Day 2

INTRODUCTION TO PYTHON

Programming Language is a form of communication that is used to instruct computer to perform some specific things. Example: Addition of two numbers.

Python is a high level; interpreter based programming language which can be used in multiple fields like Web Development, Artificial Intelligence, Networking, etc. It was created by Guido van Rossum, and released in 1991.

FEATURES OF PYTHON

1. Free and Open Source
2. Easy to Read and Code
3. Object-Oriented and Procedure-Oriented Language
4. Dynamically Typed Language
5. Easy to Debug
6. Large Standard Library
7. Interpreted Language and many more.

DIFFERENCE BETWEEN RUN TIME AND COMPILE TIME.

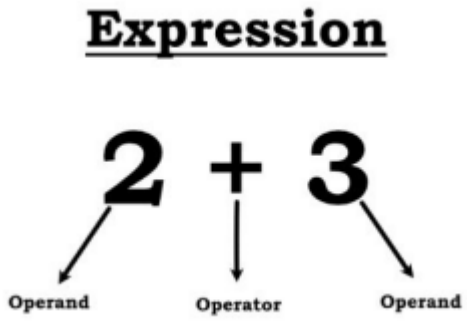
Run Time	**Compile Time**
a. Runtime is the time at which the executable code is started running.	a. Compile time is the time at which source code is converted to executable code
b. Runtime errors can be: ○ Division by zero ○ Square root of negative numbers,etc	b. Compile time errors can be: ○ Syntax errors ○ Semantic errors

DIFFERENCE BETWEEN INTERPRETER AND COMPILER.

Interpreter	**Compiler**
Interpreter is a program that converts source code to machine code line by line when program is executed.	Compiler is a program that converts source code to machine code in one go and generate executable file that can be run directly by the computer
At each execution, interpreter convert each line of user program to machine code, the process is slower	Compiler generates executable file, so it is faster to run compiled code than interpreted code.
Example: Python, Ruby, etc	Example: C, C++, Java etc

OPERATORS, VARIABLES AND KEYWORDS IN PYTHON

Operators v/s Operands v/s Expressions



- Operators are special symbols that perform specific operations on one or more operands.
- Operands are the values that an operator acts on.
- A sequence (or combinations) of operands and operators, is called an expression.

TYPES OF OPERATOR

1. **Arithmetic Operators**
   General mathematical symbols used for addition, subtraction, multiplication etc.

Operator	Name	Example
+	Addition	2 + 3
-	Subtraction	5 - 2
*	Multiplication	4 * 2
/	Division	10 / 2
%	Modulus	7 % 3
**	Exponentiation	2 ** 3
//	Floor division	10 // 3

2. **Assignment operators**
   Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3

```
In [22]: x = 5

In [23]: print(x)
5

In [24]: x += 5

In [25]: print(x)
10

In [26]: x = x + 5

In [27]: print(x)
15
```

(Note: Variables are the container where we can store certain data value.)

3. **Comparison Operators**
   Comparison operators are used to compare two values (operands or variables).

Operator	Name	Example
==	Equal	7 == 7
!=	Not equal	10 != 3
>	Greater than	8 > 4
<	Less than	2 < 9
>=	Greater than or equal to	6 >= 6
<=	Less than or equal to	3 <= 2

4. **Logical Operators**
   Logical operators are used to combine conditional expressions.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

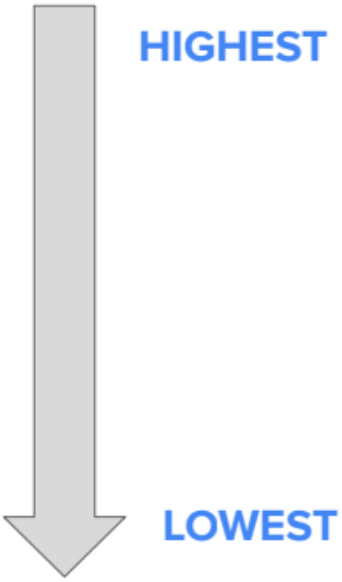
5. **Identity Operators**
   Identity operators are used to compare variables.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

OPERATORS PRECEDENCE

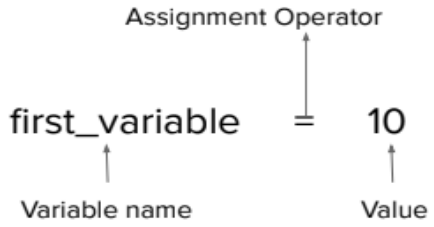
This is used in an expression with more than one operator with different precedence to determine which operation to perform first.

Precedence	Operators	Description	Associativity
1	() [], {}	Parentheses, Brackets, Braces	Left-to-right
2	**	Exponentiation	Right-to-left
3	+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT	Right-to-left
4	*, /, //, %	Multiplication, Division, Modulo	Left-to-right
5	+, -	Addition, Subtraction	Left-to-right
6	<<, >>	Bitwise shift left, Bitwise shift right	Left-to-right
7	&	Bitwise AND	Left-to-right
8	^	Bitwise XOR	Left-to-right
9	|	Bitwise OR	Left-to-right
10	<, <=, >, >=	Comparison operators	Left-to-right
11	==, !=	Equality operators	Left-to-right
12	is, is not, in, not in	Identity and membership operators	Left-to-right
13	not	Logical NOT	Right-to-left
14	and	Logical AND	Left-to-right
15	or	Logical OR	Left-to-right
16	=, +=, -=, *=, /=, //=, %=, &=, ^=, |=, <<=, >>=	Assignment operators	Right-to-left



PYTHON VARIABLES

Variables are containers for storing data values. When you assign a value to a variable, Python reserves a space in memory to store that value.



RULES FOR CREATING VARIABLES

1. A variable name must start with a letter or the underscore character
   - True: (e.g., var_name, _variable)
   - False: (e.g., 123var, @variable)
2. A variable name cannot start with a number
   - True: (e.g., variable1, _variable2)
   - False: (e.g., 5number, 10th_variable)
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
   - True: (e.g., my_variable, var123)
   - False: (e.g., my_var$, var!able)
4. Variable names are case-sensitive (var, Var, VAR are three different variables)
5. A variable name cannot be any of the Python Keywords (error)
   - False: (e.g., for, if)

NAMING CONVENTIONS

1. Camel Case: Each word except the first; starts with a capital letter.

    camelCaseDemo = "camel case"

2. Pascal Case: Each word starts with a capital letter

    PascalCaseDemo = "pascal case"

3. Snake Case: Each word is separated by underscore and is lowered cased. (preferred in python)

    snake_case_demo = "snake case"

KEYWORDS AND IDENTIFIERS.

Keywords:

Keywords are the reserved words in Python. We cannot use keywords as variable name, function name, or any other identifier. Keywords are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive. For e.g False is keyword but false is not.

```
In [5]: help('keywords')

Here is a list of the Python keywords.  Enter any keyword to get more help.

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
break               for                 not
```