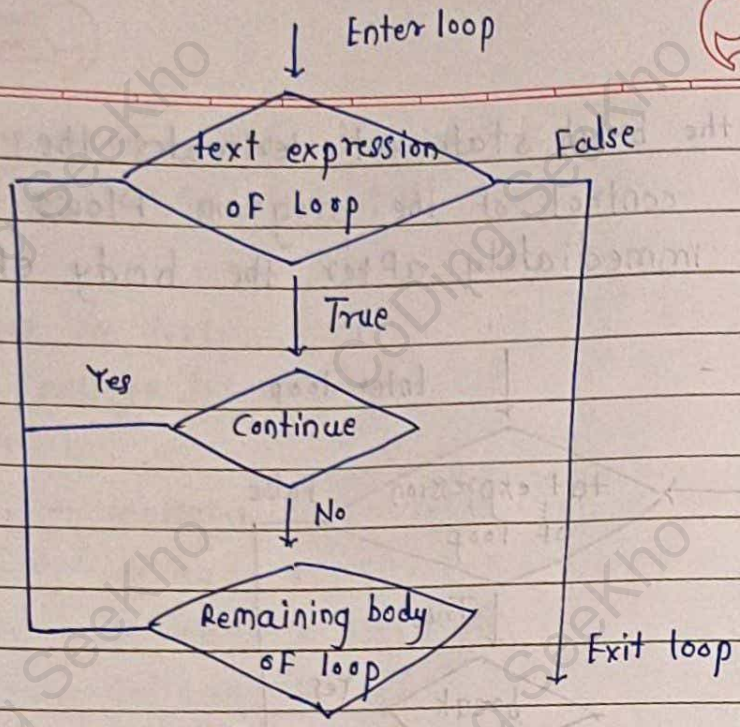
- Flowchart -



Ex-
```
For val in "MRCET COLLEGE":
if val == " ":
    break
print(val)
print("The end")
```

Output :-
```
M
R
C
E
T
The end.
```

- **Continue** - The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with next itereration.



Ex-
```
for val in "string":
    if val == "i":
        continue
    print(val)
print("The end")
```

Output -
```
s
t
r
n
g
The end
```

- Pass - the difference betn a comment and pass statement in python is that, while the interpreter ignores a comment entirely, pass is not ignored.

Functions, Arrays

- **Fruitful Functions** - return values, parameters, local and global scope, function composition recursion; Strings: string slices, immutability, string functions & methods, string module; Python arrays, Access the Elements of an Array, array methods.

- **Return values** - the keyword return is used to return back the value to the called function.

Ex-
```
def absolute_value(x):
    if x < 0:
        return -x
    if x > 0:
        return x
>>> print absolute_value(0)
None
```

- **Parameters** - parameters are passed during the definatn of functn while Arguments are passed during the function call.

Ex-
```
def add(a,b):
    return a+b
result = add(12,13)
print(result)
```
Output :- 25

- No parameters & no return type

```
def Fun1()
    print(" function 1")
Fun1()
```
Output : function 1.

- with param with out return type

```
def Fun2(a):
    print(a)
Fun2("hello")
```

Output : Hello

- without param with return type

```
def fun3():
    return "Welcome to python"
print(fun3())
```

Output - welcome to python

- with parm with return type

```
def fun4(a):
    return a
print(fun4("python is better then c"))
```

Output - python is better then c

- Local Scope - a variable is defined inside a function a local to that function.

- Global Scope - a variable which is defined in the main body of a file is called global variable.

Ex -

```
x = "global"
def f():
    print("x inside :", x)
f()
print("x outside :", x)
```

Output - x inside : global
x outside : global.

- local variable -

Ex-
```
def f1():
    y = "local"
    print(y)
f1()
```
Output :- local

- Function Composition :- Having two (or more) functions where the output of one function is the input the for another.

Ex-
```
>> def d(x):
    return x * 2
>> a(5)
>>> a
```
Output - 10

- Recursion - Recursion is the process of defining something in terms of itself.

- Recursion function -

Ex-
```
def fact(x):
 if x == 0:
    result = 1
 else:
    result = x * fact(x-1)
 return result
print("zero factorial", fact(0))
print("five factorial", fact(5))
```
Output - Zero Factorial 1
five Factorial 120

- **Strings** - a string is a group a sequence of characters. Since python has no provision for arrays, we simply use strings. This is how we declare a string. We can use a pair of single or double quotes.

```
>>> type("name")
<class 'str'>
>>> name = str()
>>> name
''
```

- **String slice** - A segment of a string is called a slice.

Ex-
```
str = 'Hello World!'
print str
print str[0]
print str[2:5]
print str[2:]
str * 2
print str + "TEST"
```

Output -
```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

- **Immutability** - it is tempting to use the [] operator on the left side of an assignment, with the intention of changing a character in a string.

Ex:
```
>>> greeting = "Hello, World!"
>>> new_greeting = 'J' + greeting[1:]
>>> new_greeting
'Jello, world!"
```

- String functions & methods -

- isalnum()
- isalpha()
- isdigit()
- islower()
- isnumeric()
- isspace()
- swapcase()
- startwith()
- istitle()
- isupper()
- replace
- Split()
- count()
- find()

① isalnum() - Isalnum() method returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

Ex - >>string = "123alpha"
>>> string.isalnum() True

② isalpha() - isalpha() method returns true if string has at least 1 character and all charcters are alphabeti -c and false otherwise.

Ex -
```
>>> String = "nikhil"
>>> string.isalpha()
True.
```

3. isdigit() - isdigit() return true if string contains only digits and false otherwise.

Ex - >>> string = "123456789"
>>> string.isdigit()
True.

4. islower() - Islower() returns true if string has characters that are in lowercase and false otherwise

Ex - >>> string = "nikhil"
>>> string.islower()
True

5. isnumeric() - string contain only numeric characters & false otherwise

Ex - >>> string = "123456789"
>>> string.isnumeric()
True.

6. isspace() - string contains only white space between & false otherwise.

Ex - >>> string = " "
>>> string.isspace()
True

7. istitle() - returns true if string is properly "titlecased" starting letter of each word is capital) & false otherwise.

Ex - >>> string = "Nikhil Is Learning"
>>> string.istitle()

8. isupper() - returns true if string has charchters that are in uppercase & false otherwise.

Ex-
```
>>> string = "HELLO"
>>> string.isupper()
True.
```

9. replace() - replaces all occurrences of old in string with new or at most max occurrences if max given.

Ex-
```
>>> string = "Nikhil Is Learning"
>>> string.replace('Nikhil', 'Neha')
'Neha Is Learning'
```

10. split() - splits the string according to delimiter str.

Ex-
```
>>> string = "Nikhil Is Learning"
>>> string.split()
['Nikhil', 'Is', 'Learning']
```

11. count() - counts the occurrence of a string in another string.

Ex-
```
>>> string = 'Nikhil Is Learning'
>>> string.count('i')
3
```

12. find() - is used for finding the index of the index of the first occurrence of a string in another string

Ex-
```
>>> string = "Nikhil Is Learning"
>>> string.find('k')
2
```

13. swapcase() - converts lowercase letters in a string to uppercase & viceversa.

Ex-
```
>>>string = "HELLO"
>>>string.swapcase()
'hello'
```

14. startswith() - determines if string or a substring of string starts with substring str; returns true if so & false otherwise.

Ex-
```
>>>string = "Nikhil"
>>>string.startswith('Ni')
True
```

15. endswith() - determines if string or a substring of string ends with substring str; returns true if so & false otherwise.

Ex-
```
>>> string = "Nikhil Learning"
>>> string.startswith('g')
True
```

- String Module - this module contains a no of functions to process standard python strings. In recent version, most functn are available as string methods as well.

- Python arrays - arrays is a container which can hold a fix no of items & these items should be of the same type. Most of the data structures make use of arrays to implements their algorithms.

- Element- each item stored in an array is called an element.
- Index - each locatn of an element in an array has a numerical index, which is used to identify the element

Elements →

Int array [10] = {10,20,30,40}

↓ ↓ ↓ ↓

Type Name Size Index 0

- Basic Operations -
- Traverse - print all the array elements one by one
- Insertion - Adds an element at the given index.
- Delection - Deletes an element at the given index.
- Search - Searches an element.
- Update - Updates an element at the given index

Typecode		Value
b	→	singed integer of size 1 byte /td>
B	→	unsigned integer of size 1 byte
c	→	character of size 1 byte
i	→	singned integer of size 2 bytes
l	→	unsigned integer of size 2 bytes
f	→	Floating point of size 4 bytes
d	→	Floating point of size 8 bytes

- Creating an array -

```
from array import *
array1 = array('i', [10,20,30,40,50])
for x in array1:
print(x)
```

Output - 10
20
30
40
50

- Access the elements of an Array.

```
from array import *
array1 = array('i',[10,20,30,40,50])
print(array[0])
print(array[2])
```

Output - 10
30

- Array methods - python has a set of built-in methods that you can use on lists/arrays.

Method	Description
append()	→ adds an element at the end of list
clear()	→ removes all the elements from list
copy()	→ returns a copy of the list
count()	→ returns the no of elements with the specified value
extend()	→ add the elements of a list, to the end of the current list
index()	→ index of the first element with the specified value
insert()	→ adds an element at the specified positn.
(pip) →	→ removes the element at the specified position.

remove() → removes the first item with the specified value

reverse() → reverses the order of the list

sort() → sorts the list

Ex-

```
>>> college = ["mrcet", "it", "cse"]
>>> college.append("autonomous")
>>> college
['mrcet', 'it', 'cse', 'autonomous']

>>> college.append("eee")
>>> college.append("ece")
>>> college
['mrcet', 'it', 'cse', 'autonomous', 'eee', 'ece']

>>> college.pop()
'ece'
>>> college
['mrcet', 'it', 'cse', 'autonomous']

>>> college.remove("it")
>>> college
['mrcet', 'cse', 'autonomous']
```

LISTS, TUPLES, DICTIONARIES

- **Lists** - list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters, list comprehension

- it is a general purpose most widely used in data structures
- List is a collection which is ordered & changeable & allows duplicate members.

Ex-
```
>>> list = [1,2,3,'A','B',7,8,[10,11]]
>>>print(list)
[1,2,3,'A','B',7,8,[10,11]]
```

- **List Operations** - Lists respond to the + & * operators much like strings; they mean concentration & repetition here too, except that the result is a new list, not a string

- List slices:-

Ex-
```
>>> list = range(1,6)
>>> list
rang(1,6)
```

- List methods-
  - Del()
  - Append()
  - Extend()
  - Insert()
  - Pop()
  - Remove()
  - Reverse()
  - Sort()

- **Delete** - Delete a list or an item from a list

```
>>> x = [5, 3, 8, 6]
>>> del(x[1])
>>> x
[5, 8, 6]
```

- **Append** - item to a list

```
>>> x = [1, 5, 8, 4]
>>> x.append(10)
>>> x
[1, 5, 8, 4, 10]
```

- **Extend** - append sequence to a list

```
>>> x = [1, 2, 3, 4]
>>> y = [3, 6, 9, 1]
>>> x.extend(y)
>>> x
[1, 2, 3, 4, 3, 6, 9, 1]
```

- **Insert** - to add an item at the specified index, use the insert() method.

```
>>> x = [1, 2, 4, 6, 7]
>>> x.insert(2, 10)
>>> x
[1, 2, 10, 4, 6, 7]
```

- **Pop** - method removes the specified index, or simply pops the last item of list & returns the item.

```
>>> x = [1, 2, 10, 4, 6, 7]
>>> x.pop()
7
```

Data
Page

- **Reverse** - reverse the order of a given list.

```
>>> x = [1, 2, 3, 4, 5, 6, 7]
>>> x.reverse()
>>> x
[7, 6, 5, 4, 3, 2, 1]
```

- **List Loop** :- Loops are control structures used to repeat a given section of code a certain number of times or until a particular condition is met.

Ex -
```
list = ['M', 'R', 'C', 'E', 'T']
i = 1
for item int list:
  print('college', 'i', 'is', item)
  i = i+1
```

Output -
```
college 1 is M
college 2 is R
college 3 is C
college 4 is E
college 5 is T
```

- **Mutability** - A mutable object can be changed after it is created, and an immutable object can't

Append - Append an item to a list

```
>>> x = [1, 5, 8, 4]
>>> x.append(10)
>>> x
[1, 5, 8, 4, 10]
```

- **Aliasing** -

1. An alias is a second name for a piece of data, often easier than making a copy.
2. If the data is immutable, alies don't matter because the data can't change.

Ex-
```
a = [81, 82, 83]
b = [81, 82, 83]
print(a == b)
print(a is b)
b = a
print(a == b)
print(a is b)
b[0] = 5
print(a)
```

Output -
```
True
False
True
True
[5, 82, 83]
```

- **Cloning Lists** - If we want to modify a list & also keep a copy the original, we need to make a copy of the list itself, not just the reference. This process is sometimes called cloning, to avoid the ambiguity of the word copy.

Ex-
```
a = [81, 82, 83]
b = a[:]
print(a == b)
print(a is b)
b[0] = 5
print(a)
print(b)
```

Output - True
False
[81, 82, 83]
[5, 82, 83]

- List parameters - passing a list as an argument acutally passes a reference to the list, not a copy of the list. list are mutable, changes made to the elements referenced by the parameter change the same list that the argument is referencing

Ex -
```
def double(List):
    """Overwrite each element in a List with double
       its value."""
    for position in range(len(List)):
        List[position] = 2 * List[position]
things = [2, 5, 9]
print(things)
double(things)
print(things)
```
Output - [2, 5, 9]
[4, 10, 18]

- List comprehension - list comprehensions provide a concise to create lists. Common applicatn are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of these elements that satisfy a certain condition.

Ex-
```
>>> list = []
>>> for x in range(10):
        list.append(x **2)
>>> list
[0,1,4,9,16,25,36,49,64,81]
```

- Tuples - a tuple is a collectn which is ordered & unchangeable.
- supports all operations for sequences.
- immutable, but member object may be mutable

Ex-
```
>>> x = (1,2,3)
>>> print(x)
(1,2,3)
```

- Access tuple items - a referring to the index no, inside square brackets

```
>>> x = ('a','b','c','g')
>>> print(x[2])
c
```

- Change tuple items - tuple is created, you cannot its values. Tuples are unchangeble.

Ex-
```
>>> x = 4,5,6,7,2,'aa'
>>> for i in x:
      print(i)
```
Output - 4
5
6
7
2
aa

- Count() - returns the no of times a specified value occurs in a tuple

```
x = (1,2,3,4,5)
x.count(2)
4
```

- Index() - Searches the tuple for a specified value & returns the posith of where it was found.

```
x = (1,2,3,4)
x.index(2)
1
```

- Length() - no of items or values present in a tuple, we use len()

```
x = (1,2,3,4,5)
y = len(x)
print(y)
4
```

- Tuple Assignment - python has tuple assignment feature which enables you to assign more than one variable at a time.

Ex -
```
tup1 = ('mrcet', 'eng college', '2004', 'cse', 'it', 'csit');
tup2 = (1,2,3,4,5,6,7)
print(tup1[0])
mrcet
```

- Tuple as return values - a tuple is a comma separated sequence of item. It is created with or without (). Tuples are immutable.

Ex-
```
def fun():
    str = "mrcet college"
    x = 20
    return str, x;
str, x = fun()
print(str)
print(x)
```
Output - mrcet college
20

- **Dictionries** - a dictionary is a colled which is unordered, changeable & indexed. In python dictionaries are written with curly brackets, & they have keys & values.

Ex- `x = {1:'A', 'B', 3:'c'}`
`x = dict(['a', 3)('b', 4)]`
`x = dict('A'=1, 'B'=2)`

- **Operations & methods** -

method	description
clear()	→ remove all items from the dictionary.
copy()	→ return a shallow copy of the dictionary
get(key[,d])	→ return the value of key. If key doesn ot exit, return d
items()	→ return a view new of the dictiorary's items.
keys()	→ remove new view of the dictionary's keys.
pop(key[,d])	→ keyError.
popitem()	→ raises KeyError if the dictionary is empty.

setdefault(key[,d]) → if not, insert key with a value of d & return d

update → overwriting existing keys

values() → return a new view of the dictionary's values.

- Below are some dictonary operations-

- To access specific value of a dictionary, we must pass its key,

```
dict = {"brand": mrcet, "model": college, year: 2004}
x = dict["brand"]
x
'mrcet'
```

- Add/change values - You can change the value of a specific item by referring to its key name.

```
dict = {"brand":"mrcet", "model":"college"; "year": 2004}
dict["year"] = 2005
dict
{"brand": 'mrcet', 'model': 'college', 'year': 2005}
```

- Comprehension - Dictionary comprehension can be used to create dictionaries from arbitary key & value.

```
z = {x: x**2 for x in (2,4,6)}
z
{2:4, 4:16, 6:36}
```

o Files and exception - A file is some information or data which stays in the computer storage devices.

Write a python program to open & read a file

```
a = open("one.txt", "r")
print(a.read())
a.close()
```

Welcome to python programming

```
f = open("1.txt", "a")
f.write("hello world")
f.close()
```

Output :- hello world

o Command line arguments - the command line arguments must be given whenever we want to give the input befor the start of the script, while on the other hand, raw_input() is used to get the input while the python program / script is running.

- 'sys' module - python sys module stores the command line arguments into a list, we can access it using sys.argv.

Ex -

```
import sys
argumentList = sys.argv
print(argumentList)
print(sys.argv[0])
print(sys.argv[1])
```

- 'getopt' module - python argparse module is the preferred way to parse command line argument.

Ex-
```
import getopt
import sys
argv = sys.argv[0]
try:
  opts, args = getopt.getopt(argv, 'hm:d", ['help', myfile
  print(args)
```
Output - error

- Errors and Exceptions - Python error & Built-in expectn : python raises exceptions when it encounters errors.

- ZeroDivisionError - indicates that the second argument
- OverflowError - arithmetic operatn has exceeded the limits of current python runtime.
- ImportError - raised when you try to import dey not exist.
- IndexError - sequence which is out range.
- TypeError - two unrelated type are combined
- Indention Error - Unexpected indent
- Syntax errors - they arise when the pytho parser is unable to understand a line of code.
- Run-time error - A run-time error happens when python understands what you are saying, but runs into trouble when following your instructn
- key error -
- value error - a value is the informatn that stored within a certain object.
- python has many built-in excepth which forces your program to output an error when something goes wrong.

Date
Page

- Different types of exceptions -
  - ArrayIndexOutOfBoundException
  - ClassNotFoundException
  - FileNotFoundException
  - IOException
  - InterruptedException
  - NoSuchFieldException
  - NoSuchMethodException

- Handling Exception - the cause of an exception is often external to the program itself.
  - Syntactical error
  - Logical error
  - Run time error

Ex -
```
a = 5
b = 2
print(a/b)
print("Bye")
```
Output - 2.5
Bye

- the except block executes only when try block has an error, check it below

```
a = 5
b = 2
try:
    print(a/b)
except Exception:
    print("number can not be divided by zero")
    print("bye")
```

Output - 2.5

- a = 5
  b = 0
  try:
  print(a/b)
  except Exception as e:
  print("no can not be divided by zero", e)
  print ("bye")
  Output - number can not be divided by zero
  division by zero
  bye

- a = 5
  b = 0
  try:
  print("resource opened")
  print(a/b)
  except Exception as e:
  print("no can not be divided by zero", e)
  print ("resource closed)
  Output - resource opened
  2.5

- But again the same problem file/resource is not closed.
- To overcome this python has a feature called Finally

```
a=5
b=0
try:
 print ("resource open")
  print( a/b)
   k = int(input("enter a no"))
    print(k)
exceptZeroDivisionError as e:
    print("the value can not divided by zero",e)
Finally:
    print("resource closed")
```

Output - resource open
the value can not be divided by zero division by zero
resource closed

- Modules (Date, Time, os, calender, math)
- modules refers to a file containing python statements definations.
- Modular programming refers to the process of breaking a large, unwidely programming task into separate, smaller, more, manggeable subtasks or modules

Ex -

```
def add(a,b):
 result = a+b
   return result
""" The program adds two no and reter the
        result""" """
```

Date Page

- Important module -

- Reloading a module -

```
def hi(a,b)
    print(a+b)
hi(4,4)
```

Output - 8

```
>>> import add
```

- >>> import imp
- >>> import my_module

- >>> example.__name__
'example'

- Datetime module -

- Write a python program to display date, time

```
>>> import datetime
>>> a = datetime.datetime(2019,5,27,6,35,40)
>>> a
datetime.datetime(2019,5,27,6,35,40)
```

- write a python program to display date

```
import datetime
a = datetime.date(2000,9,18)
print(a)
```

Output - 2000-09-18

- write a python program to print date, time for today & now.

```
import datetime
a = datetime.datetime.today()
b = datetime.datetime.now()
print(a)
print(b)
```

Output - 2019-11-29 12:49:52.235581

- write python program to print the no. of days to write to reach your birthday

```
import datetime
a = datetime.date.today()
b = datetime.date(2020,5,27)
c = b-a
print(c)
```

Output - 180 days, 0:00:00

o Time module -

write a python program to get structure of time stamp

```
import time
print(time.localtime(time.time()))
```

Output -

- os module -

```
>>> import os
>>> os.name
'nt'
```

- `>> os.access("t1.py", os.W_OK)`
  True

- `>> os.access("t3.py", os.F_OK)`
  False

- Calendar module -

Write a python program to display a particular month of a year using calender module

```
import calendar
print(calendar.month(2020, 1))
```

Output - Calendar

Write a python program to check wheather the given year is leap or not

```
import calendar
print(calendar.isleap(2021))
```

Output - False

MATH MODULE -

write a python program which accepts the radius of a circle user & computes the area.

```
import math
r = int(input("Enter radius: "))
area = math.pi * r*2
print("Area of circle is", area)
```

Output - Enter radius: 4
Area of circle is : 50.26548245743669

import with renaming -

- we can import a module by renaming it as follows.

e.g - >>> import math as m.
>>> print("The value of pi is", m.pi)

O/P - the value of pi is 3.141592653589793

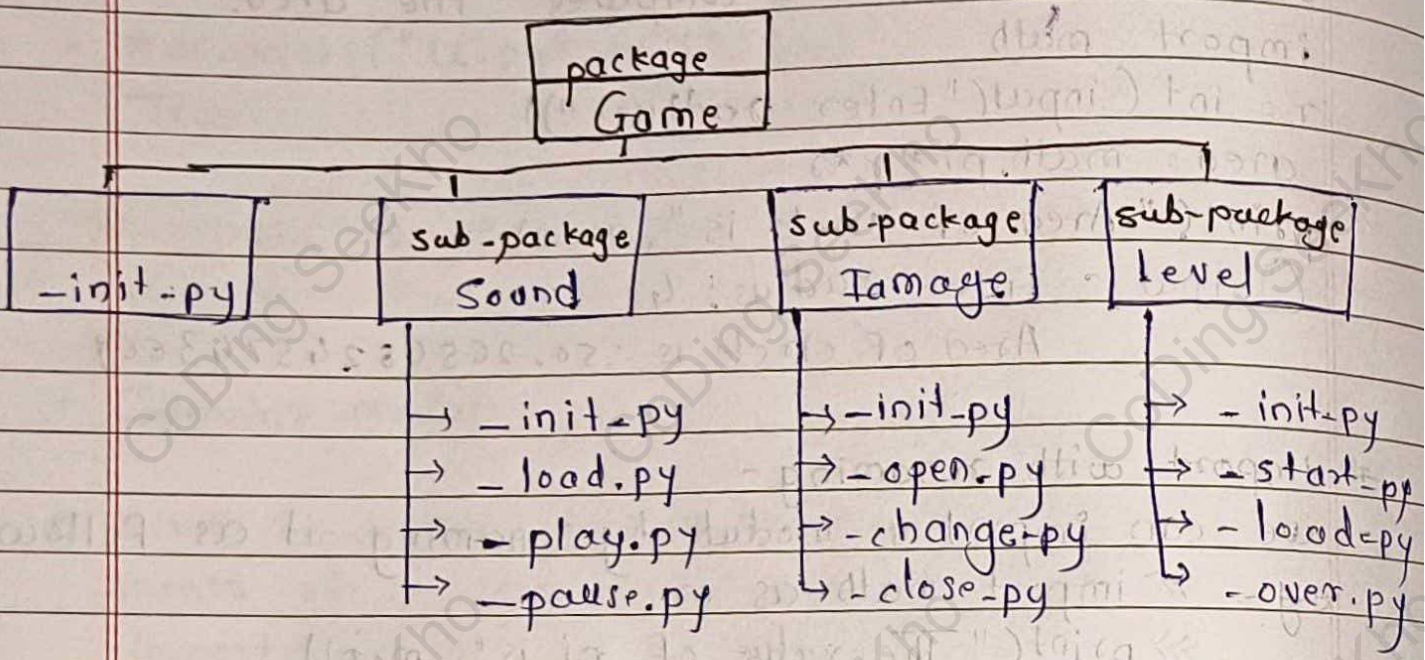
python From...import statement -

- we can import specific names from a module without importing the module as a whole.

Ex >>> from math import pi, e
>>> pi
3.141592653589793

∘ Explore Packages ∘

- We don't usually store all of our files in our computer in the same locatn. We use a well-organized hierachy of directories for easier access.



- A module in the package can access the global by importing it in turn

∘ start.select_difficulty()
Yet another way of importing just the required functn from a module within a package would be as follows

Ex -
```
def read():
    print("Department")
def write():
    print("student")
```

- >>> student.write()
  Student

Write a program to create and rename the existing module.

- def a():
    print("hello world")
a()

Output - hello world

- import exam as ex
ex.a() Output - hello world.