Python

Unit:1

1. Introduction
2. Variables And Expression
3. Conditional Statements, Looping, Control Statements

Chapter1: Introduction

- Introduction: The Python Programming Language
- Features
- Debugging
  - Syntax errors
  - Runtime errors
  - Semantic errors
  - Experimental debugging
- Difference between brackets, Braces and Parentheses

Introduction

The Python Programming Language

- Python is an object oriented, high level language, interpreted, dynamic and multipurpose programming language.
- Python is a multipurpose language due to its ease of learning, cross-platform compatibility, vast libraries, and community support, making it suitable for diverse applications like web development, data science, and more.
- In python you don't need to declare data types when defining variables (It's dynamically typed).
- Python's absence of compilation accelerates development and debugging.

Features of Programming

- ### Easy to code:
    - Python's simplicity and readability facilitate rapid learning, making it ideal for beginners.
    - As compared to other OOP like Java, C, C++, and C#.
- ### Open source and free:
    - Python is an open source ad freely available programming language, fostering collaboration and cost-effectiveness in development.
- ### Support for GUI:
    - Python offers support for creating GUI through libraries like Tkinter, making desktop application development feasible.
- ### Object-Oriented approach:
    - Python supports an OOP approach, promoting code modularity, reusability, and a structured design.
- ### Highly portable:
    - Python is highly portable, allowing code to run on various platforms without significant modification, enhancing cross compatiability.

- **Highly dynamic:**
    - Due to its dynamic typing, enabling variables to change types during runtime of flexibility.
- **Large standard library:**
    - Python includes a large standard library with built-in modules and packages for diverse tasks, reducing the need for custom code.

Debugging

- Debugging is the process of ensuring program health by identifying and resolving issues in code execution, offering complete control over program flow.
- Python provides a built in **"pdb" (python debugger)** module for debugging, allowing developers to set breakpoints, inspect variables, and control the program's execution.
- Debugging helps keep software bug free, ensuring its reliability and correctness by systematically addressing defects.
- The **"pdb"(python debugger)** module is a valuable tool for Python developers, enabling precise debugging without the need for external tools.

Syntax error

- Occurs when we do not use properly defined syntax in any programming language.
    - **Example-** Incorrect arguments, Indentation, Use of undefined variables etc.
    - **Example-**

```
1  age = 10
2  if(age>=18):
3  print("You can vote.")          #Here you will get a syntax error(Indentation)
4  else:
5  print("You can not vote.")      #Here you will get a syntax error(Indentation)
```

Runtime error

- This error doesn't appear until after the program has started running. Also known as exceptions.
- Some examples of python runtime errors:
    - Division by zero.
    - Performing an operation on incompatible types.
    - Using an identifier which hasn't been defined.

- Accessing a list element, dictionary value or object attribute which doesn't exist.
- Trying to access a file which doesn't exist.

Semantic error

- Your program will run without error but will not show the desired output. That means you wrote the program that you don't want to write.

Experimental debugging

- Debugging is an essential, intellectually challenging aspect of programming. mIt resembles experimental science, interactive problem-solving, and gradual refinement to achieve desired program behavior.

Difference b/w Brackets, Braces & Parentheses

Brackets [ ]:

- Define list or list comprehension.
- Used for indexing and lookup of elements.
- **Ex-**

```
1  L1=[10,20,30]              #Making a list
2  print(L1)
3
4  print(L1[1])               #Printing the 1 index element
```

Braces { }:

- Define dictionary or set.
- **Ex-**

```
s1 = {10,20,30}                          #Making set
print(s1)

d1 = {'rollno':13, 'name':'Mayank'}      #Making dictionary
print(d1['rollno'])
```

Parentheses ( ):

- Create immutable sequence data type called a tuple.

```
t1 = (10,20,30,40)                  #For creating tuple
print(t1)
```

- To define the paremeter of function definition and function call.

```
def mul(a,b):                    #Making a function for multiplication
    return a*b
print(mul(5,3))                  #Calling this function with paremeters
```

Chapter2:

VARIABLES AND EXPRESSION

- Values and Types
  - Variables
  - Keywords
- Type conversion
  - Implicit Type Conversion
  - Explicit Type Conversion
- Operators and Operands
- Expressions
- Interactive Mode and Script Mode
- Order of Operations

Values and Types

Data content held by an object, defining its essential information.
These are some of the common value types in Python:

1. **Integers (int):** 130201
2. **Float (float):** 13.09
3. **Strings (str):** “Hello I am Mayank”
4. **Lists:** [1,2,3]
5. **Dictionaries:** {‘name’:’Mayank’,’age’:19}
6. **Booleans (bool):** True or False
7. **None:** A special value representing the absence of a value.

```
integer = 13
FloatingPoint = 13.09
Strings = "Hello I am Mayank"
Lists = [1,2,3,4]
Dictionaries = {'name':'Mayank','age':19}
Booleans = True
x = None
```

Variables

- A variable is a named storage location for holding data values during program execution in memory.
- You can assign value using assignment operator (=)
- Variable name start with (a-z, A-Z) or (_) not with numbers, and are case sensitive.

```
var = 1
A = 'Mayank'
b = 'Dewali'
x_1 = 'hello'
```

- Can hold values of various data types, such as integers, floats, lists, dictionaries and more.
- You can reassignment the values

```
x = 79
x = "Mayank"
```

- Don’t need to specify the data type.

```
a = 13                  #a is an int
a = "Mayank"            #a is now string
```

Keywords

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

Type Conversion

Type conversion, also known as type casting, refers to the process of converting a value from one data type to another in a programming language. In Python, there are built-in functions that allow you to perform type conversion easily.
Python has two types of type conversion.

- **Implicit Type Conversion**
- **Explicit Type Conversion**

Implicit Type Comversion

Automatic conversion of one data type to another by the interpreter, usually during operations involving different types. For example, adding an integer to a float results in implicit conversion of the integer to a float.

```
a = 15          #This is an Integer
b = 13.9        #This is a floating number
add = a + b     #Implicit conversion of 15 to 15.00
```

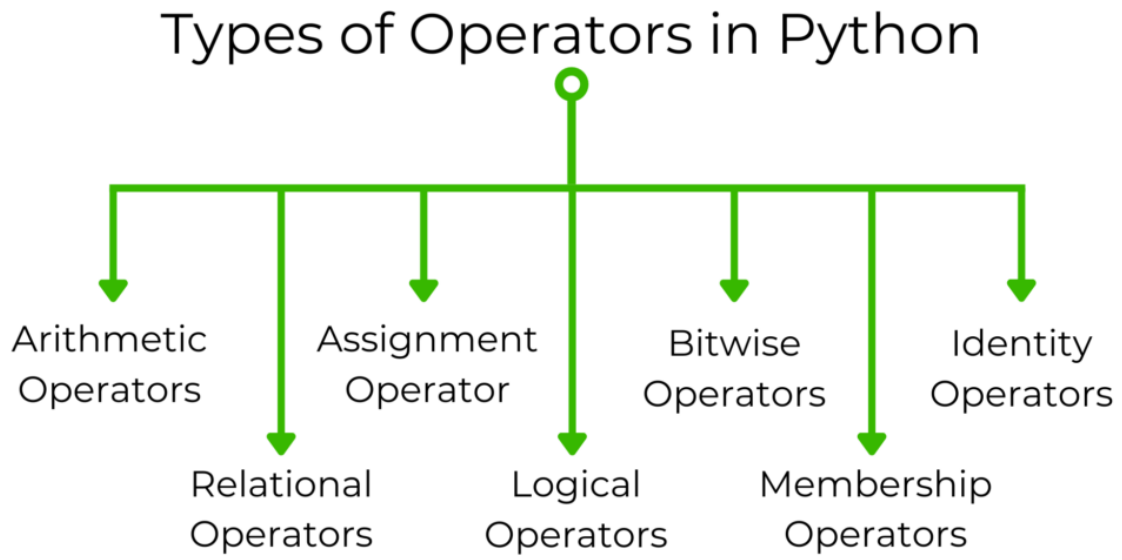
Explicit Type Conversion

Manual conversion of a value from one type to another using predefined functions. This is also known as casting.

```
a = 5                   # a is an integer
n = float(a)            # converting a(integer) to float
```

Operators and Operands

Operators are symbols or keywords in programming that perform operations on operands. Operands are values or variables the operators act upon. For example, in `5 + 3`, `+` is the operator, and `5`and `3` are operands. Operators enable computations, comparisons, and logical operations, enhancing code functionality.



Arithmetic Operators

Operators	Description
//	Perform Floor division (gives integer value after division)
+	To perform addition
-	To perform subtraction
*	To perform multiplication
/	To perform division
%	To return remainder after division (Modulus)
**	Perform exponent (raise to power)

Relational Operators

Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Assignment Operators

Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
**=	m**=2	m = m**2 or $m = m^2$	225
//=	m//=10	m = m//10	1

Logical Operators

Operators	Description
and	Logical AND (When both conditions are true output will be true)
or	Logical OR (If any one condition is true output will be true
not	Logical NOT (Compliment the condition i.e., reverse)

Bitwise Operators

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR / Bitwise XOR
~	Bitwise inversion (one's complement)
<<	Shifts the bits to left / Bitwise Left Shift
>>	Shifts the bits to right / Bitwise Right Shift

Membership Operators

Operators	**Description**
in	Returns true if a variable is in sequence of another variable, else false.
not in	Returns true if a variable is not in sequence of another variable, else false.

Identity Operators

Operators	**Description**
is	Returns true if identity of two operands are same, else false
is not	Returns true if identity of two operands are not same, else false.

Expressions

In Python, expressions are combinations of values, variables, and operators that, when evaluated, result in a single value. Examples include mathematical calculations (`**2 + 3**`), comparisons (`**x > 10**`), or string concatenation (`**"Hello" + "World"**`). Expressions are fundamental for dynamic and versatile programming.

Interactive Mode and Script Mode

Interactive Mode

- Interactive Mode allows users to enter Python commands directly into the interpreter, and the interpreter immediately executes and displays the results.
- Open a terminal or command prompt and type `python` to enter interactive mode. Useful for testing code snippets, trying out commands, and quick experimentation.

```
>>> print("Hello, World!")
Hello, World!
```

Script Mode

- Script Mode involves creating a script, which is a file containing Python code, and then executing the entire script. Script files typically have a **`.py`** extension.
- Write Python code in a file (e.g.,**`script.py`**) using a text editor, and then run the script using the command python script.py in the terminal or command prompt.

```
# script.py
print("Hello, World!")
```

Order of Operations

The order of operations, also known as the **precedence of operators**, is a set of rules that dictate the sequence in which different operations are evaluated in a mathematical expression. In Python, as in many programming languages, the order of operations follows the standard mathematical conventions.
The order of operations is often remembered using the acronym **PEMDAS**:

P - Parentheses:

Operations inside parentheses are performed first.

```
result = (2 + 3) * 4  # Parentheses take precedence: 5 * 4 = 20
```

E - Exponents (or Powers):

Exponentiation is the next operation to be performed.

```
result = 2 ** 3  # Exponentiation: 2 to the power of 3 = 8
```

M/D - Multiplication and Division:

Multiplication and division have the same precedence and are performed from left to right.

```
result = 10 * 3 / 2  # Multiplication and division: (10 * 3) / 2 = 15.0
```

A/S - Addition and Subtraction:

Addition and subtraction have the same precedence and are performed from left to right.

```
result = 5 + 7 - 3  # Addition and subtraction: (5 + 7) - 3 = 9
```

These rules ensure that expressions are evaluated in a logical and predictable manner. However, parentheses can be used to override the default order and explicitly specify the order of evaluation.

```
result = (2 + 3) * (4 / 2)  # Parentheses take precedence: 5 * 2 = 10
```