Python Notes

- Nitish Kumar Singh

✅ UNIT 1 – Introduction to Python Programming

📌 1.1 What is Python?

Python is a high-level, interpreted programming language known for its easy-to-read syntax and vast standard libraries.

📌 1.2 Evolution & History

Python is one of the most widely used programming languages today, known for its simplicity, versatility, and powerful features. Its development and evolution over the years are both fascinating and influential in the world of technology. Here's a detailed look at Python's history and evolution:

1980s: The Beginnings

- **Inspiration:** In the late 1980s, Guido van Rossum, a Dutch programmer, began working on Python while at Centrum Wiskunde & Informatica (CWI) in the Netherlands.
- **Motivation:** Guido wanted to create a language that combined the best features of scripting languages (like Perl and Shell scripting) with the strengths of more traditional programming languages (like C).
- **Predecessor:** Python was influenced by the ABC programming language, also developed at CWI. While ABC was powerful, it had some shortcomings that Guido wanted to address.

1989: Python is Born

- **Christmas Holiday Project:** During the Christmas holidays of 1989, Guido started implementing Python. He named the language not after the snake but after the British comedy show *Monty Python's Flying Circus*, reflecting his love for humor.

1991: Python 0.9.0

- The first public release of Python occurred in February 1991.
- **Features of 0.9.0:**
    - Core programming constructs: Functions, exception handling, and core data types like lists, strings, and dictionaries.
    - The inclusion of modules, which allowed code reuse.
- This was a significant step, marking Python as a general-purpose programming language.

1994: Python 1.0

- Python 1.0 was officially released in January 1994.
- **Key Features of Python 1.0:**

    - Introduction of key constructs such as lambda, map(), filter(), and reduce().
    - Basic support for object-oriented programming.
    - Widespread adoption began as developers appreciated Python's simplicity and readability.

2000: Python 2.0

- **Major Milestone:** Python 2.0 was released in October 2000.
- **Notable Features:**
    - List comprehensions for concise and readable code.
    - Garbage collection using reference counting and cycle-detecting.
    - However, Python 2.x was criticized for its backward-incompatible changes and eventually faced the challenge of being replaced.
- **Legacy Issue:** The Python 2.x series lasted for 20 years but was officially retired on January 1, 2020.

2008: Python 3.0

- **The Big Leap:** Python 3.0 (released in December 2008) was designed to fix the flaws of Python 2 and improve consistency.
- **Notable Features:**
    - Better Unicode support (everything is Unicode by default).
    - Simplified syntax (e.g., print() became a function).
    - Removal of redundant constructs and libraries.
- Python 3 introduced breaking changes, making it incompatible with Python 2, but it positioned Python for future growth.

2010s: The Growth Era

- Python became one of the most popular languages during this period due to:
    - The rise of data science, machine learning, and artificial intelligence (AI), where Python's libraries like NumPy, pandas, and TensorFlow thrived.
    - Python's adaptability for web development, with frameworks like Django and Flask.
    - Its use in education as an introductory programming language.
- Community support and an extensive library ecosystem accelerated its adoption.

2020 and Beyond: Modern Python

- **Python 3 Adoption:** By 2020, Python 3 had become the standard, and Python 2 was officially sunset.
- **Key Developments:**
    - Introduction of async and await for asynchronous programming.
    - Enhanced type hints and type checking (typing module).
    - Performance improvements and new syntactic features (e.g., assignment expressions with the := operator).
- Python's use continues to expand in domains such as:
    - Cloud computing (AWS Lambda, Google Cloud Functions).
    - Scientific research (SciPy, Matplotlib).
    - DevOps and automation (Ansible, SaltStack).

Why Python Became Popular

1. **Readability:** Emphasis on simplicity and human-readable syntax.
2. **Versatility:** Suitable for web development, data analysis, AI, automation, and more.
3. **Community:** A vast and active developer community.
4. **Libraries and Frameworks:** Extensive standard and third-party libraries for virtually any task.

The Future of Python

Python continues to evolve, with regular updates introducing new features and optimizations. As of 2023, Python remains a top choice for:

- AI and machine learning.
- Cybersecurity and ethical hacking.
- Game development.
- Quantum computing research.

📌 1.3 Features of Python

- Interpreted
- High-level
- Dynamically typed
- Object-oriented
- Portable and platform-independent
- Extensive standard libraries

📌 1.4 Advantages

- Easy to learn & write
- Large community
- Open-source
- Rich libraries
- Cross-platform

📌 1.5 Disadvantages

- Slower than compiled languages
- Not ideal for mobile app development
- Runtime errors due to dynamic typing

📌 1.6 Interpreted or Compiled?

Python is an interpreted language. The source code is first converted to bytecode (.pyc) and then executed by the Python Virtual Machine (PVM).

The main difference between an **interpreter** and a **compiler** lies in how they execute source code written in a programming language. Below is a detailed comparison:

Aspect	Interpreter	Compiler
Definition	Translates and executes code line-by-line or statement-by-statement.	Translates the entire source code into machine code before execution.
Execution Speed	Slower, as it processes code one line at a time during runtime.	Faster, as the entire code is translated into machine language beforehand.
Translation Process	Performs both translation and execution simultaneously.	Performs translation first (into machine code or an intermediate form) and execution later.
Error Detection	Detects and reports errors **immediately** during execution, stopping when an error is found.	Detects and reports errors **after compilation**, meaning all errors are found before execution.
Output	Does not generate a separate executable file; executes directly.	Generates a standalone executable file (e.g., .exe, .out).
Languages	Commonly used for scripting and dynamic languages (e.g., Python, JavaScript, Ruby).	Commonly used for languages designed for system-level or performance-oriented programming (e.g., C, C++).
Reusability	Code must be re-interpreted every time it is run.	Once compiled, the executable can be run multiple times without recompiling.
Debugging	Easier for beginners, as errors are shown immediately.	Requires fixing all errors during compilation before running the program.
Portability	Code is usually platform-independent but requires the interpreter installed on the target system.	Compiled programs may need to be recompiled for different platforms, depending on the target system.

Examples

Interpreter-Based Languages	Compiler-Based Languages
Python	C
JavaScript	C++

Interpreter-Based Languages	Compiler-Based Languages
Ruby	Java (compiled to bytecode)
PHP	Go
Bash	Fortran

Key Points

- **Flexibility**: Interpreters are more flexible for prototyping and dynamic coding.
- **Performance**: Compilers produce faster executables, making them ideal for performance-critical applications.
- **Error Handling**: Interpreters provide immediate feedback, while compilers catch all errors upfront.

Both interpreters and compilers have their advantages and are suited to different use cases depending on the requirements of the project.

📌 1.7 Statically or Dynamically Typed?

Python is dynamically typed—variables are declared without specifying types.

Example: -

```
x = 5       # integer
x = "hello"   # now a string
```

📌 1.8 Python vs Other Languages

Feature	Python	C++	Java
Syntax	Simple & concise	Complex	Verbose
Typing	Dynamic	Static	Static
Compilation	Interpreted	Compiled	Compiled
Usage	AI, Web, Data	Games, Systems	Enterprise Apps

📌 1.9 Setting up Python Environment

- Install Python from python.org
- Use IDEs: VS Code, PyCharm, Jupyter

- Verify with: python --version or python3 –version

✅ UNIT 2 – Basic Programming Concepts

📌 2.1 Data Types

- int, float, complex, str, bool
- list, tuple, set, dict

📌 2.2 Variables

No type declaration needed:

```
x = 10
name = "Nitish"
```

📌 2.3 Operators

- Arithmetic: + - * / % // **
- Comparison: ==, !=, < >, <=, >=
- Logical: and or not
- Assignment: = += -=
- Membership: in, not in
- Identity: is, is not

📌 2.4 Conditional Statements

```
x = 20
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

📌 2.5 Loops

- **for loop:**

```
for i in range(1, 6):
    print(i)
```

- **while loop:**

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

✅ Practice Programs (From your request)

- Factorial using loops
- Fibonacci sequence
- Armstrong number
- Palindrome check
- Reverse a number

✅ UNIT 3 – Data Structures & Functions

📌 3.1 List

Mutable collection:

```
lst = [1, 2, 3]
lst.append(4)
print(lst[1])  # Output: 2
```

Built-in Functions: append(), pop(), insert(), remove(), reverse(), sort()

📌 3.2 Tuple

- Tuple is immutable data structure which store data in sequential manner just like a list to create a tuple we use rounded bracket ().
- Immutable collection:
    - tpl = (1, 2, 3)

- Built-in Functions: count(), index()

📌 3.3 Set

- It is a collection of unique data. To create a set, we use curly braces the elements and items present in sets can be different types of the sets.
- Unordered, unique elements:
    - s = {1, 2, 3, 2}
    - print(s) # Output: {1, 2, 3}
- Functions: add(), remove(), union(), intersection()

📌 3.4 Dictionary

- Dictionary is a set of unordered key value pairs. In a dictionary the keys must be unique.
- To create a dictionary, we use {} curly bracket, separated by a ‘ , ’ comma. The keys used must be immutable.

- Syntax: -
    - dict_name = {___}
    - dict_name = dict ()
    - dict_name = {1: “Java”, 2: “Python”, 3: “ReactJs”}

- Functions: keys(), values(), items(), get(), update()

- Example: -
    - d = {'name': 'Nitish', 'age': 20}
    - print(d['name'])

📌 3.5 Strings

s = "Python"

print(s.upper(), s[0:3])

Functions: upper(), lower(), replace(), split(), find()

📌 3.6 Functions

```
def add(a, b):
    return a + b


print(add(2, 3))
```

✅ UNIT 4 – Modules, Exception, File Handling, OOP

📌 4.1 Modules

It is a file that contains code to perform a specific task. A module may contain variable function, classes etc. They are two types: -

- Standard Module or Build in Module
- User-defined module.

We use import keyword to include the modules. To access built-in function or use defined function in a module. We use dot operator for example: -

- import math
  print(math.pi)

We can also rename a module by using as example: -

- import math as m
  print(m.pi)

To import a specific function from a module, we can write, example: -

- import math as m
- print(m.pi)
- from math import pi

📌 4.2 User-defined Module

```
# mymodule.py

def greet():
    print("Hello!")


# main.py

from mymodule import greet

greet()
```

📌 4.3 Exception Handling

```
try:
    a = 10 / 0
```

except ZeroDivisionError:

    print("Cannot divide by zero!")

📌 4.4 File Handling

File handling refers to the ability of a program to read from and write to files on the disk. Python provides built-in functions and methods to handle files and perform operations like:

- Creating
- Reading
- Writing
- Appending
- Closing
- Deleting files

✅ File Types in Python

1. **Text files (.txt)** – Contains readable characters.
2. **Binary files (.bin, .jpg, .exe)** – Contains non-text data like images, videos, etc.

✅ Opening a File

```
file_object = open("filename", "mode")
```

✅ Closing a File

Always close the file to free system resources:

```
file_object.close()
```

✅ Reading from a File

1. **read()** – Reads entire file content as a string.

```
f = open("demo.txt", "r")
content = f.read()
print(content)
f.close()
```

2. **readline()** – Reads one line at a time.

```
line = f.readline()
```

3. **readlines()** – Returns a list of all lines.

```
lines = f.readlines()
```

✅ **Writing to a File**

1. **write()** – Writes a single string.

```
f = open("demo.txt", "w")
f.write("Hello Python!")
f.close()
```

2. **writelines()** – Writes multiple strings.

```
f.writelines(["Line1\n", "Line2\n"])
```

✅ **Deleting a File**

```
import os
os.remove("demo.txt")
```

📌 4.5 OOP

- **Class & Object**

```
class Student:
    def __init__(self, name):
        self.name = name

s = Student("Nitish")
print(s.name)
```

- Inheritance

```
class A:
    def display(self):
        print("Class A")

class B(A):
```

```
    pass


b = B()

b.display()
```

✅ UNIT 5 – Applications of Python

📌 5.1 Pandas

- Pandas is a python library used for data manipulation and analysis. It uses through data structure called as Series and Dataframe.
- Panda library is generally used for data cleaning, data transformation, data analysis, machine learning, data visualization, etc.
- To use panda, we need to install in the system if it is not present. The command is used is: - **pip install pandas**.
- In order to use library after installing it we use **import panda**.
- **Panda series: -**
    - It is one dimensional labelled array like object that can hold data of any type.
    - Panda series is considered as a column in a spread sheet or a single column of a data frame.
    - There are two components **labels** and **data.**
        - Labels are the index value assigned to each data point and 'Data' represents the actual values.
        - **Data** – the actual values.
- **Example**: -
    - import pandas as pd

      ```
      data = [10, 20, 30, 40]
      s = pd.Series(data)
      print(s)
      ```

📌 5.2 NumPy

- **NumPy** is a powerful library in Python used for numerical computations and handling large multi-dimensional arrays and matrices.
- It provides a large collection of high-level mathematical functions to operate on arrays efficiently.

🔹 **Key Features:**

- Efficient storage and manipulation of numerical arrays.

- Functions for performing element-wise operations, linear algebra, statistical operations, and more.
- Broadcasting: Ability to perform arithmetic operations on arrays of different shapes.

🔹 **Installation:**

- pip install numpy

🔹 **Importing NumPy:**

- import numpy as np

🔹 **Core Concepts:**

✅ **ndarray (N-dimensional array):**

- The primary data structure in NumPy.
- Supports multi-dimensional arrays.

```
arr = np.array([1, 2, 3])
```

✅ **Array Creation Functions:**

- np.zeros(), np.ones(), np.arange(), np.linspace(), np.eye() etc.

✅ **Array Attributes:**

- .shape, .ndim, .dtype, .size, .itemsize

✅ **Mathematical Operations:**

```
arr1 + arr2     # Element-wise addition

np.dot(a, b)    # Matrix multiplication

np.mean(), np.sum(), np.max(), np.min(), np.std() # Statistical operations
```

✅ **Indexing and Slicing:**

- Supports slicing like lists, and advanced indexing.

✅ **Reshaping:**

```
arr.reshape(3, 2)
```

✅ **Universal Functions (ufuncs):**

- Fast vectorized operations like np.sqrt(), np.exp(), np.log() etc.

📌 5.3 Matplotlib

- **Matplotlib** is a popular Python library for creating static, interactive, and animated visualizations.
- The most used module of Matplotlib is pyplot.

- **Installation: -** pip install matplotlib
- **Importing: -** import matplotlib.pyplot as plt
- **Basic Plotting:**

```
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]


plt.plot(x, y)
plt.title("Simple Line Graph")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

- **Common Plot Types:**
  - ✅ **Line Chart** – plt.plot()
  - ✅ **Bar Chart** – plt.bar()
  - ✅ **Histogram** – plt.hist()
  - ✅ **Scatter Plot** – plt.scatter()
  - ✅ **Pie Chart** – plt.pie()
- **Customization:**
  - Titles, labels, legends, grids, colors, line styles, markers.

```
plt.grid(True)
plt.legend(["Sales"])
plt.xlim(0, 10)
```

- **Subplots:**

```
plt.subplot(1, 2, 1)
plt.plot(x, y)
plt.subplot(1, 2, 2)
plt.bar(x, y)
plt.show()
```

📌 5.4 Web Scraping with BeautifulSoup

```
import requests
```

```
from bs4 import BeautifulSoup


url = 'https://example.com'
res = requests.get(url)
soup = BeautifulSoup(res.text, 'html.parser')
print(soup.title.text)
```