PYTHON NOTES

1. What is Python?

- **Python** is a high-level language like other high-level language such as Java, C++, PHP, Ruby, Basic and Perl.
- Python is an object-oriented programming language.
- Python provides security.
- The CPU understands a language which is called as **Machine Language**.
- Machine language is very complex and very troublesome to write because it is represented all in zero's and one's.
- The actual hardware inside CPU does not understand any of these high-level languages.

2. **Program**:

- A **Program** can be defined as a set of instructions given to a computer to achieve any objective.
- Instructions can be given to a computer by writing programs.
- Tasks can be automated by giving instructions to the computers.

3. **Defining Computer Hardware**:

Components:

- **CPU**: It helps in processing the instructions.
- **Main Memory**: It provides storage support during execution of any program in computer Eg: RAM.
- **The Secondary Memory**: It helps to store the data permanently inside the computer. Eg: Disk drives, flash memory, DVD and CD.
- **The Input and Output Devices**:
  - **Input Devices** helps users to generate any command or input any data.
  - **Output Device** helps user to get output from computer. Eg: Mouse, Printer, Keyboard, Monitor etc.

4. **Constants and Variables**:

- **Variables** can have any name, but Python reserved words cannot be used.
- A variable provides a named storage that the program can manipulate.

- Variables are named memory location used to store data in program which keeps on changing during execution.
- Programmers can decide the names of the variables.
- Fixed values used in programs such as numbers, letters and strings are called "**Constants"**.
- Values of constants never change during program execution.

5. **Variable Naming Conventions**:

- Must start with a letter or an underscore "_".
- Must consist of a letters, numbers and underscores.
- It is a case sensitive.
- Eg: First_Name, Age, Num1.
- Cannot be used: 1_hello, @hello, h123#2, -abc.
- **Note**: You cannot use reserved words for variable names and identifiers.

6. **Mnemonic Variable Names**:

- Use simple rules of variable naming and avoid reserved words.
- While using simple rules, we have a lot of choice for variable naming.

- Initially this choice can be confusing either in reading or writing the program.
- The following two programs are identical in terms of what they accomplish, but very different when you read and try to understand them:

  Eg 1:
  ```
  a=35.0
  b=12.50
  c=a*b
  print(c)
  ```
  O/P: 437.5

  Eg 2:
  ```
  hours=35.0
  rate=12.50
  pay=hours*rate
  print(pay)
  ```
  O/P: 437.5

7. **Reserved Words in Python**:

- and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is , lambda, not, or, pass, print, raise, return, try, while, with, yield.

8. **Compilers and Interpreters:**

- **Compiler** is a computer program(or a set of programs) that transforms source code written in a programming language into another computer language.
- **Interpreters** reads the source code of the program, line by line, passes the source code, and interprets the instructions.

9. **Python language**:

- The **Python language** acts as an intermediator between the end user and the programmer.
- Python script will have .py extensions.
- Every one line can be a program in Python.

10. **Types of errors**:

- **A syntax error**: It occurs when the "grammar" rules of Python are violated.
- **A logic error**: It occurs when the program has good syntax but there is a mistake in the order of the statements.

  Eg:

  - Using wrong variable name.

        - Making a mistake in a Boolean expression.
        - Indenting a block to the wrong level.
        - Using integer division instead of floating-point division.
- **A Semantic error**: It occurs when the description of the steps to take is syntactically perfect, but the program does not do what it was intended to do.

11. **Difference between Programmers and Users**:

- **Programmers** use software development tools available in a computer to develop software for the computer.
- A programmer may write the program to automate the task for himself or for any other client.
- After learning programming language, the programmer can develop the software that can be utilized by end users.
- **Users**use the tools available in a computer like word processor, spreadsheet etc., whereas **programmers** learn the computer language and develop these tools.

12. **<u>Building blocks of a Program</u>**:

These are some of the conceptual patterns that are used to construct a program:

- **<u>Input</u>**: Input will come from the user typing data on the keyboard.
- **<u>Output</u>**: Display the results of the program on a screen or store them in a file.
- **<u>Sequential Execution</u>**: Perform statements one after another in the order in which they are encountered in the script.
- **<u>Conditional Execution</u>**: Checks for certain conditions and then execute or skip a sequence of statements.
- **<u>Repeated Execution</u>**: Perform some set of statements repeatedly, usually with some variation.
- **<u>Reuse</u>**: Write a set of instructions once then reuse those instructions in the program.

13. **<u>Various Components of programming statements</u>**:

- Variable.
- Operator.
- Constant.

- Reserved Words.

14. **Operators and its Precedence**:

- **Operators** are used to manipulate the values of operands.
- There are various types of operators used in program:
    - Comparison (relational) operators.
    - Assignment operators.
    - Logical operators.

15. **Arithmetic Operators**:

- Are the symbols that are used to perform arithmetic operations on operands.

Types of Arithmetic operators:

- + ,- ,* ,/ ,%.

16. **Comparison Operators**:

- Compares the values of an operands and decide the relation among them.
- They are also called as **Relational Operators**.

Types of Comparison Operators:

- < - less than.
- >- greater than.
- <= - less than equal to.
- >= - greater than equal to.
- == - equal to.
- != - not equal to.

17. **Logical Operators**:

- Are used to evaluate expressions and return a Boolean value.

Types of Logical Operators:

- **x && y**: Performs a logical AND of the two operands.
- **x || y**: Performs a logical OR of the two operands.
- **! x**: Performs a logical NOT of the operand.

18. **Logical Operators (Contd..):**

- There are three logical operators and, or and not.

- The semantics of these operators is similar to their meaning in English.
- **Eg**: x>0 and x<10 (is true only if x is greater than 0 and less than 10).
- n %2==0 or n % 3==0(is true if either of the condition is true).
- **The not operator** negates a Boolean expression.
- **Eg**: not(x>y) is true if x>y is false.

19. **Operator Precedence**:

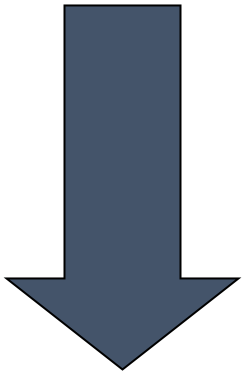
- When we use multiple operators in an expression, program must know which operator to execute first. This is called as **"Operator Precedence"**.
- The following expression multiple operators but they will execute as per precedence rule:
    - X=1+2*3-4/5**6.
- **Eg 1**: b=10, a=5, b%a O/P: 0.
- **Eg 2**: b=10, a=5, b%a==5 O/P: False.

20. **Highest Precedence rule to Lowest Precedence rule**:

- Parentheses are always respected hence given first priority.
- Exponentiation (raise to a power).

- Multiplication, Division and Remainder.
- Addition and Subtraction.
- Left to Right.

- Ie: Parentheses
  Power
  Multiplication
  Addition
  Left to Right



21. **Comments**:

- **Comments** helps in getting description about the code for future reference.
- In Python, Comment starts with # symbol.
- **Eg**: # compute the percentage of the hour that has elapsed percentage = (minute*100)/60.
- In the above case, the comment appears on a line by itself. Comments can also be put at the end of a line.
- Percentage = (minute*100)/60.
  # - Percentage of an hour.

22. **Functions**:

- In the context programming, defining a function means declaring the elements of its structure.

- The following syntax can be used to define a function:
- **Syntax**:

```
        def function_name (parameters):
function_body
return [value].
```

  - **A function** is a named sequence of statement that performs an operation.
  - After defining, the function can be executed by calling it.

23. **Built-in Functions**:

- Python provides a number of important built-in functions that can be used without needing to provide the function definition.
- abs(), divmod(), str(), sum(), super(), int(), eval(), bin(), bool(), file(), filter(), format(), type().
- **Math module**: It provides functions for specialized mathematical operations.

24. **Conditional Execution**:

- There are situations where an action performed based on a condition. This is known as **<u>“Conditional Execution”</u>**.
- The various conditional constructs are implemented using
  - If statement.
  - If else statement.
  - Chained statement.
  - Nested statement.

25. **<u>If statement</u>**:

- Contains a logical expression using which data is compared and a decision is made based on the result of comparison.
- **<u>Syntax</u>**: if condition:

```
if condition:
    action
```

26. **<u>Chained Conditions</u>**:

- **<u>Elif statement</u>**: Allows to heck multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- **<u>Nested Conditions</u>**: There may be a situation when there is need to check for another condition resolves

to true. In such a situation, the nested if construct is used.

27. **<u>Loop Pattern</u>**:

- **<u>Loops</u>** are generally used to:
  - Iterate a list of items.
  - View content of a file.
  - Find the largest and smallest data.
- There are two types of loops:
  - Infinite loops.
  - Definite loops.

28. **<u>Infinite loops</u>**:

- Sequence of instructions in a computer program which loops endlessly.
- Also known as **<u>endless loop or unproductive loop</u>**.
- Solution to an infinite loop is using break statement.

29. **<u>Break and Continue Statement</u>**:

- **The <u>break statement</u>** is used to exit from the loop.
- The break statement prevents the execution of the remaining loop.

- **The Continue Statement** is used to skip all the subsequent instructions and take the control back to the loop.

30. **For loop**:

- Used to execute a block of statements for a specific number of times.
- Used to construct a definite loop.
- **Syntax**:

```
for<destination> in <source>
            statements
print <destination>
```

31. **While loop**:

    - Is used to execute a set of instructions for a specified number of times until the condition becomes False.
    - **Syntax**:

```
while(condition)
       Executes code
    exit.
```

32. **Working of While loop**:

- Evaluate the condition, yielding True of False.

- If the condition is false, exit the while statement and continue execution at next statement.
- If the condition is true, execute the body and then go back to step 1.

33. **String**:

- **A string** is a sequence of characters.
- Single quotes or double quotes are used to represent strings.
- There are some special operators used in string.

34. **Special String Operators**:

- **Concatenation (+)**: Adds values on either side of the operator.
- **Repetition (*)**: Creates new strings, concatenating multiple copies of the same string.
- **Slice ([])**: Gives the character from the given index.
- **Range Slice ([:])**: Gives the character from the given range.
- **Membership (in)**: Returns True if a character exists in the given string.
- **Membership (not in)**: Returns True if a character does not exists in the given string.
- Some of the built-in String Methods are as follows:

- Capitalize().
- isupper().
- istitle().
- len(string).
- lower()
- lstrip().
- upper().

35. **Format Operator**:

- **“%” operator** allows to construct strings, replacing parts of the strings with the data stored in variables.
- “%” operator will work as modulus operator for strings.
- “%” operator works as a format operator if the operand is string.

36. **Exception Handling**:

- **An Exception** is an event, which occurs during the execution of a program that stops the normal flow of the program’s instructions.
- When Python script raises exception it must either handle or terminate.
- Exceptions are handled using the try and except keywords.
- **Syntax**:

```
 try
           //Code
 except Exception 1:
           //error message
 except Exception 2:
           //error message
else:
           //success message
```

37. **<u>List</u>**:

- Most versatile datatype available in python.
- Defined as a sequence of values.
- Holds values between square brackets separated by commas.
- Holds Homogenous set of items.
- Indices start at 0.
- **<u>Lists</u>** are Mutable.

38. **<u>List Functions</u>**:

- **<u>sum( )</u>**: Using this function, we can add elements of the list. It will work only with the numbers.
- **<u>min( )</u>**: Using this function, we can find the minimum value from the list.

- **max( )**: Using this function, we can find the maximum value from the list.
- **len( )**: Using this function, we can find the number of elements in the list.

39. **Dictionary**:

- It is a “bag” of values, each with its own label.
- It contains the values in the form of key-value pair.
- Every value in Dictionary is associated with a key.

40. **Characteristics of Dictionary**:

- **Dictionaries** are Python’s most powerful data collection.
- Dictionaries allows us to do fast database-like operations in Python.
- Dictionaries have different names in different languages.
    - Associative Arrays- Perl/PHP.
    - Properties or Map or HashMap- Java.
    - Property Bag- C#/ .NET.
- Dictionary Keys can be of any Python data type. Because keys are used for indexing, they should be immutable.

- Dictionary Values can be of any Python data type. Values can be mutable or immutable.

41. **Difference between Lists and Dictionary**:

LIST	DICTIONARY
List is a list of values. Starting from zero.	It is an index of words and each of them has a definition.
We can add the element index wise.	Element can be added in Dictionary in the form of key-value pair.

42. **Tuples**:

- **A Tuple** is an immutable List.
- A Tuple stores values, similar to a List, but uses different syntax.
- A Tuple cannot be changed once it is created.
- Tuple uses parentheses, whereas lists use square brackets.
- A Tuples is a sequence of immutable Python objects.

43. **Features of Tuples**:

- **Tuples** are more efficient.
- Tuples are faster than Lists.

- Tuples are converted into Lists, and vice-versa.
- Tuples are used in String Formatting.
- Note: We cannot perform Add, Delete and Search operations on Tuples.

44. **Operations which can be performed on Tuples**:

- You can’t add elements on tuple. Tuples have no append or extend method.
- You can’t remove elements from a tuple. Tuples have no remove or pop method.
- You can’t find elements in a tuple. Tuples have no index method.
- You can however, check if an element exist in the tuple.

45. **Creating Tuples**:

- **Creating a Tuple** is as simple assigning values separated with commas.
- Optionally, you can put these comma-separated values between parentheses also.
- Eg: # Zero-element tuple.

```
   a= ( )
 # One-element tuple.
   b= (“one”,)
```

# Two-element tuple.
c= (“one”,”two”)

46. **Updating Tuples**:

- **Tuples** are immutable.
- An immutable object cannot be changed once it is created it always remain the same.
- The values of Tuple element cannot be updated or changed.
- Portions of existing tuples can be used to create new tuples.

47. **Deleting Tuples**:

- Removing individual Tuple elements is not possible.
- “del” statement is used to remove an entire Tuples.
- It is possible to merge two Tuples.
- Tuples can be reassigned with different values.