ROLL NO:- U15KZ25S0003 | UNIT-I

Introduction to python

* What is python?
  -> Python is a interpreted object oriented general purpose high level programming language.

* Interpreted :- Python code is executed line by line by an interpreter at runtime rather than being compiled into machine code

* Object oriented :- Python is built around the Concept of objects & classes which allows you to organize code in a reusable & modular way.

* General purpose :- Means it can be used for multiple application such as data science, machine learning, desktop application, web application, Scripts etc.

* High-level programming language :- means human understandable language i.e Human readable.

-> History of Python :-

* It was created by Guido van Rossum, and released in 1984.
* Guido van Rossum Ps Pan of "MONTY PYTHON'S FLY-ING CIRCUS", this is a famous to show in netherlands
* Named after monty pyton.
* Open Source can be downloaded from ~~www.~~ www.python.org

U15KZ25S0003

* First version of python released in 1991.
* In 1994, python 1.0 was released with new feature which map, filter, lambda.
* Python 2.x also add some more feature like comprehension, garbage collection system.
* After python 2.x, python 3.x released in 2008.
* Latest version is Python 3.14.6 (released june 10, 2026)

U15KZ25S0003

1) Why PYTHON?

=) * Python works on different platform (windows, mac, linux, raspberry pi, etc.
* Python has a simple syntax similar to the English language
* Python has syntax that allows developers to write programs with fewer lines than some other programming language
* Python runs on interpreter system, meaning that code can be executed so as soon as it is written.

* Applications of PYTHON :-
- NETWORK PROGRAMMING
- DATA ANALYSIS
- ROBOTICS
- WEBSITE & APPLICATION DEVELOPMENT
- DESKTOP APPLICATION
- GAMES DEVELOPMENT
- DATA VISUALIZATION
- SCIENTIFIC CALCULATION
- MACHINE LANGUAGE & ARTIFICIAL INTELLGENCE
- 3D APPLICATION DEVELOPMENT
- AUDIO & VIDEO SOFTWARE DEVELOPMENT
- GOOGLE, NETFLIX, NASA, QUORA, FACEBOOK

Features of Python:-

1) Simple :
* Python is a simple programming language.
* When we read python program, we can feel like reading english statements.

www.python.org download

U1SKZ25S0003

* It means more clarity and less stress on understanding the syntax of the language

2] Easy to learn :-
* Python uses very few keywords
* Its programs use very simple structure.
* Developing programs in python is Easy

③ Open Sources :-
* There is no need to pay for python software.
* Python can be freely downloaded from www.python.org ~~web~~ website.
* Its source code can be ~~rea~~ read, modified and can be used in programs as desired by the programmers.

4] High level language :-
* High level languages use English words.
* These are Easy to learn & use like COBOL, PHP or java.
* Python also uses English words in its programs.

U15KZ25S0003

5] Dynamically typed :-

* In python we need not declare data type for variables, whenever we are assinging the value based on value, type will be allocated automatically.
* But java, C etc are statically typed languages because we have to provide data type at the beginning only.

6] Platform independents :-

* When a python program is compiled using a python compiler it generates byte code.
* Python's byte code represents a fixed set of instruction's that run an all operating systems and hardware.
* We can use python on almost all operating system like UNIX, LINUX, windows etc.

7] Portable :-

* Once a python program is written, it can run on any computer system using Python virtual machine (PVM)

8] Procedure and object oriented :-

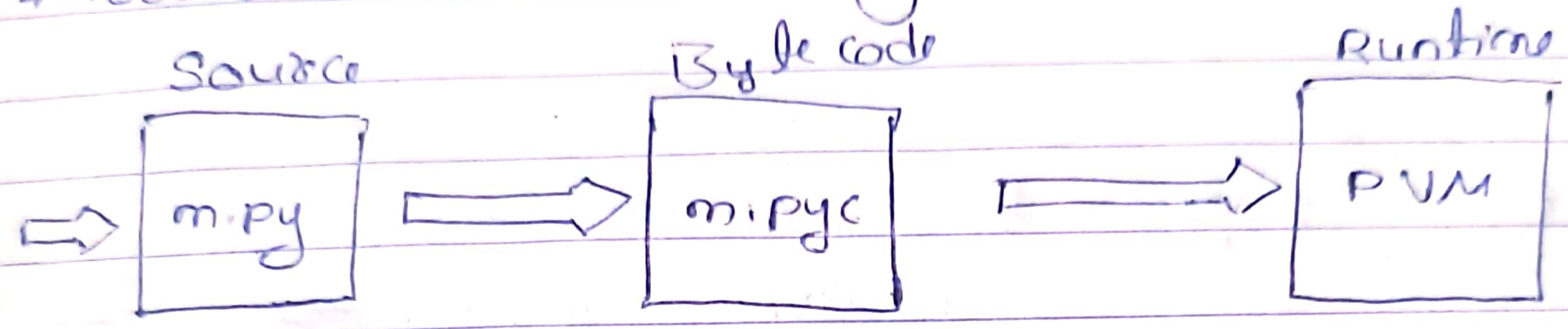
* Python is a procedure oriented as well as an object oriented programming language
* In object procedure oriented programming language C Ex:- (& pascal), the programs are building using fun & procedures.
* But in object oriented language C ex:- C++ & java, the programs uses classes and objects.

U15KZ25S0003

WORKING WITH ~~EXECUTION~~ : PYTHON

PYTHON CODE EXECUTION:

* Python's traditional runtime execution model :- Source code you type is translated to bytecode, which is then run by the python virtual machine (PVM).
* Your code is automatically Compiled, but then it is interpreted.



Source code extension is .py
Byte Code extension is .pyC (compiled python Code).

There are two modes for using the python interpreter :-

* INTERACTIVE MODE
* SCRIPT MODE

Running Python in Interactive Mode :-

* Without passing python Script file to the interpreter, directly execute code to python prompt.
* Once you're inside the python interpreter, then you can start

```
>>> Print ("Hello world")
Hello world
```

Relevant output is displayed on subsequent lines without the >>> symbol.

U15KZ25S0003

① Write a python program to find area of a Circle where radius is ~~fixed~~ 5 & Pi

⇒ r = 5

pi = 3.142

area = pi * r * r

print ("area of the circle is =", area)

~~area of the circle is = 71.55~~

Output

Area of the Circle is = 78.55

② Write a python program to calculate area of a Square.

⇒ a = 5

area = a * a

print ("area of the Square =", area)

Output: area of the Square is = 25

③ Write a python program to check the given number is positive or not

⇒ if num > 0 :

print ("Positive")

else :

print ("Negitive")

num = 5

if num >= 0:

print ("positive")

else :

print ("Negative")

Output

positive

④ Write a python program to find sum two given numbers.

⇒ a = 5

b = 10

sum = a + b

print ("Sum of two no", sum)

~~The sum is~~

Output :

⇒ ~~positive~~ 15

⑤ Write a python program to identify whether the given character vowel or ~~and~~ consonants.

⇒ ch = input ("Enter a character :")

if ch.lower() in ['a', 'e', 'i', 'o', 'u'] :

print ("It is a vowel")

else :

print ("It is a consonant")

Output :

consonant

U15KZ25S0003

① Write a python program to check whether the given year is leap year or not

⇒ year = 2024

if year%.4 == 0:

print ("It is leap year");

else:

print ("It is not leap year");

output :-

It is leap year

② Write a python program to check the given number is even or odd

⇒ a = 6 n = input ("Enter the number:");

~~if a/6 == 0~~ if n/2 == 0:

print ("The given number even)

else:

print ("The given number odd")

③ Write a python program to check calculate simple interest.

⇒ P = 100

T = 12

R = 3500

SI = P*T*R/100

print ("SI = " P*T*R ")

output

Simple interest = 60.

4) Write a python program to display largest number

⇒ num1 = 60

num2 = 30

if num1 > num2 :

print ("The given number is greater")

else:

print ("The given number is greater")

Output :- num1 i

~~output: The given no is greater~~

Output :- num1 is greather than

5) Write a python program to calculate the percentage by student by reading total mark

⇒ Marks = input ("Total Marks:")

marks = marks/625 × 100 :

print ("Total Marks:", marks)

Char in Python :-

* There is no separate char datatype in Python
* A single character is treated as a string of length 1.
* Python uses the str type for both single characters & longer strings

Ex: >>>type ('h')
<class 'str'>

Ex: You can access characters in a string using indexing:
text = 'python'
print (text [0]) #output : 'p'

RUNNING PYTHON IN SCRIPT MODE :-

* Programmers can be store Python script source code in a file with the .Py extension, and use the interpreter to execute the contents of the file.
* To execute the script by the interpreter, you have to tell interpreter the name of the file
* Working with the interactive mode is better when python programmers deal with small pieces of code as you can type & executed the immediately.

Ex - print (2)
x = 2 Output: >>2
print (x) 2

Comments :-

* SINGLE-LINE COMMENTS Begins with a hash (#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of line

* Multi line Comment is useful when we need to comment on many lines. In python, triple double quote (" " ") & single quote (' ' ') are used for multi-line commenting.

Ex:

# Write a python program to add numbers
a = 10 # assigning value to variable a
b = 20 # assigning value to variable b
" " " print the value of a and b using a new variable " " "
' ' ' print the value of a and b using a new variable ' ' '
c = a+b
print (c) Output : c = 30

DATATYPES IN PYTHON :-

* A datatype represents the type of data stored into a variable or memory and determines what operations can be done on it.
* In python, there is no limit for the size of an int datatype.

Data Types
- Numbers
  - Integer
  - Floating Point
  - complex
- Boolean
- sequences
  - String
  - Tuple
  - List
- Sets
- Dictionary

1) Numbers :-

* Number data types store numerical values
* This data type ~~stores numerical~~ is immutable [i.e. values / items cannot be changed].
* Python supports integers, floating point numbers & complex numbers

Integer	Long	Float	complex
* They are of the called just integers or int. * They are positive or negative whole numbers with no decimal part Ex: - 56	* They are long integers * They can also be represented in octal & hexadecimal representation Ex: - 56984321L	* They are written with a decimal point dividing the integer & the fractional parts. Ex: 56.778	* They are of the form a+bj where a & b are floats and j represents the square root of -1 (which is an imaginary number). * The real part of the number is a, and the imaginary part is b. Ex: square root of -1 is a complex number

- Example ON int :

```
a=10
print(a)        #output: 10
print(type(a))  # output: <class 'int'>
```

* Example ON complex :

```
c = 2+3j
print(c)        #output: (2+3j)
print(type(c))  # output: <class 'complex'>
```

② BOOLEAN : (True / False)

- In python, boolean (bool) is a data type that represents one of two values:
  - True
  - False

Example:-

```
x = True
y = False
print(x)        #output :- True
print(y)        #output :- False
print(type(x))  #output :- <class 'bool'>
```

③ Sequence :-

* A sequence is an ordered collection of items, indexed by positive integer
* It is a combination of mutable (value can be changed) & immutable (values cannot be changed) data types
* There are three types of sequence data type available in python, they are
  i) Strings
  ii) Lists
  iii) Tuples

* Example ON Float :

```
b=10.5
print(b)        #output: 10.5
print(type(b))  #output: <class 'float'>
```


i) String

* A string in python consists of a series or sequence of characters
* String in python are surrounded by either single quotation marks, or double quotation marks.
* 'Hello' is the same as "Hello".
* Individual character in a string is accessed by using a subscript (index)
* Characters can be accessed using indexing and slicing operations
* Strings are immutable i.e, The contents of the string cannot be changed after it is created.

→ Example on string :-

```
print("hello")
print('hello')
```

output :-
Hello
hello

* OPERATIONS ON STRING
  - ✓ Indexing → indexing the character
  - ✓ Slicing → dividing
  - ✓ Concatenation → Joining the data
  - ✓ Repetitions →
  - ✓ Membership →

Creating a string	>>> s="good morning"	Creating the list with elements of different data type.
* Indexing	>>>print(s[0]) g >>>print(s[2]) o	* Accessing the item in the position 0 * Accessing the item in the position 2
* Slicing (ending position -1)	>>> print(s[2:]) od morning	- Displaying items from 2nd to last
→ Slice operators used to extract part of a data type	>>> print(s[0:4]) Good	- Displaying items from 1st position to position till 3rd
* Concatenation	>>> print(s+"friends") good morningfriends	- Adding & printing the characters of two strings
* Repetition	>>>print(s*2) good morning good	Create new strings, concatenating multiple copies of the same string

in, not in (membership operator)	>>> s="goodmorning"
	 >>> 'm' in s True
	 >>> 'a' not in s True
		Using membership operators check a particular character in string or not. Returns true if present.

2) LISTS :-

* List is an ordered sequence of items.
* Values in the list are called elements / items
* It can be written as a list of comma-separated items (value) between square brackets [].
* Items in the lists can be of different data types

Operations on list :-

* Indexing
* Slicing
* Concatenation
* Repetition
* Updation, insertion, deletion.

Ex :-

my-list [10, 20, 30]

My-list [1] = 25 # Updating value

print (my-list) # Output [10, 25, 30]

Creating a list	>>> list1 = ["Python", 7.79, 101, "hello"] >>> list2 = ["god", 6.78, 9]	Creating the list with elements of different data type.
Indexing	>>> Print (list1[0]) Python >>> list1[2] 101	* Accessing the item in the position * Accessing the item in the position 2
Slicing (ending position-1)	>>> print (list1[1:5] [7.79, 101]	* Displaying items from 1st ~~item~~ till 2nd
Slice operator is used to extract part of a string, or some part of a list	>>> print (list1[1:] [7.79, 101, 'hello']	* Adding & printing the items of the list * Displaying items from 1st position till last 2nd
Concatenation	>>> print (list1 + list2) ['Python', 7.79, 101, 'hello', 'god', 6.78, 9]	* Adding & printing the items of two lists.
Repetition	>>> list2 * 3 ['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]	Creates new strings, concatenating multiple copies of the same string.

Updating the list	>>> list1[2] = 45
	 >>> print (list1)
	 ['Python', 7.79, 45, 'hello']
		Updating the list using index value

Inserting an element	>>> list1.insert (2, "program")
	 >>> print (list1)
	 ['Python', 7.79, 'program', 45, 'hello']
		Inserting an element in 2nd position

Removing an element	>>> list1.remove (45)
	 >>> print (list1)
	 ['Python', 7.79, 'program', 'hello']
		Removing an element by giving the element directly.

3) Tuple :-

* Tuple is same as list, except that the set of elements is enclosed in parentheses instead of square brackets.
* A tuple is an immutable list. i.e once tuple has been created, you can't add elements to a tuple or remove elements from the tuple.

Benefit of tuple :-

* Tuples are faster than lists.
* If the user wants to protect the data from accidental changes, tuple can be used.

U15KZ25S0003

Example ON TUPLE :-

My_tuple = (10, 20, 30)

my_tuple[1] = 15 ⇒ This will cause an error

print (my_tuple) # Output - (10, 20, 30)

Creating the Tuple	>>> t = ("python", 7.79, 101, "hello")	Creating the tuple with elements of different datatype
Indexing	>>> print (t[0]) python >>> t[1] 101	* Accessing the item in the position 0 * Accessing the item in the position 1
Slicing (ending position - 1)	>>> print (t[1:3]) (7.79, 101)	* Displaying items from 1st till end.
Concatenation	>>> t + ('raw', 67) ('python', 7.79, 101, 'hello', 'raw', 67)	* Adding tuple elements at the end of another tuple elements
Repetition	>>> print (t * 2) ('python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')	* Creates new strings, Concatenating multiple copies of the same string.

4) SETS :-

* A set is an unordered collection of unique elements.
* It is mutable (can add or remove items)
* Duplicates are automatically removed

Example on sets

a = {1, 2, 3}

b = {3, 4, 5}

print (a.union(b)) # {1, 2, 3, 4, 5}

print (a.intersection(b)) # {3}

print (a.difference(b)) # {1, 2}

print (a.union(b)) → {1, 2, 3, 4, 5}

U15KZ25S0003

Method	Example	Output	Description
add()	a = {1, 2}; a.add(3)	{1, 2, 3}	Adds a single element to the set
remove()	a = {1, 2, 3}; a.remove(2)	{1, 3}	Removes specified element (Error if not found)
pop()	a = {1, 2, 3}; a.pop()	Random element removed	Removes & returns a random element
clear()	a = {1, 2}; a.clear()	set()	Removes all elements from the set
~~clear()~~	~~a = {1, 2}; a.clear()~~	set()	
union()	{1, 2}.union({2, 3})	{1, 2, 3}	Combines all elements from both sets
intersection()	{1, 2}.intersection({2, 3})	{2}	Returns common elements
difference()	{1, 2, 3}.difference({2, 3})	{1}	Elements in first set but not in second
issubset()	{1, 2}.issubset({1, 2, 3})	True	Checks if all elements of one set are in another

5) DICTIONARY :-

* Dictionaries fall under mappings and are mutable
* Lists are ordered sets of objects, whereas dictionaries are unordered sets.
* Dictionary is created by using curly brackets. {}
* Dictionaries are accessed via keys & not via their position
* The values of a dictionary can be any Python data type
* So dictionaries are Unordered key-value-pairs (the association of a key & a value is called a key-value pair)
* Dictionaries don't support the sequence operation of the sequence data types like strings, tuples & lists.

Creating a dictionary	>>> food = {"ham": "yes", "egg": "yes", "food": 500}
	>>> print (food)
	{'food': 500, 'egg': 'yes', 'ham': 'yes'}
		Creating dictionary with elements of different data types
		Accessing the item with keys

Indexing	>>> print (food['food'])	Accessing the item with keys
Slicing (ending position - 1)	>>> print (t[1:3]) (7.79, 101)	Displaying items from 1st till end

If you try to access a key which doesn't exist, you will get an error message

```
>>> words = {"house": "how", "bat": "bater"}
>>> words["car"]
```

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'car'

creating a dictionary → Assignment 6

```
student={
"name": "Sowmya",
"age": 25,
"course": "BCA"
}
# Accessing value
print (student["name"]) #Output: sowmya
print (student["age"]) # output: 25
# Adding a new key-value pair
student["grade"] = "A"
Print (student)
#output: {'name': 'Sowmya', 'age': 25, 'course': 'BCA', 'BCA',
'grade': 'A'}
```

VARIABLES, EXPRESSIONS AND STATEMENTS :-

Values & Types :

* A value is one of the basic things a program works with.
* It may be like 2, 10.5 "hello" etc
* Each value in python has a type
* Type of 2 is integer; type of 10.5 is floating point number; "hello" is string etc
* The type of value can be checked using type function as shown below


```
>>> type ("hello")
<class 'str'>
>>> type (3)
<class 'int'>
>>>type (10.5)
<class 'float'>
>>> type ("15")
<class 'str'>
```

* In the above four examples, one can make out various types str, int & float.
* Observe the 4th example - it clearly indicates the whatever enclosed with in a double quote is a string.

ASSIGNMENT - 1

1] Slicing in list :

```
>>> list1 = ["Python", 7.79, 101, "hello"]
>>> list2 = ["good", 6.789]
Print (list1[:3])
#output = ["python", 7.75, 101]
print (list1[2:])
# Output = [101, "hello"]
```

2] Removing an element is list

```
>> list1.remove (101)
print (list1)
["python", 7.79, "hello"]
```

Using index value

```
list1.remove[0]
```

Index value will not support.

U15KZ25S003

3] Inserting an element in list

```
>list1.insert ("Bhagy")
print (list1)
# output: ["Python", 7.9, 101, "Hello", "Bhagya"]
list1.insert (2,5)
print (list1)
# output - error
```

4] Sets in union

```
> a = {1,2,3}
b = {3,4,5}
Print (b union (a))
# output = {1,2,3,4,5}
```

5] Dictionary adding new key

```
Student = {"name": "Bhagya",
           "age": 25,
           "Course": "BCA"}

Student ["grade"] = "A"
print (Student ["age"])   #output = 25
print (Student ["grade"]  #output = A
```

U15KZ25S0003

VARIABLES :-

* A variable which helps to store a value in the program
* Variable may take value that can be modified whenever required in the program.
* Note that, in Python, a variable need not be declared with a specific type before its use.
* Whenever we wanted a variable, just use it. The type of it will be decided by the value assigned to it
* A value can be assigned to a variable using assignment operator (=).

* Assigning value to variable :
* Value should be given on the right side of assignment operator (=) and variable on left side.

Ex :- >>> counter = 45
print (counter)

* Assigning a single ~~multiple~~ values to several variables simultaneously:

Ex >>> a = b = c = 100

* Assigning multiple values to multiple variables:

Ex: >>> a, b, c = 2, 4, "ram" // uppercase

Imp

Imp Rules for Python Variables :-

* A variable name must start with a letter or the underscore character.
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric character & underscores (a-z, 0-9)
* Which is usually used to combine variables with two words like my_salary, student_name etc. no other special characters like @, $ etc are allowed

U15KZ25S0003

* Variable names are case-sensitive (age, Age & AGE are three different variables.
* Keywords should not use as a variable name.

Examples of illegal variable names:
- 1 var (starts with a digit)
- my-variable (contains a hyphen)
- class (is a reserved keyword)
- my var (contains a space.
- @ count (contains a special character)

Keyword
- Keywords are the reserved word in python.
- We cannot use a keyword as variable name, fun name, or any other identifier.
- They are used to define the syntax and structure of the python language.
- keywords are case sensitive.

False	class	finally	is	return
None	continue	for	lamada	try
True	def	from	Nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

U15KZ25S0003

IDENTIFIERS :-
* Identifiers is the name given to entities like class, functions, variables etc. in python.
* Identifiers can be a combination of letters in lowercase (a to z) or uppercase (a to z) or digits (0 to 9) or an underscore (_).
* An identifier cannot start with a digit.
* keywords cannot be used as identifiers.
* Cannot use special symbols like !, @, #, $, % etc. In our identifier.
* Identifier can be of any length.

Valid declarations	Invalid declarations
Num	Number 1
Num	num 1
Num 1	the addition of program
_Num	1Num
NUM_temp2	Num.no
	if
	else

STATEMENTS :-
* A statement is a small unit of code that can be executed by the python interpreter.
* It indicates some action to be carried out.
* In fact, a program is a sequence of such statements.
* Two kind of statements are: print being an expression statement and assignment statement.

Following are the examples of statements

```
>>> print("hello") # printing statement
hello
>>> x = 5 # assignment statement
>>> print(x) # printing statement
5
```

OPERATORS AND OPERANDS :-

- special symbols used to indicate specific tasks are called operators.
- An operator acts on one or single operand (unary operator) or two operands (binary operator).
- There are several types of operators like arithmetic operators, relational operators, logical operators etc in python.
- Arithmetic operators are used to perform basic operations as listed below.

Operator	Meaning	Example
+	Addition	Sum = a+b
-	Subtraction	Diff = a-b
*	Multiplication	Pro = a*b
/	Division	Q = a/b x = 5/3 (x will get a value 1.66666667)
//	Floor Division - returns only integral part after division	F = a//b x = 5//3 (x will get a value 1)
%	Modulus remainder after division	R = a%b (Remainder after dividing a by b)
**	Exponent	E = x**y (means x to the power of y)

- Relational (or) Comparison operators are used to check the relationship (like less than, greater than etc) between two operands.
- These operators return a boolean value - either True or False.

- Assignment operators :- apart from simple assignment operator = which is used for assigning values to variables, python provides compound assignment operators.

For example :-

x = x+y can be written as x += y

- Now, += is compound assignment operator.
- Similarly, one can used most of the arithmetic and bitwise operators (only binary operators, but not unary) like *, /, %, //, &, ^ etc. As compound assignment operators.

Example :-

```
>>> x=3
```

Expressions :-

- A combination of values, variables and operators is known as expression.
- Following are few examples of expression -

```
x = 5
y = x+10
z = x-y*3
```

- The python interpreter evaluates simple expressions & gives results even without print().

Example :-

```
>>> 5
5          # displayed as it is
>>> 1+2
3          # displayed the sum.
```

U15KZ25S0003

15+20=35+6

((3+4)*5)+6 ⇒ 41

ORDER OF OPERATIONS :-

- When an expression contains more than one operator, the evaluation of operators depends on the precedence of operators.
- The python operators follow the precedence (which can be remembered as pemdas) as given below.

-> Parenthesis have the highest precedence is any expression.
✓ The operations within parenthesis will be evaluated first.
✓ For example, in the expression (a+b)*c, the addition has to be done first and then the sum is multiplied with c.

-> Exponentiation has the end precedence.
✓ But, it is right associative. That is, if there are two exponentiation operations continuously, it will be evaluated from right to left. (unlike

For example:

```
>>>print (2**3)   # it is 2^3
9
>>> print (2**3**2)   # 2^3^2
512
```

- Multiplication & Division are the next priority. out of these two operations, whichever comes first in the expression is evaluated.

```
>>>print(5*2/4)   # multiplication & then division e.g.
>>> print (5/4*2)   # division & then multiplication e.g.
```

- Addition & Subtraction are the least priority. out of these two operations, whichever appears first in the expression is evaluated i.e., the are evaluated from left to right.

U15KZ25S0003

STRING OPERATION :-

✓ String concatenation can be done using + operator as shown below.

```
>>> x = "32"
>>> y = "45"
>>> print (x+y)
3245
```

- observe the output :- here, the value of y (a string "45", but not a number 45) is placed just in front of value of x (a string "32"). Hence the result would be "3245" & its type would be string.

- Note :- One can use single quotes to enclose a string value, instead of double quotes.

ACCEPTING USER INPUT :-

- Python uses the built-in function input() to read the data from the keyboard.
- When this function is invoked, the user input is accepted. The input is read till the user presses enter-key.

For example:

```
>>> str1 = input()
hello how are you?        # User input  ⇒ Assignment - 7
>>>print ("string is ", str1)
string is hello how are you?   # Priority str1
```

- When input() function is used, the cursor will be blinking to receive the data.
- For a better understanding, it is better to have a prompt message for the user informing what needs to be entered as input.

Logical operators

1) and → if both the operands are True
2) or → True or False
3) not → [illegible]

U15KZ25S0003

CONDITIONAL EXECUTION :-

- In general, the statements in a program will be executed sequentially.
- But, sometimes we need a set of statements to be executed based on some conditions.

Boolean expressions :

- A boolean expression is an expression which results in True or False.
- The true and false are special values that belong to class bool

check the following

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

OPERATOR	MEANING	EXAMPLE
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b
==	Comparison	a == b
!=	Not equal to	a != b
is	is same as	a is b
is not	is not same as	a is not b

Logical Operators :-

There are 3 logical operator in Python as shown in Table

U15KZ25S0003

Operator	Meaning	Example
and	Returns true, if both operands are true	a and b
or	returns true if any one of two operands is true	a or b
not	Return true, if the operand is false (it is a unary operator)	not a

Note :-

1) Logical operators treat the operands as boolean (true or False)
2) Python treats any non-zero number as true & zero as false
3) While using & operator, if the first operand is false, then the second operand is not evaluated by Python. Because false and'ed with anything is false
4) In case of or operator, if the first operand is true, the second operand is not evaluated. Because true or'ed with anything is true

INDENTATION :-

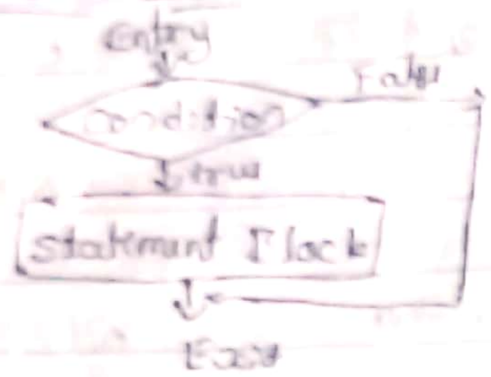
- Most of the programming languages like C, C++, Java use braces {} to define a block of code. But, python uses indentation
- Blocks of code are denoted by line indentation
- It is a space given to the block of code for codes for classes and function definitions or flow control.

Conditional execution :-

- The basic level of condition execution can be achieved in Python by using if statement
- The syntax and flowcharts are as below.

If condition:

Statement block :-



* Observe the colon symbol after condition.
* When the condition which is a boolean Expression is true, the statement block will be Executed.
* Otherwise, it is skipped.
* A set (block) of statements to be executed under it is decided by the indentation (tab space) given

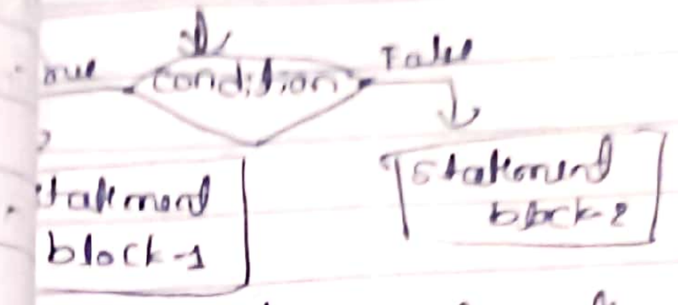
Ex: age =35
if age >=18:
print ("Eligible to value")

* Usually, the if conditions have a statement block.
* In any case, the programmer fails to do nothing when the condition is true, the statement block can be skipped by just typing pass statement as shown below.

Ex: if x<0:
pass # do nothing when x is negative.

Alternative Execution :-

* A second form of if statement is alternative execution, in which there are two possibilities based on condition evaluation
* Here, when the condition is true, one set of statements will be executed and when the condition is false, another set of statements will be executed.
* The Syntax & flowchart are as give below



* As the condition will be either true or false, only one among statement block-1 & Statement block-2 will be get executed,
* These two alternations are known as branches

Ex:- age = 2
if age >=18:
print ("Eligible for vote")
else:
print ("Not Eligible for vote")

Nested and conditionals :-

* The conditional statements can be nested
* That is, one set of condition as statements can be nested inside the other.
* It can be done in multiple ways depending on programmer requirements.

Ex:- if user == "admin":
print ("access granted").
else:
print ("Wrong password")
else:
print ("Unknown user")

Chained Conditions :-

* Some of the programs require more than one possibility to check for checking each of statements.
* That means, we many have more than one branch. This is solved with the help of chained conditionals.

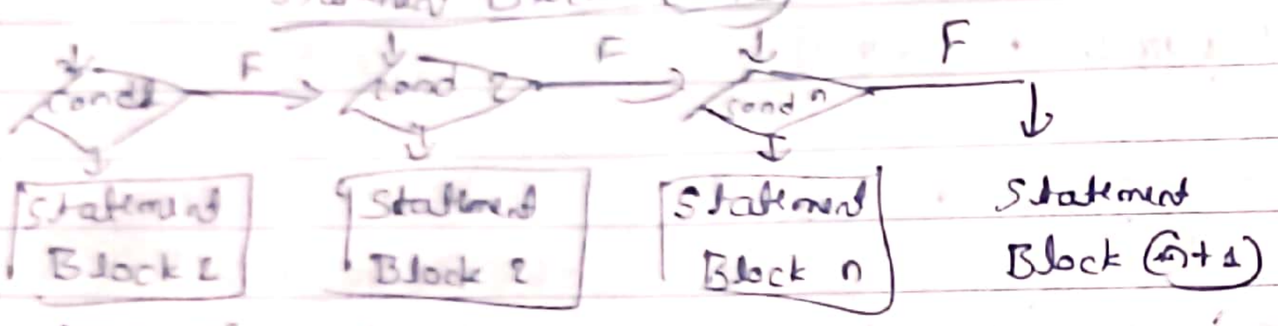
Syntax :-
if condition 1:
Statement Block -1

U15KZES0003

elif Condition 2:
    statement Block 2

elif Condition n:
    statement Block n

else
    statement Block (n+1)



Ex: if marks >=90:
    print ("a grade")
elif marks >=75:
    print ("b grade")
else:
    print ("C grade")

Iterations:-

- Computers are often used to automate repetitive tasks. Repeating identical or similar tasks without making errors is
- Something that computers do well and people do poorly
- Repeated execution of a set of statements is called "iteration"

Updating Variables:

- A common pattern in assignment statements is an assignment statement that updates a variable, where the new value of the variable depends on the old.

$x = x+1$

U15KZES0003

- This means 'get the current value of x, add 1, and then update x with the new value.
- If you try to update a variable that doesn't exist, you get an error, because Python evaluates the right side before it assigns a value to x:

$x = x+1$

Name Error: name 'x' is not defined

- Before you can update a variable, you have to initialize it, usually with a simple assignment:

$x = 0$

$x = x+1$

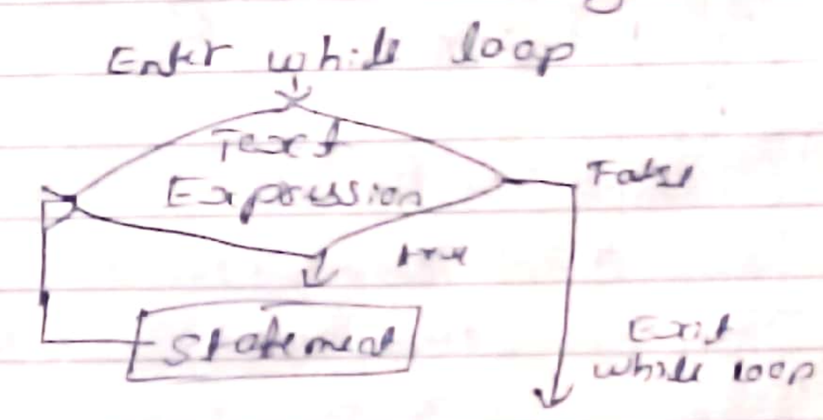
Updating a variable by adding is called an increment, subtraction, is called 'Decrement.

The while loop:

While loop is used to execute a block of statements repeatedly until a given condition is satisfied. when the condition becomes False, the line immediately after the loop in the program is executed.

Syntax:-

While expression:
    statement (s)



Python example for while loop

```
count = 0
while (count <3):
    count = count + 1
    print ("Hello BCA")
```

output:- Hello BCA
Hello BCA
Hello BCA

count = 0
while (count < 3):
count = count + 1
print ("Hello")
output: Hello 1
Hello 2
Hello 3

U15KZ25S0003

Infinite loops :-

- The value of the condition is always true. Therefore, the body of the loop is run infinite times until the memory is full.
- It is an infinite loop because there is No iteration variable telling you how many times to execute the loop.

Ex: age = 21
- The test condition is always True
while age > 10:
print (Infinite Loop!)

Output: Infinite loop
Infinite Loop
Infinite Loop
...

Finishing iterations with continue :-

The continue statement skips the current iteration & moves to the next one, without executing the remaining code inside the loop for that iteration.

SYNTAX :-

```
While condition :
    # Code before continue
    if some condition:
        Continue
    #code after continue (this will be skipped if continue runs)
```

Example:

```
i = 0
while i<5:
    i = i+1
    if i==3:
        continue
    print (i)
```

Output :- 1
2
3
4
5

U15KZ25S0003

DEFINITE LOOPS USING FOR

- Sometimes we want to loop through a set of things such as a list of words, the lines in a file, or a list of numbers
- When we have a list of things to loop through, we can construct a definite loop using a for statement
- We call the while statement An indefinite loop because it simply loops until some condition becomes false, whereas the loop is looping through a known set of items so it runs through as many iterations as there are items in the set.

Syntax :-

```
For variable in sequence:
    #body of loop
```

- Variable :- a temporary name for each item in the sequence.
- Sequence : a range, list, string, or other iterable

Example 1:- PRINT NUMBERS FROM 1 TO 5

```
for i in range(1,5):
    print (i)
```

output :- 1
2
3
4

→ Assignment - 9 → [illegible]

Example 2 :- ITERATE THROUGH CHARACTERS IN A STRING

```
for ch in "Python":
    print (ch)
```

output :- P
y
t
h
o
n

Example 3 :- LOOP THROUGH A LIST.

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print (fruit)
```

output : apple
banana
cherry

U15KZ25S0003

FUNCTIONS:

* Functions are the building blocks of any programming language.
* A group of instructions intended to perform a specific independent task is known as "fun".
* fun is a sub program which consists of set of instructions used to perform a specific task. A large program is divided into basic building Blocks called "fun"

Need for function:

* When the program is too complex & large they are divided into parts. Each part is separately loaded & combined into single program. each subprogram is called as "fun".
* Debugging testing & maintenance becomes easy when the program is divided into subprogram.
* Fun are used to avoid rewriting same code again & again in a program.
* Fun provides code re-usability.
* The length of the program is reduced.

* Syntax of the fun Declaration & Definition.
* A fun is defined using the def keyword, followed by a fun name & parenthesis

Syntax :- def my_function():
    print ("hello from a function")

This creates a fun named my_fun that prints "hello from a fun" when called

U15KZ25S0003

function calls

* A fun is a named sequence of instructions for performing a task.
* When we define a fun we will give a valid name to it & then specify the instructions for performing required task.
* Later, whenever we want to do that task, a function is called by its name.

Ex :- >>>type (15)
    <class int>

* here type is a fun name, 15 is the argument to a fun and <class 'int'> is the result of the fun.
* usually, a fun takes zero or more arguments & returns the result.

Built in fun:-

* Python provides a rich set of built-in fun for doing various tasks.
* The programmer/user need not know the internal working of these fun: instead, they need to know only the purpose of such fun.
* Some of built in fun are give below -
* max() : this fun is used to find maximum value among the arguments. it can be used for numeric values or even to strings.

Ex : Max (10, 20, 14, 12) # maximum of 4 integers 20

* Min () :- as the name suggests, it is used to find minimum of arguments.

Ex :- 1) min (10, 12, 14, 12) # minimum of 4 integers 10
    2) min (" hello world") # space has least ascii code here
    3) len () : this fun takes a single argument & finds its length. the argument can be a string, list, tuple etc

U15KZ22S0003

Ex :- ling ("hello how are you ?")
18

Type conversion functions :-

- As we have seen earlier (while discussing input () fun), the type of the variable / value can be converted using fun int (), float () , str()
- Python provides built-in fun that convert values from one type to another. Consider following few example.

The int fun takes any value & converts it to an integer, if it can, or complains otherwise.

- int ("20") # integer enclosed within single quote.
  20 # converted to integer type.
- int ("20") # integer enclosed within double quotes
  int ("hello") # actual string cannot be converted to int.
- Trace back (most recent call last):
- File "<pyshell # 23>", line 1, in <module> int ("hello")
- ValueError :- invalid literal for int () with base 10: 'hello'

Math Fun :-

- Python provide a rich set of mathematical fun through the module math
- To use these fun, the math module has to be imported in the code some of the important fun available in math are given here under.

IPR

U15KZ22S0003

Sqrt () :- this fun takes one numeric argument & finds the square root of that argument.
>>> math. sqrt (30) # integer argument 5.830951894845301.
>>> math. sqrt (2.5) # floating point 1.5811388300841898.

- pi :- The constant value pi can be used directly whenever we require.
  >>> print (math. pi) 3.141592653589793.

- log 10 () :- this fun is used to find to compute natural log of the given argument, to the basic base 10.
  >>> math. log10 (2) 0.3010299956639812.

- log () :- this is used to compute natural logarithm (base e) of a given number
  >>> math. log (2) 0.6931471805599453

- Sin () :- as the name suggests. It is used to find sine value of a given argument. Note that, the argument must be in radians (not degrees). One can convert the no of degrees into into radians by multiplying pi / 180 as show below.
  >>> math. sin (90 * math. pi / 180) # sin (90) => 1
  => 1.0

- Cos () :- used to find cosine value :- >>> math cos (45 * math. pi/180)
  0.7071067811865476

- tan () :- function to find tangent of an angle, given as argument
  >>> math. tan (45 * math. pi / 180) 0.9999999999999999

- Pow () :- this function takes two arguments x & y, then finds the power of y. >>> math pow (3, 4)
  81.0

U15KZ25S0003

* Python has a module random for the generation of random numbers. One has to import this module in the program. The function used is also random ():

Example -

```
import random
num = random.random()
print(num)
```

or

Example

```
import random
print(random.randrange(1,10))
```

RECURSION:- They call itself only.

* Imagine you want to know the name of a person in the queue that you are standing in. You ask people to find out about it. They keep asking the next person until they find an answer. Once an answer is found, they send it back until it reaches you.
* The above examples depict a process in which a particular action is repeated until a condition is met.
* A process in which a function calls itself is called recursion.
* This process helps ease the method of solving problems by replacing iterative code with recursive statements.

* Recursion is used in python when problems can be broken into simpler parts for easier computation & more readable code.

Types :-

1) BASE CASE :- this helps to terminate the recursive functions. It is a simple case that can be answered directly without recursion. If satisfied, it returns the final computable answer. If this is omitted, the function will run till infinity.
* Python interpreter limits the number of recursive calls for a function to 1000 by giving a recursion error.

2) GENERAL (RECURSIVE) CASE :- this case uses recursion & is called unless the base condition is satisfied.

U15KZ25S0003

Syntax :-

```
def rec_func_name():
    if (condition):          # base case
        simple statement without recursion
    else # general case
        statement calling rec_func_name()
```

Ex :-

```
def factorial(n):
    if n==0 or n==1:  # base case
        return 1
    else:
        return n*factorial(n-1)  # recursive call

num = 5
result = factorial(num)
print("Factorial of {num} is {result}")
```

Output :-
120

SCOPE: Local and Global Variables :-

* The scope defines the accessibility of the python object. To access the particular variable in the code,
* The scope must be defined as it cannot be accessed from any where in the program.
* A variable is only available from inside the region it is created. This is called scope.

Based on the scope, we can classify python variables into two types

1) Local variables
2) Global variables

1) local variable :

- When we declare variable inside a function, this variable will have a local scope (within the function) but cannot access them outside the function
- These types of variables are called local variables.

Ex : def greet ():
    # local variable
    message = 'hello'
    print ('local', message)
greet ()
try to access message variable
outside greet() function
print (message)

Output :
local hello
NameError : name 'message' is not defined.

2) Global Variable :-

- In python, a variable declared outside of the function or in global scope is known as a global variable.
- This means that a global variable can be accessed inside or outside of the function

Ex : message = 'hello'
def greet ():
    print ('local', message)
greet ()
print ('global', message)

Output :
local hello
global hello

MODULES :

- A module is a file containing python definitions, functions, statements & instructions
- Standard library of python is extended as modules
- To use these modules in a program, programmer needs to import the module.
- Once we import a module, we can reference or use to any of its functions or variables in our code.

- There is large number of standard modules also available in python
- Standard modules can be imported the same way as we import our user-defined modules
- Every module contains many functions
- To access one of the functions you have to specify the name of the module & the name of the function separated by dot. This format is called dot notation

Syntax :
import module_name
module_name.function_name (variable)

Importing Builtin Module :	Importing User Defined Module :
import math x = math.sqrt (25) print (x)	import cal x = cal.add (5, 4) print (x)