MODULE-1

1] - Entering expressions in the interactive shell :-

→ Run the interactive shell by launching IDLE which is installed with python.

→ On windows :-

1. Open start menu
2. select All Programs.
3. select Python 3.3
4. Then select IDLE (Python GUI).

→ On OS x,

1. select Applications.
2. select Mac Python 3.3
3. select IDLE.

→ In Ubuntu

1. Open Terminal
2. enter >>> idle3.

A prompt will appear. That is the interactive shell.

→ The most basic kind of programming instruction in Python is called an expression.

→ An expression consists of values and operators. and they can always evaluate down to a single value. eg :- >>> 2 + 2
4.

→ A single value can also be called an expression.

2]- Salient features of python.

→ ① Easy to code.

⇒ Python is a high-level programming language.

⇒ Hence it is very easy to learn as compared to C, C++

⇒ It is also a developer friendly language.

② Object - oriented language

⇒ Supports object oriented programming and concept of classes and encapsulation.

③ GUI programming support

⇒ GUI can be made using PyQt5, PyQt4, or Tk.

⇒ PyQt5 is most popular for creating graphical apps with python.

④ High-level language

⇒ we dont need to remember the system architecture, nor do we need to manage the memory.

⑤ Extensible

⇒ we can write a python code into C or C++ and also compile it in C/C++.

⑥ Portable :-

⇒ Code once, Run anywhere

⇒ same program can be run on any platform.

(7) It is an Integrated language

⇒ can be integrated with C, C++

(8) Interpreted language

⇒ code is executed line by line at a time.

⇒ There is no need to compile python code.

⇒ This makes it easier to debug the code.

⇒ The source code of python is converted into an immediate form called byte code.

(9) Large - standard library.

⇒ Python has a large standard library that provides a set of modules & functions.

⇒ so, we don't have to write our own code for every single thing.

⇒

(10) Dynamically Typed language.

⇒ The type of variable (int, float, etc) is decided at run time and not in advance.

⇒ so we don't need to specify the type of variable.

3]- Math operators :-

Operator	Operation	Eg.	Evaluates to
* *	Exponent	2 * * 3	8
%	Modulus / remainder	10 % 5	0
//	Integer division	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	10 - 2	8
+	Addition	2 + 2	4

→ the order of operations occurs according to the precendence.

→ * * is evaluated first, (% // / *) are evaluated next and then + and -.

→ To override the usual precendence, we can use ( ) ⇒ parantheses.

eg :- Normal evaluation

```
>>> 2 + 3 * 6
20
```

Evaluation with paranthesis

```
>>>(5 -1) *((7 + 1) / (3 -1))
```

↓

4 * (8 / 2)

↓

4 * 4.0

>>> 16.0

4]. Data types – int, float, string.

①

Data type	Keyword	Examples
Integer	int	-2, -1, 0, 1, 2, 3
Floating -point	float	2.56, -1.2, 0.5, etc.
Strings	str	'a', 'aaaa', 'fo'.

→ Integers are whole numbers. eg. 2, 5678, 99.

→ Numbers with a decimal point are floating point numbers. eg. 2.0, 5.99, 6278.0.

→ Strings are set of characters enclosed within ' '. eg:- 'hello'

* String replication ⇒ '*' [can only be used with 1 string & 1 number].

* String concatenation ⇒ '+'

eg:->>> 'Alice' + 'Bob'

'AliceBob'

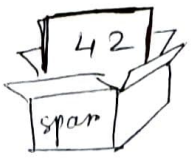
>>> 'Alice' * 5

'AliceAliceAliceAliceAlice'.

* Assignment operator

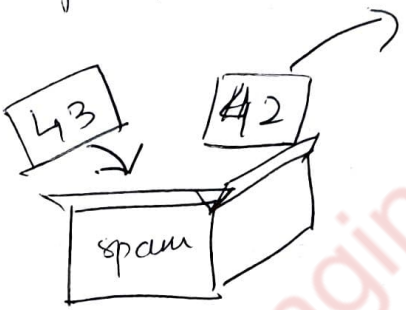
⇒ To asign a value to a variable name

⇒ eg. spam = 40.



⇒ If a variable is assigned a new value after the previous assignment, the old value is forgotten.

eg:-




```
eg:- >>> spam = 'Hello'
     >>> spam
     'Hello'
     >>> spam = 'Goodbye'
     >>> spam
     'Good bye'
```

* Rules for variable names

① It can be only 1 word

② can use only letters, numbers & underscore

③ Can't begin with a number.

④ they are case sensitive.

⑤ A good variable describes the data it contains.

5). Basic functions used in Python

① print()

→ prints / displays the string value inside parantheses

eg:- >>>print ('Hello world!')
Hello world!

→ when python executes this line, it is calling the print function and the string value is being passed to the functions.

→ The value passed to the function is called an argument.

② input()

→ waits for user to type some text and press ENTER.

→ Evaluates to a string equal to the user's text.

eg:- myName = input().

⇒ Here if the user types in 'Anushka' then the expression would evaluate to

myName = 'Anushka'.

*To print the value entered:-

>>> print ('It is good to meet you,' + myName)

* It is good to meet you, Anushka.

③ len()

→ To find number of characters in the string.

eg:-

```
>>> len('hello')
5
>>> len('Hi, my name is Bob')
18
```

→ counts the spaces as characters as well.

④ str(), int(), float()

→ evaluates to string version of a number.

eg:-

```
>>> str(29)
'29'
>>> print('I am' + str(20) + 'years old')
I am 20 years old.
```

→ similarly int() & float() evaluate to the integer and floating point value of the number.

```
>>> spam = input()
101
>>> spam
'101'
```

(input() function always returns a string, even if user enters a number)

Hence, we write it as

```
>>> spam = int(spam)
>>> spam
101
```

* Program to find Area of square, rectangle & circle.

~~print ('Enter two~~

```
# Area of rectangle
print ('Enter length:')
l = float (input())
print ('Enter breadth:')
b = float (input())
a = l * b
print ('Area=' + a)
```

```
# Area of square
print ('Enter side length')
s = float (input())
a = s * s
print ('Area = ' + a)
```

```
# Area of circle
print ('Enter radius')
r = float (input())
a = 3.14 * r * r
print ('Area = ' + a)
```

6]- Boolean and comparison operators:-

* ~~Boolean opers~~ Comparison Operators

→ Compare 2 values and evaluate down to a single value.

Operator	Meaning
= =	equal to
! =	not equal to
<	less than
>	greater than
> =	greater than equal to
< =	less than equal to.

→ These operators evaluate to true or false depending on the values we give them.

eg:-

```
>>> 42 = = 42
True
>>> 42 > 58
False.
>>> 2 ! = 3
True.
```

→ The = = & ! = work with any data type.

```
>>> 'hello' = = 'hello'
True
>>> 'dog' ! = 'cat'
True.
>>> 42 = = '42'
False.
```

* Boolean operators :-

→ Three boolean operators → and, or, not

① and :-

Expression	evaluates to
True and True	True
True and False	False
False and True	False
False and False	False

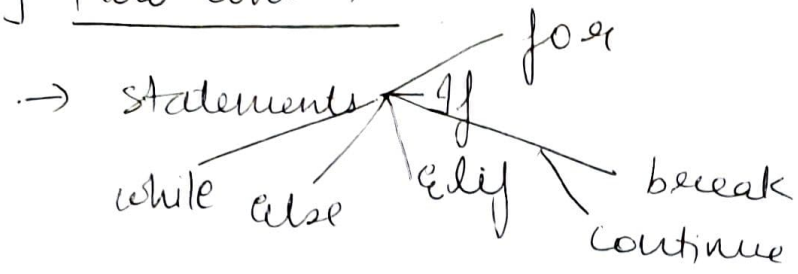
② or :-

Expression	evaluates to
True or True	True
True or False	True
False or True	True
False or False	False

③ not :-

Expression	evaluates to
not True	False
not False	True

4]- Flow control



① If :-

→ The statement clause will execute if condition is True.

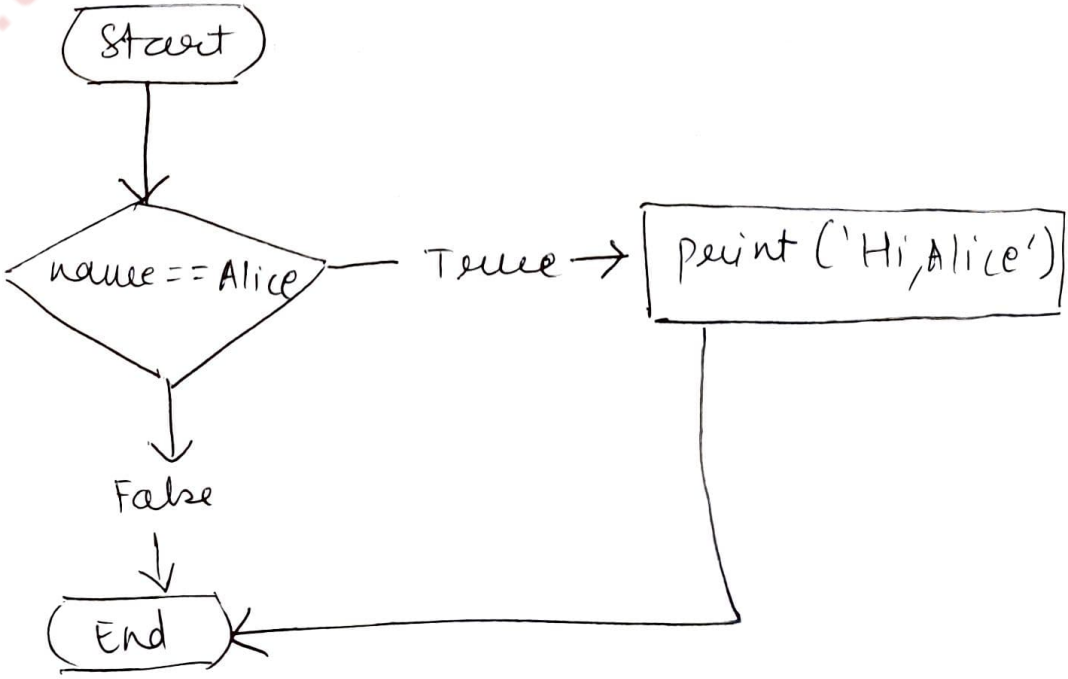
→ otherwise clause is skipped.

→ contains :-

⇒ if keyword
⇒ condition
⇒ colon (:)
⇒ if clause.

eg :- if name == 'Alice':
print ('Hi, Alice.').

=Flow chart.



② else :-

→ 'if' can optionally be followed by else

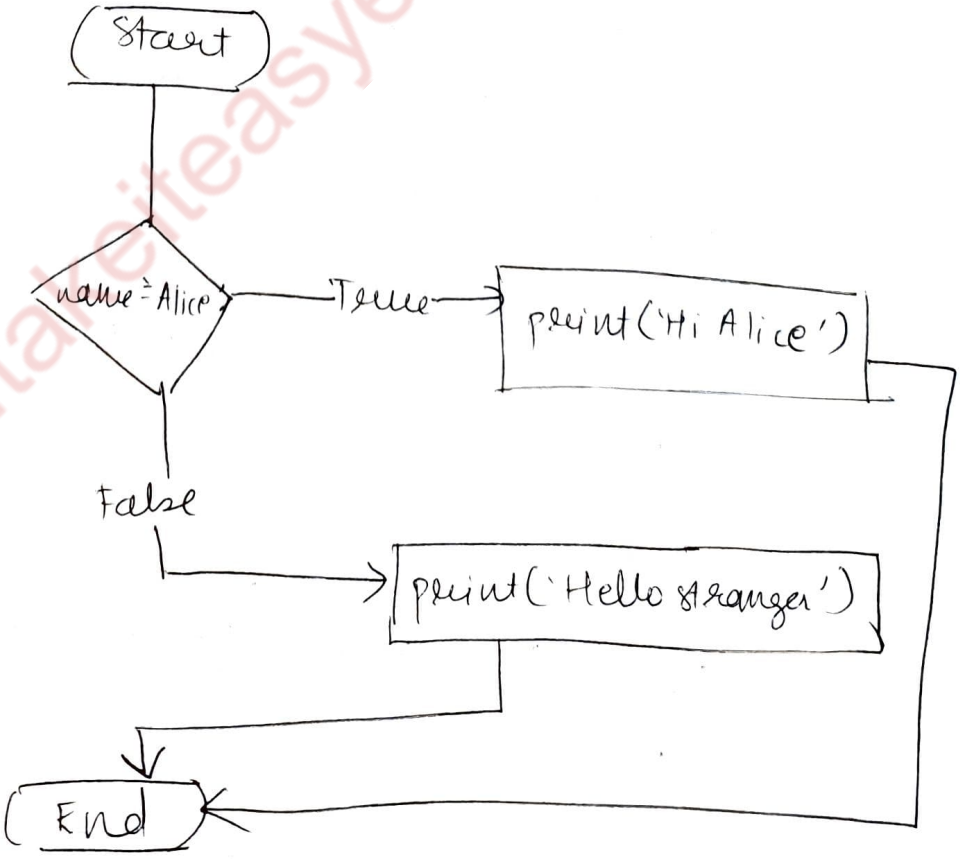
→ 'else' is executed when the if statement's condition is false.

→ contains

- the else keyword
- colon
- indented block of code (clause)

eg:- if name == 'Alice':
print ('Hi Alice!')
else:
print ('Hello, stranger')

⇒ Flow chart :-



③ elif :-

→ when we need more than one else conditions and clauses, elif is used.

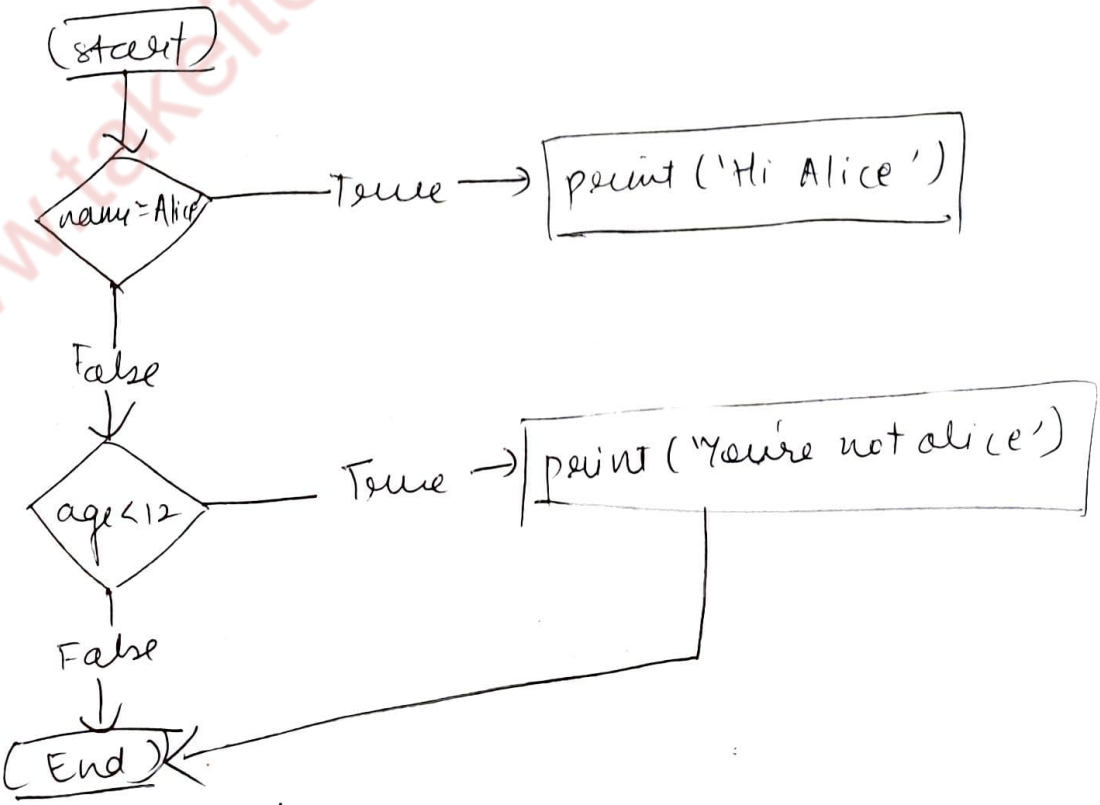
→ It is an "else if" statement

→ It provides another condition that is checked, only if the previous conditions were False.

→ contains :-

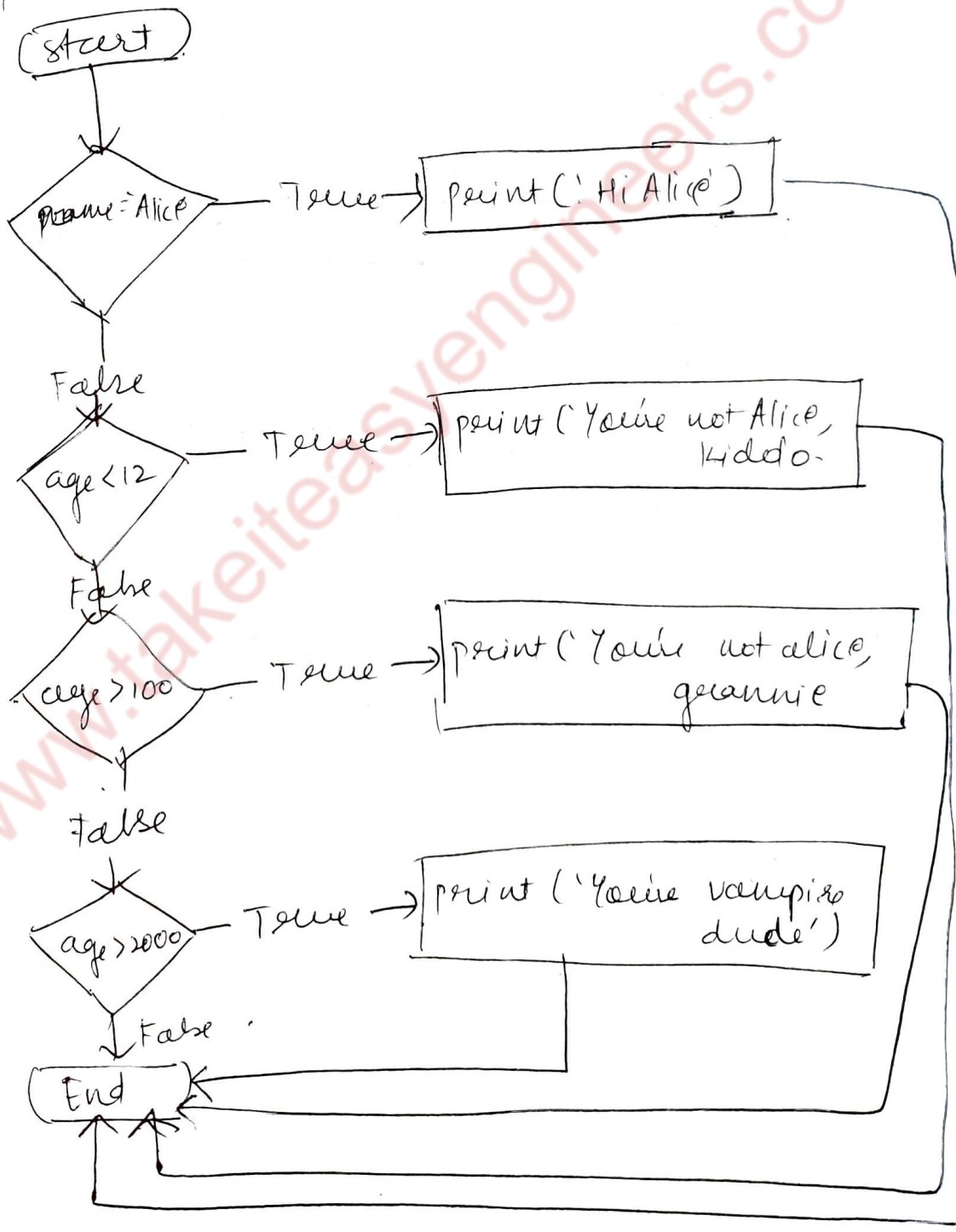
- The elif keyword
- condition
- A colon
- elif clause.

eg :-
```
if name == 'Alice':
    print ('Hi, Alice.')
elif age < 12:
    print ('You are not alice')
```



→ There can be a chain of elif statements :-

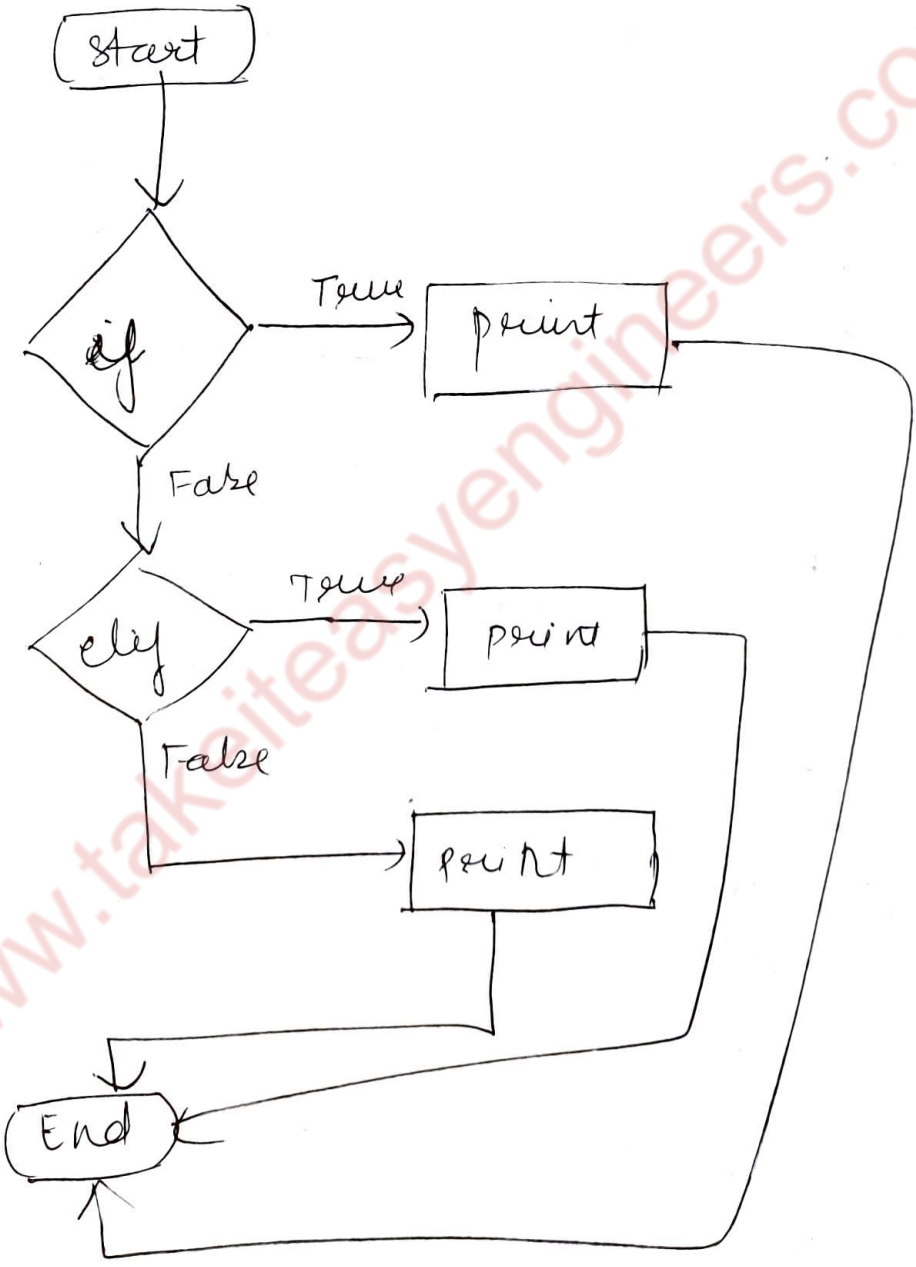
```
if name = 'Alice':
print('Hi Alice')
elif age < 12:
print('Youre not alice, kiddo')
elif age > 100:
print('Youre not alice, grannie')
elif age > 2000:
print('Youre a vampire, dude');
```



(44) → we can also have an else statement after elif if we want a guaranteed execution of ~~else~~ at least one clause.

→ else is executed if every elif statement is false.

Flow chart :-



④ while

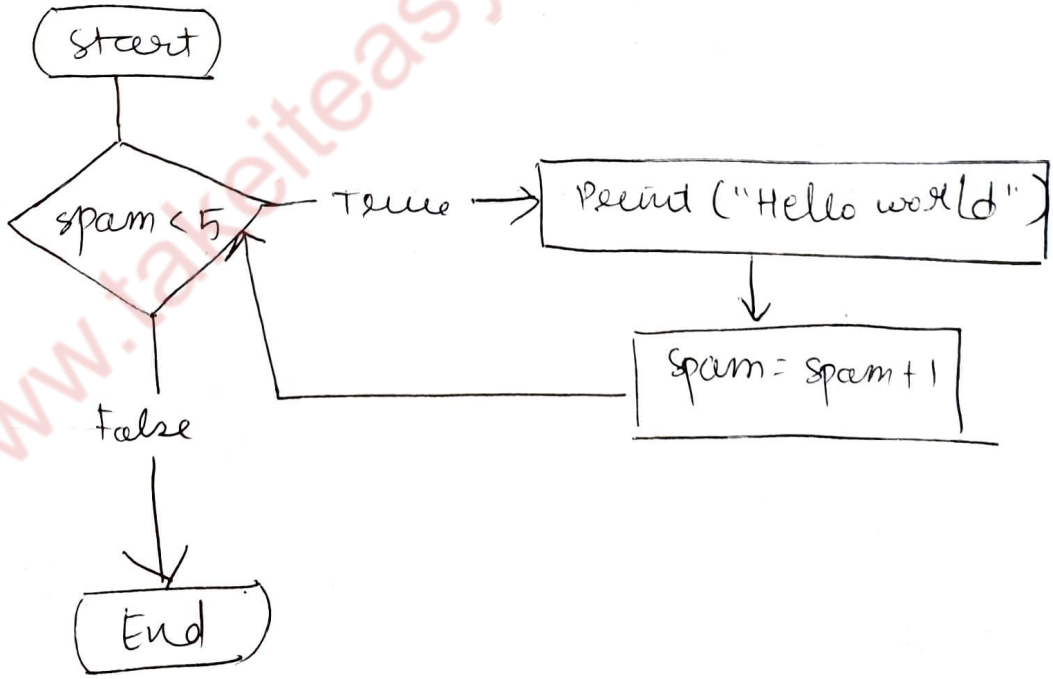
→ used to make a block of code execute over and over again

→ clause is executed as long as the while statement's condition is True!

→ contains :-

① while keyword
- condition
- colon
- clause.

eg :- spam = 0
while spam =><5 :
print ("Hello world").
spam = spam + 1



→ the clause is executed until the result of while condition is false.

→ The condition is checked at the start of each iteration.

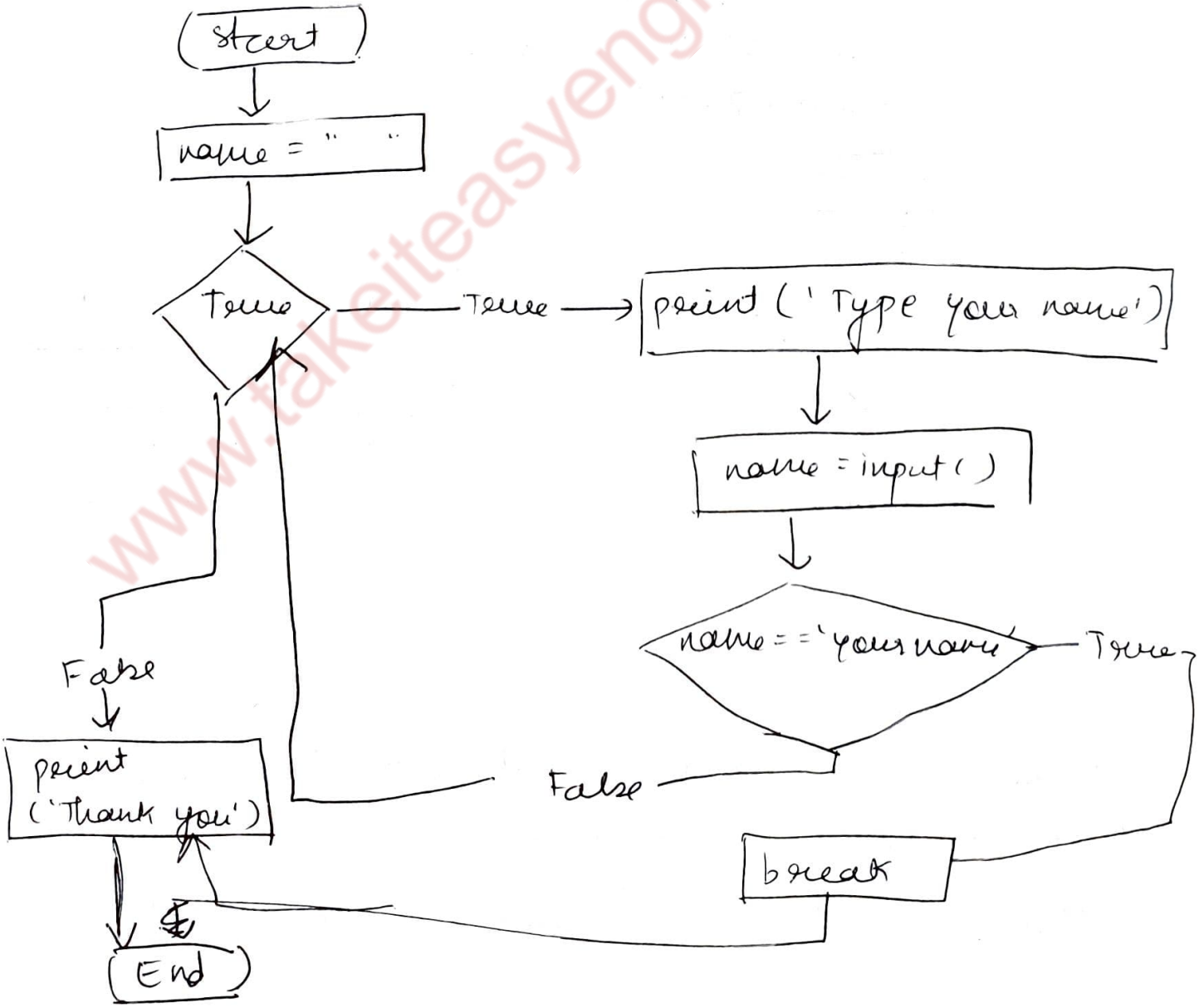
⑤ break:

→ It is a shortcut to break out of a while loop.

→ If execution reaches break statement, it immediately exits the while loop.

→ It just contains the break keyword.

Eg:-
```
while True:
    print ('Type your name')
    name = input()
    if name == 'your name':
        break
print('Thank you')
```

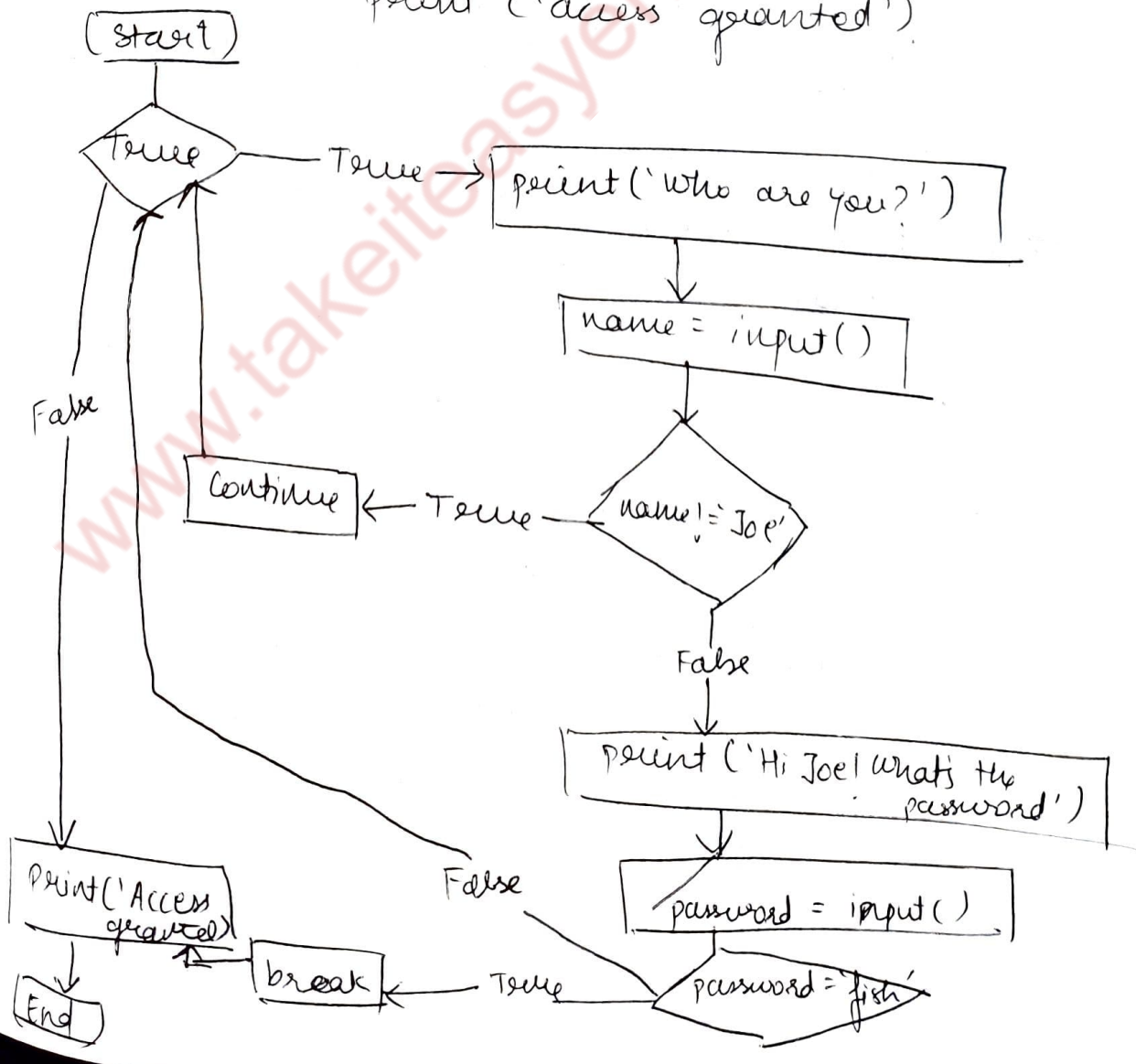


⑥ b continue :

→ when a program execution reaches a continue statement, it immediately jumps back to the start of the loop & re-evaluates the loop's condition.

→ eg :-

```
while True:
    print ('Who are you')
    name = input()
    if name != 'Joe':
        continue
    print ('Hi joe! whats the password')
    password = input()
    if password = 'fish':
        break
    print ('access granted')
```



⑦ for loop and the range() function

→ when we want to execute a block of code only a certain number of times, we use for loop.

→ It contains

- for keyword
- ~~to~~ A variable name (i, j, k, etc)
- The 'in' keyword.
- ~~The~~ A call to range() where upto 3 integers can be passed
- A colon
- clause.

→ eg:-

```
print ('My name is')
for i in range (5):
    print ('Jimmy Five times (' + str(i) + ')')
```

→ This code runs 5 times

→ The first time, i is set to 0. Then it keeps iterating until i = 4

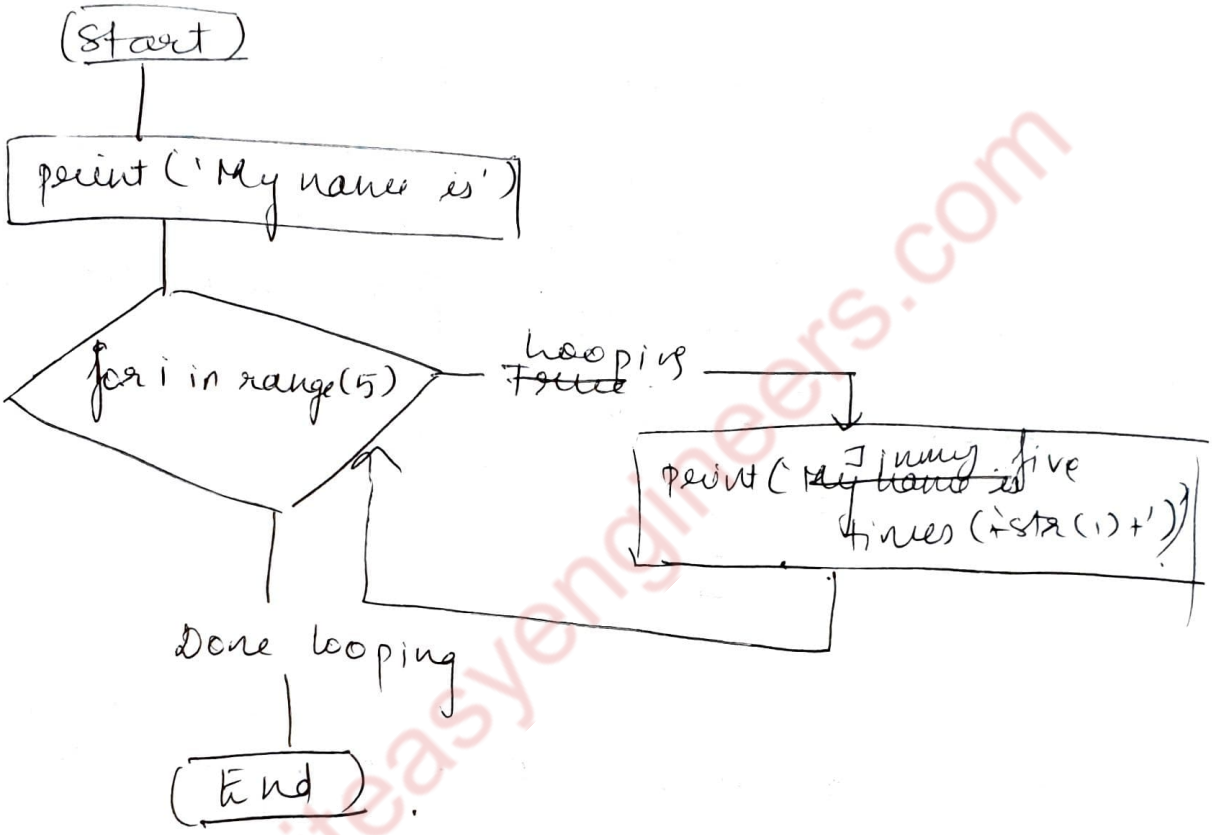
output:-

```
My name is
Jimmy Five times (0)
Jimmy Five times (1)
Jimmy Five times (2)
Jimmy Five times (3)
Jimmy Five times (4)
```

→ After every iteration, the execution goes back to the top, and the for statement increments i by one.

→ The variable i will go up to but not include the integer passed to range().

→ Flow chart:



→ In for ⇒ we can also add a start, stop & step argument for the range() function.

→ eg:- for i in range (12, 16):
print (i)

Here 12 is the start of range & 16 is the end of range

⇒ so loop will satisfy when the iteration number is b/w 12 & 16.

⇒ But, the ~~argum~~ loop starts with '12' and executes upto 16 but won't include 16.

⇒ Hence only o/p will be ⇒ 12, 13, 14, 15.

→ range() can also be called with 3 arguments.

eg: for i in range(0, 10, 2):
    print(i)

Here ⇒ • 1st two arguments will be start and stop i.e start with 0 and count upto 10 but don't include 10.

• The last argument is the step.

The step is the amount that the variable is increased by after each iteration.

⇒ Hence in the above eg:-

o/p ⇒ 0
2
4
6
8

⇒ In steps of 2, i.e, skipping 2 variables after every iteration.

8] - Importing modules:-

import random() ⇒ imports the random() module from standard library to generate a random integer.

similarly there are different modules such as sys, os, math.

* To exit a program anywhere before the actual ending of execution we use

sys.exit()

9] - Functions

→ A function is like a mini-program within a program.

→ We can write anything inside it and the functions that are defined by the user are called user defined functions.

→ They can be defined as:-

```
def name()
```

→ Here, def keyword is used to say that this is a user defined function

→ The function name is then kept using () following the name.

→ eg:-

```
① def hello():
      ② print('Yo')
        print('Halo frands')
        print('Annyeong')

③ hello()
  hello()
  hello()
```

→ The first line ① is the def statement that defines a function named hello().

→ The code block ② is the body of the function. Whenever the function hello() is called, this block will be executed.

→ The hello() lines ③ are the function calls. These will call the hello() function to get executed.

→ When the execution reaches these calls, it will jump to the top line in the function & begen executing the code there.

→ After completion, the program continues moving throughout the code as before.

∴ o/p of eg :-

```
# Yo
  Halo frands
  Annyeong
  Yo
  Halo frands
  Annyeong
  Yo
  Halo frands
  Annyeong.
```

→ The major purpose of a function is to group code that we would want to execute multiple times.

→ Instead of repeating the whole code again and again, we just define it inside a function once and call it everytime we want it to execute or be used.

* Functions can also be defined using arguments inside the ( ).

eg:- ① def hello(name):

② print ('Annyeong' + name)

③ hello('Alice')
hello('Bob').

→ The definition of the hello() function here has a parameter name.

→ A parameter is a variable where we can store ~~the~~ an argument when a function is called i.e, we can put any name in place of that parameter wherever we want to call that function.

→ Hence the o/p will be

Annyeong Alice
Annyeong Bob.

→ when the function hello('Alice') is called, automatically the name variable is set to 'Alice' while execution of the hello(name) function.

→ Also, the value stored in a parameter is forgotten when the function returns.

~~to A program local~~

* Return values and return statements.

→ The value that a function call evaluates to (The result of the evaluation) is called as return value of the function.

→ we can specify a return value within a def statement.

→ It consists:

- return keyword
- The value / ~~expression~~ ~~function~~ that the function should return.

eg:- ① import random.

② def getAnswer(answerNumber):

③ if answerNumber == 1:
return 'Hello ji'
elif answerNumber == 2:
return 'Annyeonghaseo'
elif answerNumber == 3:
return 'Namastey'
elif answerNumber == 4:
return 'Moshi Moshi'

④ r = random.randint(1, 5)

⑤ greeting = ~~answerName(r)~~ getAnswer(r).

⑥. print(greeting).

① ⇒ we import the random module to generate a random number

② ⇒ a function def getAnswer( ) is defined. It is just defined and not called as the execution initially skips this block of code.

③ ④ ⇒ A variable r arms a function random.randint(1, 5) to generate any random integer between 1 to 5.

⑤ ⇒ The getAnswer() function is called and the value of r is passed as an argument.

⑥ ⇒ The execution now enters the getAnswer function and starts finding for the possible string value to return according to the random integer value r.

⑦ ⑤ ⇒ The program execution then returns to the line at the bottom that equals to getAnswer()

⑥ ⇒ The obtained string (return value) is assigned to the variable greeting and printed on screen.

we can also shorten the last lines as

print (getAnswer (random.randint (1, 5)))

10]- Local and Global scope:-

Local

→ parameters & variables assigned inside a called function, are said to exist in the function's local scope

→ A variable in local scope is called local variable.

→ A local scope is created when a function is called

→ When a function returns, the local scope is destroyed and the variables are forgotten.

Global

→ parameters and variables assigned outside all functions are said to exist in the global scope.

→ A variable in global scope is called global variable.

→ A global scope is created when a program begins.

→ A global scope is destroyed when the whole program terminates / ends and all variables are forgotten.

* A variable must either be one of these two, it cannot be both.

* some rules of global & local scope:-

① local variables cannot be used in Global scope

② local scopes cannot use variables in other local scopes.

③ local scope can access a global variable

④ Same name can be used for variables if they are in different scopes. i.e, there can be a local variable named spam and also a global variable named spam.

11]- The Global Statement -

→ If we want to modify a global variable from within a function, we use the global statement.

→ For eg:-

If we have a line that says - global eggs This tells Python - "In this function, 'eggs' is a global variable, so don't create a local variable with this name."

eg:- def spam():
global eggs
eggs = 'spam'

→ We cannot simply change the value of a global variable. A special permission is needed to do so. This permission is obtained using the Global statement..

def

12] - Exception Handling :-

→ Exception Handling is used when we want a program to detect errors and handle them to continue running rather than crashing the entire program

→ Hence, errors can be handled by ~~exceptio~~ try and except statements.

→ The code that could potentially cause an error is put in the try clause.

→ The execution moves to the start of the except clause if an error happens.

→ eg :- The ZeroDivision error is common error that occurs when we to try to divide a number by zero.

eg :- so we write it as

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print ('Error')

print (spam(2))
print (spam(12))
print (spam(0))
```

o/p ⇒ 21.0
3.5
Error.