***PYTHON SEE EXAM NOTES***

***Module 1 chapter1***

**List The Salient Features Of Python Programming Language**

Ans:1. Easy to learn and read: Python has a simple and readable syntax, making it easy for beginners to understand and write code.

2. Interpreted language: Python is an interpreted language, which means that the code is executed line by line, making it easier to debug and test.

3. Cross-platform compatibility: Python is available for various operating systems, including Windows, macOS, and Linux, making it highly portable.

4. Extensive library support: Python has a vast collection of libraries and modules that provide ready-to-use functions and tools for various tasks, such as data analysis, web development, and machine learning.

5. Object-oriented programming (OOP) support: Python supports OOP principles, allowing developers to create reusable and modular code.

6. Dynamic typing: Python is dynamically typed, which means that variable types are determined at runtime. This allows for flexible and easier code development.

7. Strong community and ecosystem: Python has a large and active community of developers who contribute to its growth and provide support through forums, tutorials, and online resources.

8. Integration capabilities: Python can easily integrate with other languages, such as C, C++, and Java, making it a versatile language for building complex applications.

9. Scalability: Python is known for its scalability, allowing developers to build small scripts or large-scale applications with ease.

Open-source: Python is an open-source language, which means that it is free to use and modify, making it accessible to everyone.

***02 explain operators and types; Operators:***

The operators are identifiers or tokens which are used to perform some specific operation.

Ex=a++,a+b,a%b,a\*b a<b.

***Types of Operators:***

1. ***Airthmetic operations:***

These are the operators which are used to perform some arithmetic operations on operand. Such as addition ,subtraction ,multiplication and division etc… .Since it operates on two operand they are also called as binary operations.

Ex=a+b, a/b, a\*c etc.

***Presidence of operators:***

The order in which an operators are evaluated in an given expression is called as

presidence of operators.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***SL NO*** | ***OPERATORS*** | ***SIGN/ASSOCIATIVITY*** | ***PRESIDENCE*** | ***EXAMPLE*** |
| 1 | MULTIPLICAION | \* LEFT(L)-  >RIGHT(R) | 01 | 2\*3=6 |
| **2** | DIVISION | / L->R | 01 | 4/2=2 |
| **3** | MODULUS | % L->R | 01 | 4%2=0 |
| **4** | ADDITION | + L->R | 02 | 4+2=6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | SUBTRACTION | - L->R | 02 | 4-2=2 |

Example:2+3\*(3-2)+2/2 //brackets are evaluated first

2+3\*1+2/2 //division is done

2+3\*1+1 //multiplication is done

2+3+1 // left to right associative

5+1 //addition is done

6

1. ***Relational or comparison operators:***

The operators which are used to find relationship and compare the values of two operands are called relational or comparison operators.

If the result is correct then 1 or True .else 0 or False. Ex=>, <, >=,<=,!=

***Presidence of relational operators***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***SL***  ***NO*** | *OPERATORS* | *SIGN/ASSOCIATIVITY* | *PRESIDENCE* | *EXAMPLE* |
| 1 | Greater than | > LEFT(L)->RIGHT(R) | 01 | a>b |
| **2** | Lesser than | < L->R | 01 | a<b |
| **3** | Greater than equal to or less than or equal to | >=,<= L->R | 01 | a>=b |
| **4** | Less than or equal to | = L->R | 02 | a<=b |
| **5** | Not equal to | != L->R | 02 | a!=b |

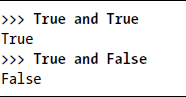
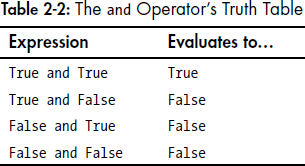
.

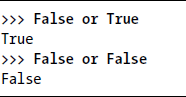
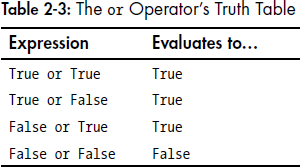
1. ***Bollean Oprerators***
   * + The three Boolean operators (and, or, and not) are used to compare Boolean values.

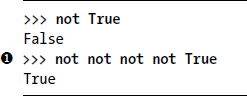
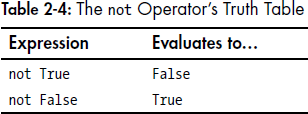
#### Binary Boolean Operators

* + - The and and or operators always take two Boolean values (or expressions), so they‘re considered binary Operators.

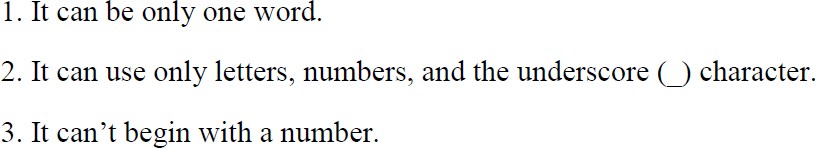
***and operator:*** The and operator evaluates an expression to True if both Boolean values are True; otherwise, it evaluates to False.



***or operator:*** The or operator valuates an expression to True if either of the two Boolean values is True. If both are False, it evaluates to False.

***not operator:*** The not operator operates on only one Boolean value (or expression). The not operator simply evaluates to the opposite Boolean value. Much like using double negatives in speech and writing, you can nest not operators ❶, though there‘s never not no reason to do this in real programs.

***Variables:***

The variables are the names given to the programming elements memory location .

***1.4 Storing Values in Variables***

* A variable is like a box in the computer‘s memory where you can store a single value.
* If we need to use variables later, then the result must be stored in variable.

***Data types:***

The expressions are just values combined with operators,and they are evaluated down to a single expression.

Data types is a category for a values, and every value belong to exactly one data .Data types is a classification of input data for easy processing

1. ***int:***

The int is a data type, which is used to hold only an integer valued (whole number)data.it usually holds 2 or 4 bits of memory. ex=int a, 10,20,30

1. ***float:***

The float is a data type, which is used to hold only a floating point numbers (fractional or decimal number)data.it usually holds 2 or 4 bits of memory. ex=float a ,1.5 3/6

1. ***String:***

The string is “Any thing which is enclosed with in a pair of single or double quotes “ ex=str”abc”.

***Assignment Statements***

* You‘ll store values in variables with an assignment statement.
* An assignment statement consists of a variable name, an equal sign (called the assignment operator), and the value to be stored.
* Ex: spam = 42

***String conacatenation and replication:***

The meaning of an operator may change based on type of

operand.(data type of the value next to it)

For an example:+ is addition operator when it operates on two integers or floating point number.

However , when + is used on two stings values, it joins the two strings as a single strings as concatenation operator

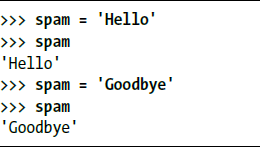
Ex=”shiv”+ “mogga” “shivamogga”

The \* is mulitiplication operator when it operates on two integers or floating point number.

However , when \* is used on stings values, it multiplies the strings as a single strings as replication operator

Ex:”arun”\*2 “arunarun”

## 



**Identifiers**

Identifiers are the name used to identify the variable,functions,any other objects. It is a sequence of letter ,number,underscore and every identifier must start with either letters or underscore

rules:

1. It can be only one word.
2. It can use only letters, numbers, and the underscore (\_) character.
3. It can‘t begin with a number.
4. It cannot be a keyword

# Keywords

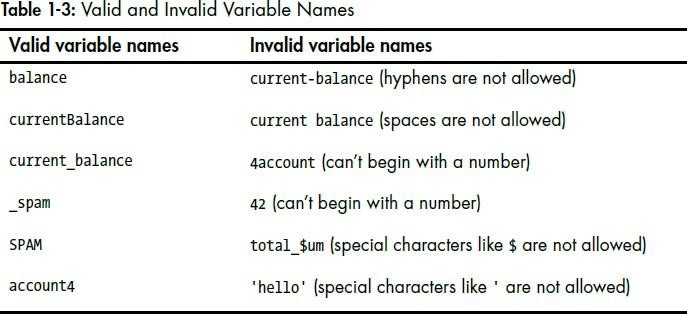
Keywords are the reserved word that are used to define or structure of the language.since they are predefined or reserved words used for definite meanings.

Ex int, float

Rules:

1. it should not be an identifiers
2. They cannot be used as variable names or function name
3. 3The meaning of keyword cannot be changed be programmer

4.all keywords written in lowercase letters



## The print() Function

The print() function used to display output from the console.it takes one or more arguments,which can be a string variable or a number and display them as output.



A value that is passed to a function call is an *argument.*

The quotes are not printed to the screen. They just mark where the string begins and ends; they are not part of the string value.

## The Input Function

* + - The input() function used to get user’s input from the console.it allows the programmer to waits for the user to type some text on the keyboard and press ENTER
    - Syntax:

variable\_name=input(prompt)



## Printing the User’s Name

* + - The following call to print() actually contains the expression 'It is good to meet you, ' + myName between the parentheses.



* + - Remember that expressions can always evaluate to a single value.
    - If 'Al' is the value stored in myName on the previous line, then this expression evaluates to 'It is good to meet you, Al'.
    - This single string value is then passed to print(), which prints it on the screen.

## The len() Function

* + - * The len() function used to determine the number of elements present in sequence or collection. len() function takes an arguments that can be a string value ,list or tuples, and any other iterable data types and return number of elemnts in that object.



## The str(), int() and float() Functions

1. **str()**

The str() is a built in function , that converts any values to string representation . it takes an arguments such as int, float, boolean or objects and returns it string representation.

Ex=num=42

str(num)

#returns “42”

Ex=num=42.0

str(num)

#returns “42.0”

1. **int()**

The int() is a built in function , that converts a values to integer.. it takes various types of arguments such as int, float, strings containing number.

Ex=num=42

str(num)

#returns 42

Ex=num=”42”

str(num)

#returns 42

1. **float()**

The float() is a built in function , that converts a values to floating point numbers. It takes various types of arguments such as int, float, strings containing number.

Ex=num=42

str(num)

#returns 42.0

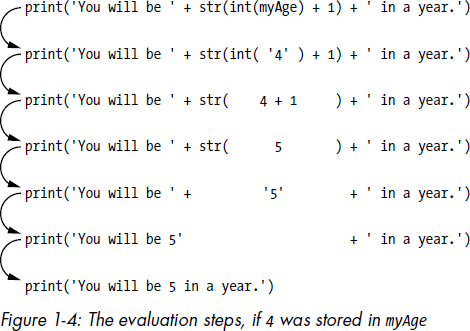
Ex=num=”42.5”

str(num)

#returns 42.5

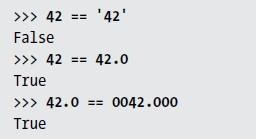
**Another input:**

* Let‘s say the user enters the string '4' for myAge.
* The string '4' is converted to an integer, so you can add one to it. The result is 5.
* The str() function converts the result back to a string, so we can concatenate it with the second string, 'in a year.', to create the final message. These evaluation steps would look something like below:



## Text and Number Equivalence

* Although the string value of a number is considered a completely different value from the integer or floating-point version, an integer can be equal to a floating point.



**Write a function to calculate factorial of a number. Develop a program to compute binomial coefficient (Given N and R).**

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n-1)

def binomial\_coefficient(n, r):

if r > n:

print(“invalid input”)

else:

return factorial(n) / (factorial(r) \* factorial(n-r))

n = 5

r = 2

coefficient = binomial\_coefficient(n, r)

print(f"The binomial coefficient of {n}C{r} is {coefficient}.")

**What is variable. Rules of variable. Rules for scope of variables:**

Ans: the variables are the names given to the programming elements memory locations.

Rule:

* It cannot be a key word
* It can be only a word
* Cannot begin with number
* Can use letters ,numbers and underscore characters

Rules for the scope of variables :

* Local variables cannot have global scope

Ex: def spam():

eggs=500

spam()

print(eggs)

output:

error

* Local scope cannot be used as global scope

Ex: def spam():

eggs=99

spam1()

print(eggs)

def spam1():

eggs=0

print(eggs)

spam()

output:

0

99

* Global variables can be read from local scope

Ex: def spam1():

print(eggs)

eggs=42

spam()

print(eggs)

output:

42

42

* Local variable and global variables with same name

Ex: def spam():

eggs=101

print(eggs)

def spam1():

eggs=105

print(eggs)

eggs=501

spam()

spam1()

print(eggs)

**What is an arithmetic expression? . What is the output of this statement?**

**‘hello world’ + 100 + ’how are you’ explain the reason if the statement**

**produces an error.**

Ans: an arithmetic expression consists of operands and arithmetic operators like +, - ,\* , /, % and \*\* that are evaluated to a single value.

The following statement ‘hello world’ + 100 + ’how are you’ results in

TypeError: can only concatenate str (not "int") to str.

Its because we can concatenate a string type value with another string type value, but not with integer

**Explain TWO ways of importing modules into application in Python with syntax and suitable programming examples.**

* All Python programs can call a basic set of functions called built-in functions, including the print(), input(), and len() functions.
* Python also comes with a set of modules called the standard library.
* Before we can use the functions in a module, we must import the module with an import statement. In code, an import statement consists of the following:

1. The import keyword
2. The name of the module
3. Optionally, more module names, as long as they are separated by commas

* Example with output:

* The random.randint() function call evaluates to a random integer value between the two integers that you pass it.
* Since randint() is in the random module, we must first type random. in front of the function name to tell Python to look for this function inside the random module.
* Here‘s an example of an import statement that imports four different modules:



***from import Statements***

* An alternative form of the import statement is composed of the from keyword,followed by the module name, the import keyword, and a star; for example, from random import \*.
* With this form of import statement, calls to functions in random will not need the random prefix.
* However, using the full name makes for more readable code, so it is better to use the normal form of the import statement.

**Develop a Python program to generate Fibonacci sequence of length (N). Read N from the console.**

def fibonacci(n):

if n< 0:

print("incorrect input") elif n==1:

return 0

elif n==2:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

n=int(input("Enter the value N")) if n>0:

x = fibonacci(n) print(x)

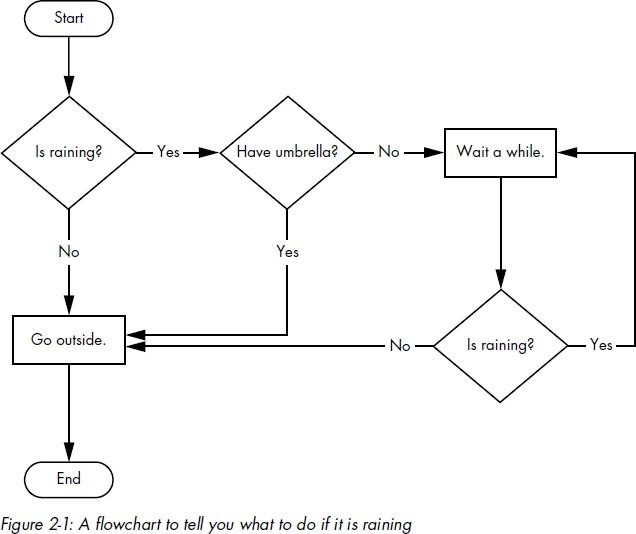
else:

print("Enter the correct values")

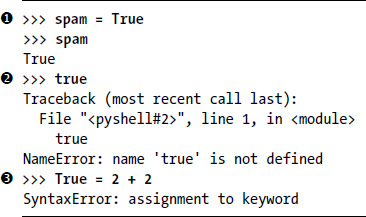
# CHAPTER 2: FLOW CONTROl

## Introduction

* + - Flow control statements can decide which Python instructions to execute under which conditions.
    - These flow control statements directly correspond to the symbols in a flowchart
    - In a flowchart, there is usually more than one way to go from the start to the end.
    - Flowcharts represent these branching points with diamonds, while the other steps are represented with rectangles.
    - The starting and ending steps are represented with rounded rectangles.



* + - The Boolean data type has only two values: True and F
    - When typed as Python code, the Boolean values True and False lack the quotes you place around strings, and they always start with a capital T or F, with the rest of the word in lowercase.
    - Examples:

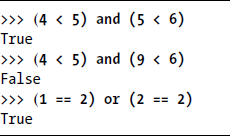


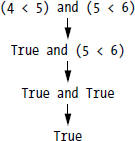
* + - Like any other value, Boolean values are used in expressions and can be stored in variables ❶. If we don‘t use the proper case ❷ or we try to use True and False for variable names ❸, Python will give you an error message.

### Difference Between the == and = Operators

* + - The == operator (equal to) asks whether two values are the same as each other.
    - The = operator (assignment) puts the value on the right into the variable on the left.
    -  We often use comparison operators to compare a variable‘s value to some other value, like in the eggCount <= 42 ❶ and myAge >= 10 ❷ examples.

## Mixing Boolean and Comparison Operators

* + - Since the comparison operators evaluate to Boolean values, we can use them in expressions with the Boolean operators. Ex:
    - The computer will evaluate the left expression first, and then it will evaluate the right expression. When it knows the Boolean value for each, it will then evaluate the whole expression down to one Boolean value. You can think of the computer‘s evaluation process for (4 < 5) and (5 < 6) as shown in Figure below:



* + - We can also use multiple Boolean operators in an expression, along with the comparison operators.



* + - The Boolean operators have an order of operations just like the math operators do. After any math and comparison operators evaluate, Python evaluates the not operators first, then the and operators, and then the or operators.

## Elements of Flow Control

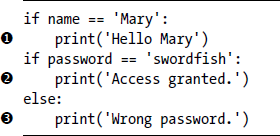
* + - Flow control statements often start with a part called the condition, and all are followed by a block of code called the clause.

##### Conditions:

* + - The Boolean expressions you‘ve seen so far could all be considered conditions, which are the same thing as expressions; condition is just a more specific name in the context of flow control statements.
    - Conditions always evaluate down to a Boolean value, True or False.
    - A flow control statement decides what to do based on whether its condition is True or False, and almost every flow control statement uses a condition.

***Blocks of Code:***

* + - Lines of Python code can be grouped together in blocks. There are three rules for blocks.

1. Blocks begin when the indentation increases.
2. Blocks can contain other blocks.
3. Blocks end when the indentation decreases to zero or to a containing block‘s indentation.
   * + The first block of code ❶ starts at the line print('Hello Mary') and contains all the lines after it. Inside this block is another block ❷, which has only a single line in it: print('Access Granted.'). The third block ❸ is also one line long: print('Wrong password.').

##### Program Execution:

* The program execution (or simply, execution) is a term for the current instruction being executed.

***Control statements or branching statements***

The statements which transfers the control from one place to another place in program with or without condition is called a control statements or branching statements

***Conditional statements***

The conditional statements are those statements in which the control may be transform from one place to another place in a program as to execute a certain set of statements if some condition is met else some other statements will be executed.

***01.If statements :***

An if statement Is a single selection statements . or a control statements. When a set of statements must be executed when a condition is evaluated to true else When a set of statements must be skipped when a condition is evaluated to false.

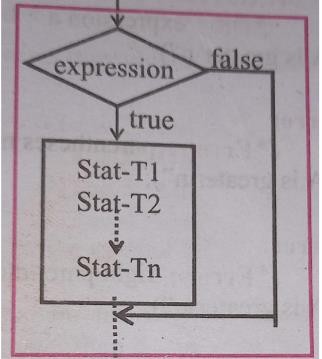
**Syntax:** if expression:

Block of code

**Example:** if name==”alice” ::

Print(“hi”)

**Flow chart:**



***Write a program to find given number is even or odd If statemants***

a=int(input(“enter the number”)) print(a)

if a%2==0:

print(“the given number is even”)

print(“the given number is odd”)

***Write a program to find greatest of two number***

a=int (input(“enter the number”)

b=int(input(“enter the value of b”)

if a>b :

print(“a is greater”)

if b>a:

print(“b is greater”)

***2.else Statements:***

An if else statement Is a two way selection statements . or a control statements. When a set of statements must be executed when a condition is evaluated to true else When an other set of statements must be executed, then a condition is evaluated to false.

**Syntax:**

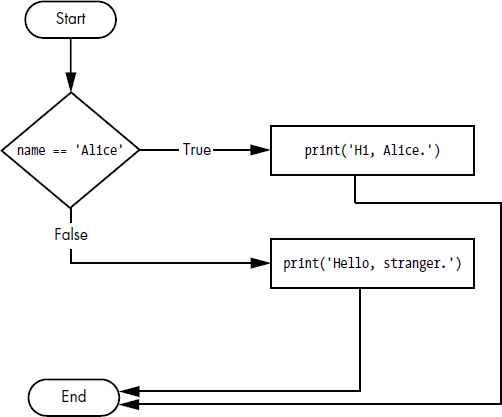
If expression:

Block of code

Else:

Block of code

**Flowchart:**



1. ***elif Statements:***

>The elif statements is an “else if” statements that always follows an if or another

elif statements

>It provides another condition that is checked only if all the previous conditions were false. And will be checked and so on.

>When there is a chain of elif statements ,only one or none of the clauses will be executed.

**Syntax:**

If condition:

#code block1

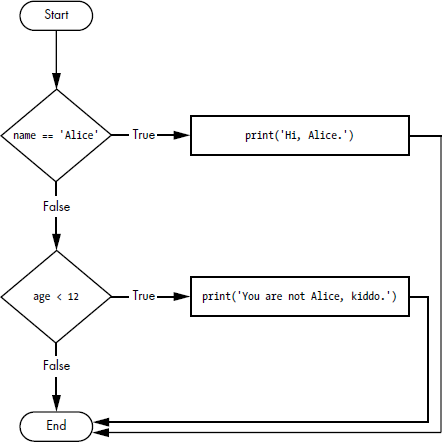
Elif condition:

#code block2

Else:

#code block 3

* + - Flowchart:



# 04.Nested if statements:

an if or ifelse statemnt with in another if or if else statemnt is called as nested if statement

# **Example program for nested if statements**

# Prompting the user to enter their age

age = int(input("Enter your age: "))

# Checking the age using nested if statements

if age >= 18:

print("You are eligible to vote.")

if age >= 21:

print("You are also eligible to drink alcohol.")

else:

print("You are not eligible to drink alcohol.")

else:

print("You are not eligible to vote.")

##### Looping statemnts:

***A set of condition must be repeatedly executed for a specified number of times or till some condition is met. The statements that help us to execute a set of statements for a specified number of times is called looping statements***

##### There are of 2 types

1. ***while loops***

While loop is a control statements using which a programmer can give instructions to computer to execute a set of statements repeatedly as long as specified condition is satisfied.once condition is false control comes out of loops.

* + In code, a while statement always consists of the following:

1. The while keyword
2. A condition (that is, an expression that evaluates to True or False.
3. A colon
4. Starting on the next line, an indented block of code (called the while clause)

Example:

num = int(input("Enter a positive integer: "))

sum = 0

while num > 0:

sum += num

num -= 1

print("The sum of numbers from 1 to the entered number is:", sum)

1. ***for loops***

***For*** loop is a control statements using which a programmer can give instructions to computer to execute a set of statements repeatedly as long as specified condition is satisfied. Once specified condition is false control comes out of loops.

It is also called pre tested loops

* + In code, a for statement looks something like for i in range(5): and always includes the following:

1. The for keyword
2. A variable name
3. The in keyword
4. A call to the range() method with up to three integers passed to it
5. A colon
6. Starting on the next line, an indented block of code (called the for clause)
   * Example and output:

|  |  |
| --- | --- |
| Example | Output |
|  |  |

Ex:numbers = [1, 2, 3, 4, 5]

print("Printing numbers using a for loop:")

for num in numbers:

print(num)

***break Statements:***

* + The break is a unconditional branching statements which are used in loops or conditional statements.
  + The statements following break statements are skipped and terminated the lops
  + In code, a break statement simply contains the break keyword.

Ex:

n=int(input(“enter the value of n”))

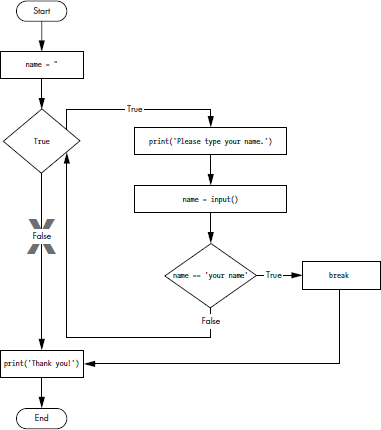
if n==1:

break:

else:

print(n)

* + Flowchart:

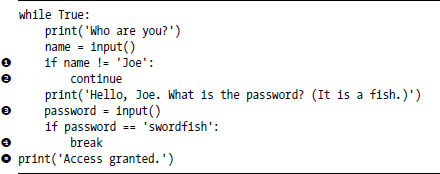


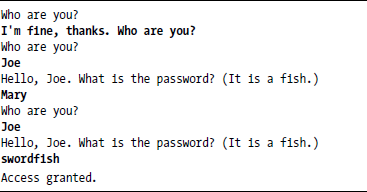
##### Continue statement

* + Like break statements, continue statements are used inside loops. And it is also unconditional branching statements.

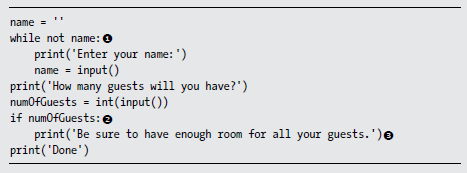
When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and reevaluates the loop‘s condition.

* + Example and Output:





* + Flowchart



**Discuss various methods of importing modules in Python programs. Which method is best?. Explain.**

# Ans:

# 4 ways to import a module in Python

## Importing the whole module object

## The first way is to use the import statement, using the syntax import module\_name:

## >>> import math

## >>> math.sqrt(25)

## 5.0

## Importing specific things from a module

## If you wanted to just type pi instead of math.pi, you could use the from syntax for importing:

## >>> from math import pi

## >>> pi

## 3.141592653589793

## Avoiding name collision when importing

## To fix this we could rename sqrt from the cmath module as we import it, by using the as syntax:

## >>> from math import sqrt

## >>> from cmath import sqrt as csqrt

## >>> sqrt(25)

## 5.0

## >>> csqrt(25)

## (5+0j)

## Importing a module under a different name

**>>>** import math as m

Now we can use m.pi or m.sqrt:

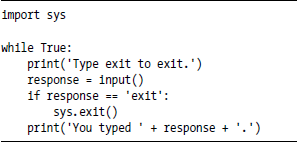
>>> m.pi

3.141592653589793

>>> m.sqrt(25)

5.0

* 1. **Ending a Program Early with sys.exit()**
     + The last flow control concept is how to terminate the program. This always happens if the program execution reaches the bottom of the instructions.
     + However, we can cause the program to terminate, or exit, by calling the sys.exit() function. Since this function is in the sys module, we have to import sys before your program can use it.

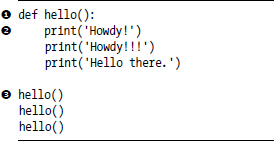


* + - This program has an infinite loop with no break statement inside. The only way this program will end is if the user enters exit, causing sys.exit() to be called.
    - When response is equal to exit, the program ends.
    - Since the response variable is set by the input() function, the user must enter exit in order to stop the program.

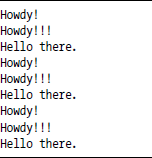
# CHAPTER 3: FUNCTIONS

## Introduction

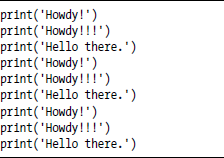
* + - A function is like a mini-program within a program.
    - A large programs are divided into manageable small pieces called modules. Where each module does specific task .thus, each module is called function
    - Example:



* + - the output looks like this:



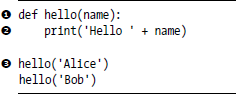
* + - A major purpose of functions is to group code that gets executed multiple times. Without a function defined, we would have to copy and paste this code each time, and the program would look like this:



## def Statements with Parameters

def is a keyword which is used to define a function.it is placed before a function name that is provided by user to create a user defined function

* + - We can also define our own functions that accept arguments.
    - Example with output:

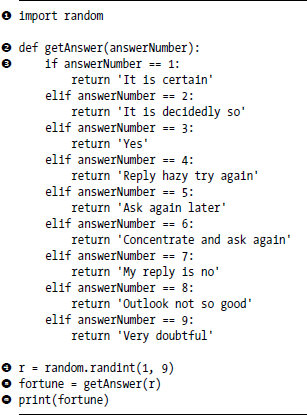


* + - The definition of the hello() function in this program has a parameter called name ❶.
    - A parameter is a variable that an argument is stored in when a function is called. The first time the hello() function is called, it‘s with the argument 'Alice' ❸.
    - The program execution enters the function, and the variable name is automatically set to 'Alice', which is what gets printed by the print() statement ❷.
    - One special thing to note about parameters is that the value stored in a parameter is forgotten when the function returns.

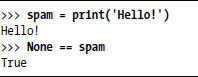
## Return Values and Return Statements

* + - The value that a function call evaluates to is called the return value of the function.
    - Ex: len(‗Hello‘)  Return values is 5
    - When creating a function using the def statement, we can specify what the return value should be with a return statement.
    - A return statement consists of the following:

1. The return keyword
2. The value or expression that the function should return.
   * + When an expression is used with a return statement, the return value is what this expression evaluates to.
     + For example, the following program defines a function that returns a different string depending on what number it is passed as an argument.



## The None Value

* + - In Python there is a value called None, which represents the absence of a value.
    - None is the only value of the None Type data type.
    - This value-without-a-value can be helpful when we need to store something that won‘t be confused for a real value in a variable.
    - One place where None is used is as the return value of print().
    - The print() function displays text on the screen, but it doesn‘t need to return anything in the same way len() or input() does. But since all function calls need to evaluate to a return value, print() returns None.

## Keyword Arguments and print()

* + - Most arguments are identified by their position in the function call.
    - For example, random.randint(1, 10) is different from random.randint(10, 1).
    - The function call random.randint(1, 10) will return a random integer between 1 and 10, because the first argument is the low end of the range and the second argument is the high end while random.randint(10, 1) causes an error.
    - However, keyword arguments are identified by the keyword put before them in the function call.
    - Keyword arguments are often used for optional parameters.
    - For example, the print() function has the optional parameters end and sep to specify what should be printed at the end of its arguments and between its arguments (separating them), respectively.



* + - The two strings appear on separate lines because the print() function automatically adds a newline character to the end of the string it is passed.
    - However, we can set the end keyword argument to change this to a different string.
    - For example, if the program were this:



* + - The output is printed on a single line because there is no longer a new-line printed after 'Hello'. Instead, the blank string is printed. This is useful if we need to disable the newline that gets added to the end of every print() function call.
    - Similarly, when we pass multiple string values to print(), the function will automatically separate them with a single space.



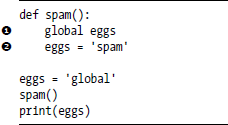
* + - But we could replace the default separating string by passing the sep keyword argument

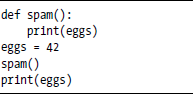
.

**Explain Local and Global Scope in Python programs. What are local and global variables? How can you force a variable in a function to refer to the global variable?**

* Parameters and variables that are assigned in a called function are said to exist in that function’s *local scope.*
* Variables that are assigned outside all functions are said to exist in the *global scope*.
* A variable that exists in a local scope is called a *local variable*, while a variable that exists in theglobal scope is called a *global variable.*
* Scopes matter for several reasons:
  1. Code in the global scope cannot use any local variables.
  2. However, a local scope can access global variables.
  3. Code in a function‟s local scope cannot use variables in any other local scope.
  4. We can use the same name for different variables if they are in different scopes. That is, therecan be a local variable named spam and a global variable also named spam.
* We can force a variable in a function to refer to the global variable using global statement as shown below:

**Program Output**

 F:\Ours\SuViKa\Lockdown\ODD SEM-2020\Python\Images\Chapter 3\23.PNG

Because eggs is declared global at the top of spam() ❶, when eggs is set to 'spam' ❷, thisassignment is done to the globally scoped eggs. No local eggs variable is created.

***Local variable:***

The variable which are declared within a function is called local variable .it can be accessed within the block

##### Global variable:

The variable which are declared outside a function call is called global variable .it can be accessed anywhere in the program .normally the variable is declared at beginning of program.

## The Global Statement

a global statement is used to access and modify a global variable within a function. By default, variables defined inside a function are considered local to that function, meaning they cannot be accessed or modified outside of it. However, by using the global statement, you can indicate that a variable is intended to be used globally.

global\_var = 5 # global variable

def update\_global\_var():

global global\_var # indicating global\_var is a global variable

global\_var += 10

def print\_global\_var():

print("Global variable:", global\_var)

print\_global\_var() # Output: Global variable: 5

update\_global\_var()

print\_global\_var() # Output: Global variable: 15

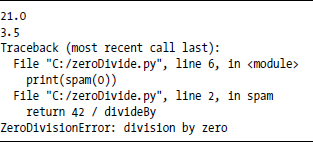
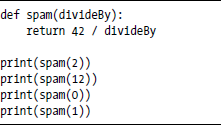
In this program, we have a global variable `global\_var` initially set to 5. The function `update\_global\_var()` uses the global statement to indicate that `global\_var` is a global variable and increments its value by 10. The function `print\_global\_var()` simply prints the value of `global\_var`.

## Exception Handling

It is a mechanism to handle the run time error ,when an error is encountered in a statement .the program will not crash .instead it detects, handle them and continue to run

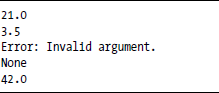
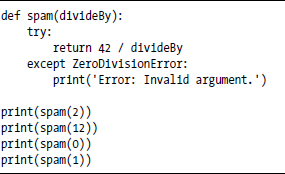
* + - If we don‘t want to crash the program due to errors instead we want the program to detect errors, handle them, and then continue to run.
    - For example,

#### Program Output



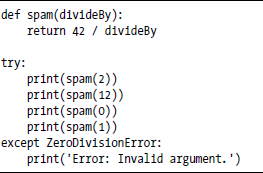
* + - A ZeroDivisionError happens whenever we try to divide a number by zero. From the line number given in the error message, we know that the return statement in spam() is causing an error.
    - Errors can be handled with try and except statements.
    - The code that could potentially have an error is put in a try clause. The program execution moves to the start of a following except clause if an error happens.
    - We can put the previous divide-by-zero code in a try clause and have an except clause contain code to handle what happens when this error occurs.

#### Program Output



* + - Note that any errors that occur in function calls in a try block will also be caught. Consider the following program, which instead has the spam() calls in the try block:

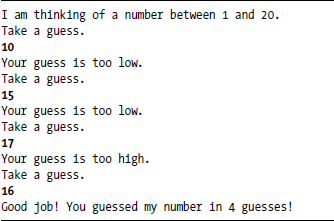
#### Program Output

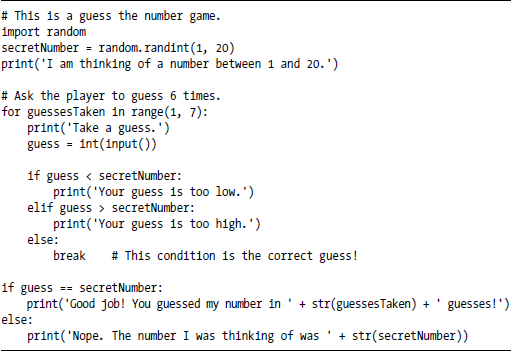


* + - The reason print(spam(1)) is never executed is because once the execution jumps to the code in the except clause, it does not return to the try clause. Instead, it just continues moving down as normal.

## A Short program: Guess the Number

* + - This is a simple ―guess the number‖ game. When we run this program, the output will look something like this:





**Write a function named DivExp which takes TWO parameters a, b and returns a value c (c=a/b). Write suitable assertion for a>0 in function DivExp and raise an exception for when b=0.**

**Develop a Python program which reads two values from the console and calls a function DivExp**.

Def DivExp(): try:

a=int(input(“enter an integer:”)) b=int(input(“enter another integer”)) c=a/b

except ZeroDivisionError:

c=”you cannot divide by 0”

print(c) DivExp()

**OUTPUT:**

Enter an integer :10 Enter another integer:5 2.0

**What is a need for role of precedence? Illustrate the rule of precedence in python program with example**

The role of precedence in programming languages, including Python, is to determine the order in which operators are evaluated in an expression. It helps to avoid ambiguity (confusion)and ensure that expressions are evaluated correctly.

In Python, the precedence of operators is defined by a set of rules. These rules specify which operators are evaluated first when an expression contains multiple operators.

Here's an example program that illustrates the rule of precedence in Python:

python

x = 10 + 5 \* 2 - 3

print(x)

Output:

17

In this example, the expression `10 + 5 \* 2 - 3` contains multiple operators: `+`, `\*`, and `-`. According to the rule of precedence, the multiplication operator `\*` has a higher precedence than the addition and subtraction operators `+` and `-`.

So, the expression is evaluated in the following order:

1. `5 \* 2` is evaluated first, resulting in `10`.

2. `10 + 10` is evaluated next, resulting in `20`.

3. Finally, `20 - 3` is evaluated, resulting in `17`.

Therefore, the value of `x` is `17`.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&