UNIT-I

Introduction

- Python is a **general-purpose**, **dynamically typed**, **high-level**, **compiled and interpreted**, **garbage-collected**, and purely **object-oriented** programming language that supports **procedural, object-oriented**, and **functional programming**.
- It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation.
- Python versions: **Python 2 and Python 3**.

Features of Python

1. Easy to use and Read
2. dynamically typed
3. High-level
4. Compiled and Interpreted
5. Garbage Collected
6. Purely Object-Oriented
7. Cross-platform Compatibility
8. Rich Standard Library
9. Free and Open Source
10. Robust
11. Embeddable

Applications

1. **Web Development**

- **Frameworks:** Django, Flask, Pyramid
- **Applications:** Web applications, server-side scripting, API development

2. **Data Science and Machine Learning**

- **Libraries:** Pandas, NumPy, SciPy, scikit-learn, TensorFlow, PyTorch, Keras
- **Applications:** Data analysis, data visualization, predictive modeling, machine learning algorithms, deep learning

3. **Automation and Scripting**

- **Tools:** Selenium, Beautiful Soup, Scrapy, PyAutoGUI
- **Applications:** Web scraping, task automation, process automation, scripting for system administration

4. **Software Development and Prototyping**

- **Tools:** PyQt, Tkinter, Kivy
- **Applications:** Rapid prototyping, GUI application development, software testing

5. **Scientific Computing**

- **Libraries:** SciPy, NumPy, Matplotlib, SymPy
- **Applications:** Scientific research, simulations, mathematical modeling, computational physics

6. **Game Development**

- **Libraries:** Pygame, Panda3D
- **Applications:** 2D and 3D game development, game prototyping, educational games

7. **Cybersecurity**

- **Tools:** Scapy, Nmap, PyCrypto
- **Applications:** Penetration testing, network scanning, cryptographic applications, malware analysis

8. **Embedded Systems**

- **Tools:** MicroPython, CircuitPython
- **Applications:** IoT devices, hardware interfacing, robotics

9. **Education**

- **Applications:** Teaching programming concepts, coding bootcamps, introductory programming courses

10. **Financial Analysis**

- **Libraries:** QuantLib, Pandas
- **Applications:** Quantitative finance, algorithmic trading, financial modeling

11. **Natural Language Processing (NLP)**

- **Libraries:** NLTK, SpaCy, Gensim
- **Applications:** Text processing, sentiment analysis, language translation, chatbot development

12. **Artificial Intelligence (AI)**

- **Libraries:** TensorFlow, PyTorch, Keras
- **Applications:** AI research, neural networks, reinforcement learning, computer vision

13. **Desktop Applications**

- **Libraries:** PyQt, Kivy, wxPython
- **Applications:** Cross-platform desktop applications, productivity tools, multimedia applications

INSTALLATION

1. **Windows**

a. **Download the Installer**

- Go to the official Python website. (https://www.python.org/downloads/)
- Download the latest stable release of Python for Windows.

b. **Run the Installer**

- Run the downloaded installer.
- Make sure to check the box that says **"Add Python to PATH"**.
- Choose **"Install Now"** for a quick installation, or **"Customize Installation"** if you need to configure the installation options.

c. **Verify the Installation**

- Open Command Prompt (type cmd in the search bar and press Enter).
- Type **python --version** and press Enter.

2. Ubuntu

a. Using a Package Manager

Most Linux distributions come with Python pre-installed.

If you need to install a specific version, you can use a package manager.

- sudo apt update
- sudo apt install python3

b. Verify the Installation

- Open Terminal.
- Type **python3 --version** and press Enter.

Keywords

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

```
import keyword
print(keyword.kwlist)
```

Brief Explanation of Some Keywords:

1. **False, True, None:** Boolean values and the null value.
2. **and, or, not:** Logical operators.
3. **if, elif, else:** Conditional statements.
4. **for, while, break, continue:** Looping constructs.
5. **def, return, yield:** Function definition and return/yield values.
6. **class:** Class definition.
7. **try, except, finally, raise:** Exception handling.
8. **import, from, as:** Module importation.
9. **global, nonlocal:** Variable scope control.
10. **with:** Context management.

Data types

1. Text Type: str
2. Numeric Types: int, float, complex
3. Sequence Types: list, tuple, range
4. Mapping Type: dict
5. Set Types: set, frozenset
6. Boolean Type: bool
7. Binary Types: bytes, bytearray, memoryview
8. None Type: NoneType

- To check the data type of any object **type()** is used.

EXAMPLES

Example	Data Type
`x = "Hello World"`	str
`x = 100`	int
`x = 3.14`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	Dict
`x = {"apple", "banana", "cherry"}`	Set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset

Example	Data Type
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	Memoryview
`x = None`	NoneType

Operators

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Arithmetic Operators

➢Arithmetic operators are used with numeric values to perform common mathematical operations.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x – y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

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
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Operator	Example	Same As
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Comparison Operators

- Comparison operators are used to compare two values.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

- Logical operators are used to combine conditional statements.

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)

Identity Operators

- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

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

Membership Operators

- Membership operators are used to test if a sequence is presented in an object.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise Operators

- Bitwise operators are used to compare (binary)

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Input and Output

- With the print() function, display output in various formats, while the input() function enables interaction with users by gathering input during program execution.
- to find and print the address of the Python variable use id() function.

Output

- print() function allows to display text, variables, and expressions on the console.

Example:

print(“Hello World!”)

- assigns values to variables name and age, then prints them with labels.

Example:

name = “Pritesh"

age = 30

print("Name:", name, "Age:", age)

- Output formatting in Python with various techniques including the format() method, manipulation of the sep and end parameters, f-strings, and the versatile % operator.
- These methods enable precise control over how data is displayed, enhancing the readability and effectiveness of Python programs.

1. Using Format()

Example:

amount = 150.75

print("Amount: ${**:.2f**}".format(amount)) → Amount: $150.75

2. Using sep and end parameter

Example:

print("Python", end='@') → Python@

print('G', 'F', 'G', sep='') → GFG

print('10', '07', '2024', sep='-') →10-07-2024

3. Using f-string

Example:

name = “Pritesh”

age = 20

print(f"Hello, My name is {name} and I'm {age} years old.")

→Hello, My name is Pritesh and I'm 20 years old.

4. Using % Operator

- use **‘%’** operator. % values are replaced with zero or more value of elements.

- The formatting using % is similar to that of 'printf' in the C programming language.

1. %d –integer
2. %f – float
3. %s – string
4. %x –hexadecimal
5. %o – octal

Example:

num = 20

add = num + 10

print("The sum is **%d**" %add) →The sum is 30

Input

- For taking input from the user use input() function.

Example:

```
name=input(“Enter the name:”)
print(name)
```

- input from the user in a single line, splitting the values entered by the user into separate variables for each value using the split() method. Then, it prints the values with corresponding labels, either two or three, based on the number of inputs provided by the user.

Example:

```
x, y = input("Enter two values: ").split()
print("Number one: ", x)
print("Number two: ", y)
```

Indentation in Python

- Whitespace is used for **indentation in Python**.
- Unlike many other programming languages which only serve to make the code easier to read, **Python indentation** is mandatory.
- One can understand it better by looking at an example of indentation in Python.

Comments in Python

- Python comments start with the hash symbol # and continue to the end of the line.
- Comments in Python are useful information that the developers provide to make the reader understand the source code.
- It explains the logic or a part of it used in the code.
- Comments in Python are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it.
- These are often cited as useful programming convention that does not take part in the output of the program but improves the readability of the whole program.

Types of comments in Python

1. **Single-line comment :-**Python single-line comment starts with a hash symbol (#) with no white spaces and lasts till the end of the line.

1. **Multiline comment**

- Use a hash **(#)** for each extra line to create a multiline comment.
- In fact, Python multiline comments are not supported by Python's syntax, use Python multi-line comments by using multiline strings.
- It is a piece of text enclosed in a delimiter (“”””) on each end of the comment.
- Again there should be no white space between delimiter (“”””).
- They are useful when the comment text does not fit into one line; therefore need to span across lines.
- Python Multi-line comments or paragraphs serve as documentation for others reading your code.

Python block

- In Python, a block is a group of statements that are indented together.
- Blocks are used to define the scope of variables and to control the flow of execution.
- The statements in a block are executed as a unit.
- This means that if one statement in the block raises an exception, the remaining statements in the block will not be executed.
- Blocks are also used to define the scope of variables.
- The scope of a variable is the part of the program where the variable is accessible.
- The scope of a variable in a block is limited to the block itself.
- This means that a variable defined in a block cannot be accessed outside of the block.

some of the rules for Python block syntax:

1. Blocks must be indented by four spaces.
2. Statements in a block must be indented by the same amount of space.
3. The scope of a variable in a block is limited to the block itself.

EXAMPLE

```
j = 1
while(j <= 5):          #python block start
    print(j)             statment1
    j = j + 1            statment2
                         #python block end
```