ADVANCED PYTHON PROGRAMMING FOR DATA SCIENCE

-Dhurba Subedi
dsubedi164@gmail.com

Syllabus of Adv.Python

Course Objectives:

The objective of this course is to develop advanced proficiency in Python programming for data science applications. It focuses on efficient coding practices, sophisticated data manipulation, statistical analysis, and data visualization using modern Python libraries. Students will also learn fundamental data engineering and pipeline design concepts to automate and scale real-world data workflows.

Syllabus

1. Advanced Python Concepts and Best Practices 7 Hours
    1.1. Review of Python essentials and coding conventions
    1.2. Advanced data structures: Collections, iterators, generators, and decorators
    1.3. Functions and lambda expressions
    1.4. Object-Oriented Programming for data science applications
    1.5. Exception handling, debugging, and logging
    1.6. Working with modules and packages
2. Data Sources and APIs 7 Hours
    2.1. Reading and writing structured/unstructured data (CSV, JSON, Excel, text)
    2.2. Database access with relational database and non-relational database
    2.3. Accessing and processing data from APIs (REST, SOAP)
    2.4. Web scraping using requests and BeautifulSoup
    2.5. Handling large datasets with chunking and lazy evaluation

Syllabus

3. Advanced Data Wrangling and Transformation 9 Hours
    - 3.1. Advanced Pandas operations: Merging, joining, reshaping, pivoting
    - 3.2. Handling missing, categorical, and time-series data
    - 3.3. Feature transformation, scaling, and encoding
    - 3.4. Memory optimization and efficient data processing
    - 3.5. Building a reusable data-cleaning pipeline
    - 3.6. Introduction to data pipeline components (Ingestion, transformation, storage)
4. Applied Statistics and Exploratory Analysis 7 Hours
    - 4.1. Statistical measures: Correlation, covariance, skewness, kurtosis
    - 4.2. Probability review, sampling, and hypothesis testing
    - 4.3. Regression and trend analysis using stats models
    - 4.4. Exploratory data analysis (EDA) using descriptive and inferential methods
    - 4.5. Automation of EDA workflows using Python

Syllabus

5. Data Visualization and Storytelling 7 Hours
    - 5.1. Principles of effective visualization and dashboard design
    - 5.2. Visualization with Matplotlib: Line, bar, histogram, scatter, subplots
    - 5.3. Seaborn for statistical visualization: Box plot, pair plot, heat map
    - 5.4. Interactive visualization using Plotly
    - 5.5. Visualization driven insight generation
    - 5.6. Case study: End-to-end visualization and reporting project
6. Data Engineering and Automation 8 Hours
    - 6.1. Overview of data engineering in applied data science
    - 6.2. Designing and implementing ETL pipelines
    - 6.3. Automating workflows with schedulers (CRON, schedule)
    - 6.4. Logging, monitoring, and error handling in pipelines
    - 6.5. Data storage and retrieval strategies for pipelines
    - 6.6. Automated report generation (Excel, HTML, PDF)
    - 6.7. Case study: End-to-end automated analytics pipeline

Syllabus

Laboratory Works:

1. Setting up Python environment for applied data workflows and writing modular programs using OOP and functions
2. Collecting data via APIs and web scraping
3. Building advanced data cleaning and transformation pipelines using Pandas
4. Conducting exploratory data analysis and statistical summaries
5. Developing interactive visualizations using Plotly, matplotlib and Seaborn
6. Automating ETL tasks and data refresh using Python schedulers
7. Generating summary dashboards and automated analytical reports
8. Mini Project: Build a complete applied data pipeline from ingestion to visualization and reporting on a real-world dataset

Syllabus

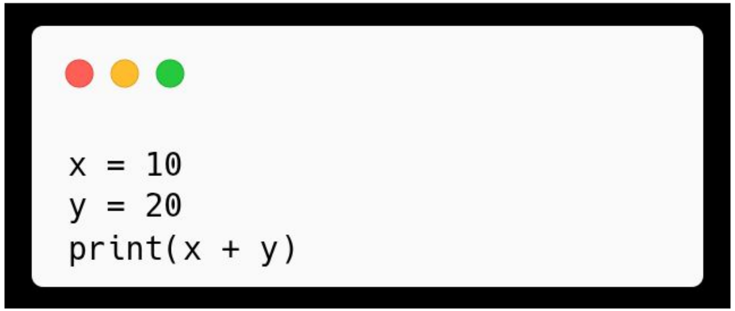
Reference

- McKinney, W. (2022). Python for data analysis. O’Reilly Media.
- VanderPlas, J. (2016). Python data science handbook. O’Reilly Media.
- Beazley, D. (2021). Python cookbook. O’Reilly Media.

1: Advanced Python Concepts and Best Practices

Introduction to Python

- Python is a high-level, interpreted, general-purpose programming language designed with an emphasis on code readability and simplicity.
- Key Features:
  - Interpreted (no separate compilation)
  - Dynamically typed
  - Object-oriented and multi-paradigm
  - Automatic memory management
  - Platform independent
  - Extensive standard library



```
x = 10
y = 20
print(x + y)
```

PEP 8 : Python Style Guide

- ❖ PEP 8 stands for Python Enhancement Proposal 8.
- ❖ It is the official style guide for writing clean, readable, and consistent Python code.
- ❖ PEP 8 is a recommended guideline rather than a strict rule, but it is widely followed in the Python community.
- ❖ Its purpose is to improve code readability, maintainability, and team collaboration.

Rules for PEP 8

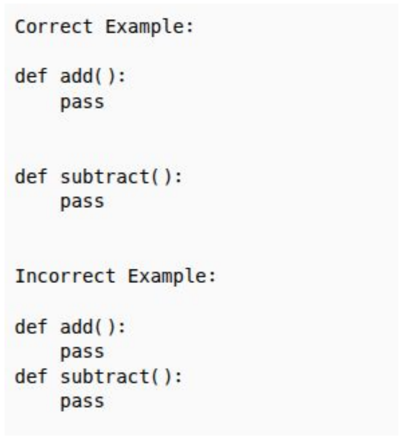
1. Indentation Rule:
   - PEP 8 specifies that Python code must use 4 spaces per indentation level.
   - Tabs should not be used for indentation because they can cause inconsistent formatting across different editors.
   - Proper indentation is essential in Python as it defines the block structure of the program.
2. Line Length Rule:
   - According to PEP 8, each line of Python code should be limited to 79 characters, while comments and docstrings should be limited to 72 characters.
   - This rule ensures that code remains readable on smaller screens and when printed.
   - Long lines should be broken using parentheses, brackets, or line continuation techniques.

```
Correct Example:

if marks >= 40:
    print("Pass")


Incorrect Example:

if marks >= 40:
  print("Pass")
```

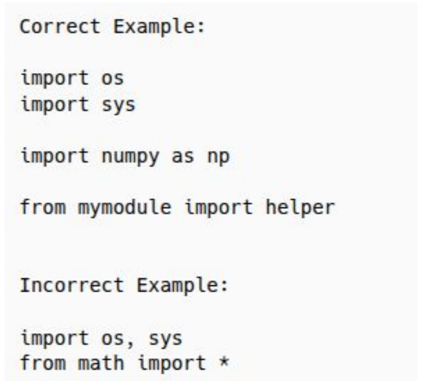
```
Correct Example:

total = (maths + science +
         english + nepali)


Incorrect Example:

total = maths + science + english + nepali + computer + social
```

Rules for PEP 8

3. Blank Line Rule:
    - PEP 8 recommends using two blank lines between top-level function definitions and class definitions.
    - Inside a class, one blank line should be used to separate methods.
    - Blank lines help visually organize code and make it easier to understand the structure of a program.
4. Import Rule:
    - All import statements should be placed at the top of the file, after module comments and docstrings.
    - Imports must follow a specific order: first standard library modules, then third-party modules, and finally local application modules.
    - Each import should be written on a separate line, and wildcard imports should be avoided to maintain code clarity.



```
Correct Example:

def add():
    pass


def subtract():
    pass


Incorrect Example:

def add():
    pass
def subtract():
    pass
```



```
Correct Example:

import os
import sys

import numpy as np

from mymodule import helper


Incorrect Example:

import os, sys
from math import *
```

Rules for PEP 8

5. **Naming Convention Rule**:
    - PEP 8 defines consistent naming conventions to improve code readability.
    - Variable and function names should use lowercase letters with underscores, class names should follow the CamelCase format, and constants should be written in uppercase letters.
    - Private members should begin with a single underscore.
6. **Whitespace Usage Rule**:
    - PEP 8 emphasizes proper use of whitespace to enhance readability.
    - Spaces should be placed around operators and after commas, but extra spaces inside brackets or parentheses should be avoided.
    - Trailing whitespace at the end of lines should be removed, as it can cause unnecessary formatting issues.
7. **Comment Rule**
    - Comments should be written as complete sentences that clearly explain the purpose of the code.
    - They should begin with a capital letter and have a space after the hash (#) symbol.
    - PEP 8 advises that comments should explain why a piece of code is written, rather than what it does, and unnecessary comments should be avoided.

Rules for PEP 8

```
Naming Convention Rule:
Correct Example:

total_marks = 450

def calculate_average():
    pass

class StudentRecord:
    pass

PI = 3.14159


Incorrect Example:

TotalMarks = 450
def CalculateAverage():
    pass
```

```
Whitespace Usage Rule:
Correct Example:

x = a + b
my_list = [1, 2, 3]


Incorrect Example:

x=a+b
my_list = [1,2,3]
```

```
Comment Rule:
Correct Example:

# Calculate total marks
total_marks = math_marks + science_mark


Incorrect Example:

#add a and b
total = a + b
```

Rules for PEP 8

8. Boolean Comparison Rule:
    - PEP 8 discourages comparing boolean values explicitly with True or False.
    - Instead, boolean variables should be used directly in conditions.
9. None Comparison Rule:
    - When checking for None, PEP 8 recommends using is or is not instead of equality operators.
    - This ensures that identity comparison is performed correctly and avoids logical errors.
10. Return Statement Rule:
    - PEP 8 advises avoiding unnecessary else blocks after a return statement.
    - Once a return statement is executed, the function exits, making the else block redundant.

Rules for PEP 8

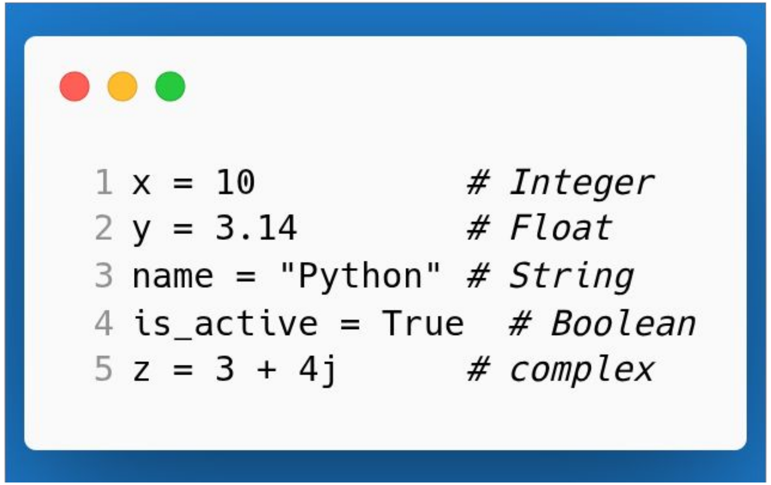
```
Boolean Comparison Rule:
Correct Example:

if is_active:
    print("Active")


Incorrect Example:

if is_active == True:
    print("Active")
```

```
None Comparison Rule:
Correct Example:

if value is None:
    print("No value")


Incorrect Example:

if value == None:
    print("No value")
```

```
Return Statement Rule:
Correct Example:

def check(x):
    if x > 0:
        return True
    return False


Incorrect Example:

def check(x):
    if x > 0:
        return True
    else:
        return False
```

Variables and Data Types in Python

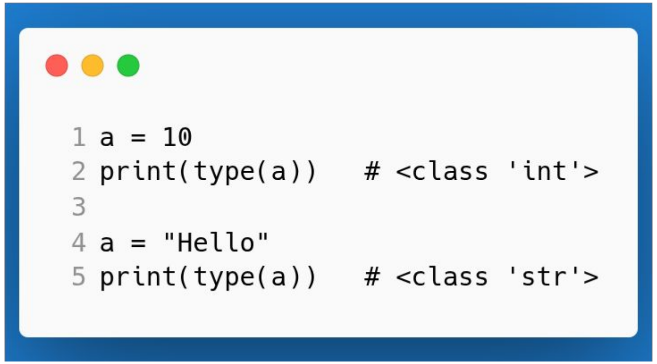
- ❖ Python is a dynamically typed language, which means you do not need to declare the data type of a variable explicitly.
- ❖ The interpreter automatically determines the type at runtime based on the value assigned.



```
x = 10               # Integer
y = 3.14             # Float
name = "Python"  # String
is_active = True   # Boolean
z = 3 + 4j           # complex
```

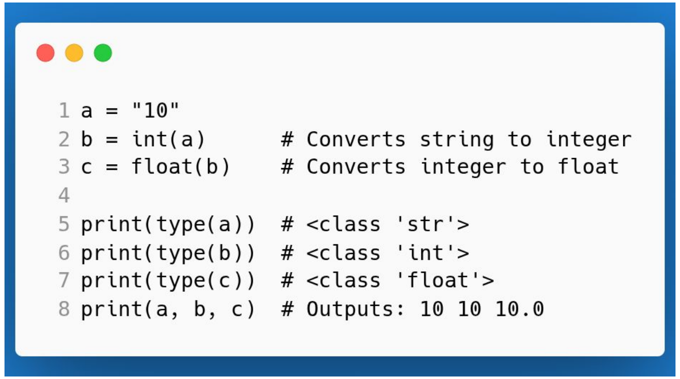
Variables and Data Types in Python

- Dynamic Typing (No Explicit Declaration):
  - A variable can change its data type during execution.



```python
a = 10
print(type(a))    # <class 'int'>

a = "Hello"
print(type(a))    # <class 'str'>
```

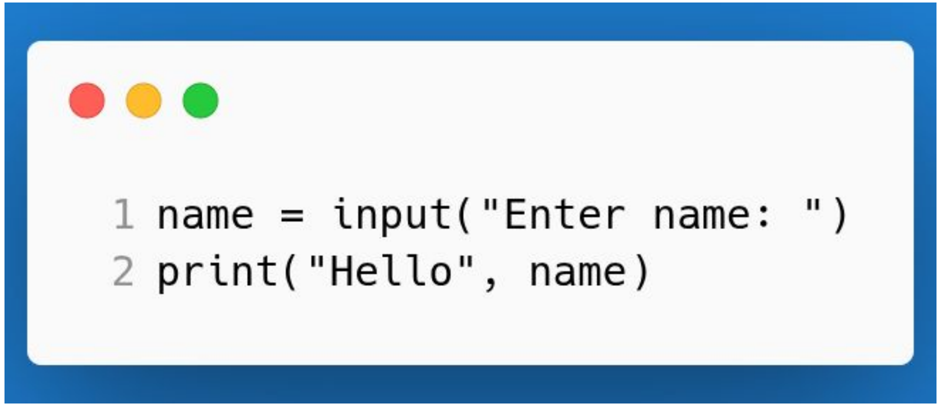
- Type Conversion (Type Casting)
  - Even though Python is dynamically typed, you can explicitly convert types.



```python
a = "10"
b = int(a)       # Converts string to integer
c = float(b)     # Converts integer to float

print(type(a))  # <class 'str'>
print(type(b))  # <class 'int'>
print(type(c))  # <class 'float'>
print(a, b, c)  # Outputs: 10 10 10.0
```

Input and Output

- ❖ Python provides built-in functions for taking input from the user and displaying output.
- ❖ input() takes user input as string.
- ❖ print() displays output.



```python
name = input("Enter name: ")
print("Hello", name)
```

Operators in Python

- Operators are symbols or keywords used to perform operations on variables and values.
- Types:
  - Arithmetic:
    - Used to perform mathematical calculations.
    - Eg: +, -, *, /, %, //, **
  - Relational:
    - Used to compare two values. Result is True or False.
    - Eg: ==, !=, >, <, >=, <=
  - Logical:
    - Used to combine conditional statements.
    - Eg: and, or, not
  - Assignment:
    - Used to assign and update values.
    - Eg: =, +=, -=, *=
  - Membership:
    - Used to test if a value is present in a sequence.
    - Eg: in, not in
  - Identity:
    - Used to compare memory locations, not values.
    - Eg: is, is not

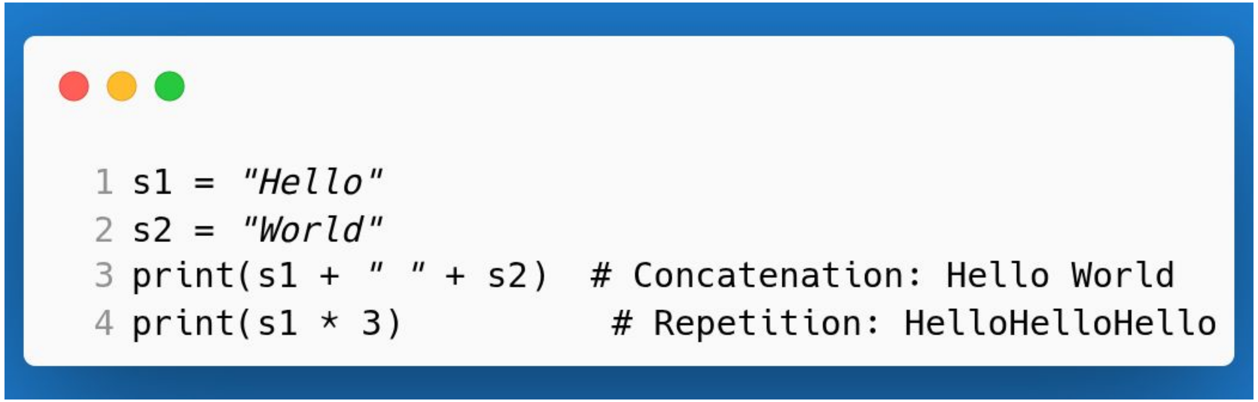
Strings in Python

- ❖ A string is a sequence of characters enclosed in single ', double ", or triple quotes ''' / """.
- ❖ Strings in Python are immutable, meaning you cannot change a string in place.

Method	Description	Example
`lower()`	Converts string to lowercase	`"Python".lower() → "python"`
`upper()`	Converts string to uppercase	`"Python".upper() → "PYTHON"`
`strip()`	Removes leading/trailing whitespaces	`" hello ".strip() → "hello"`
`split()`	Splits string into list of substrings	`"a,b,c".split(',') → ['a', 'b', 'c']`
`replace()`	Replaces substring with another	`"Python".replace('Py', 'Ja') → "Jathon"`

Strings in Python

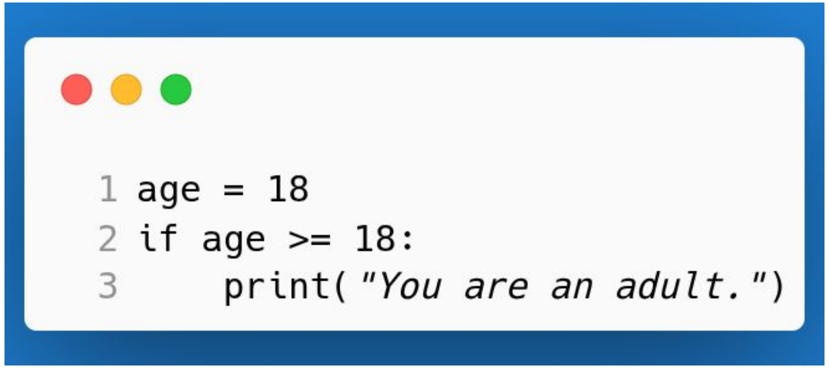
❖ String Concatenation and Repetition:



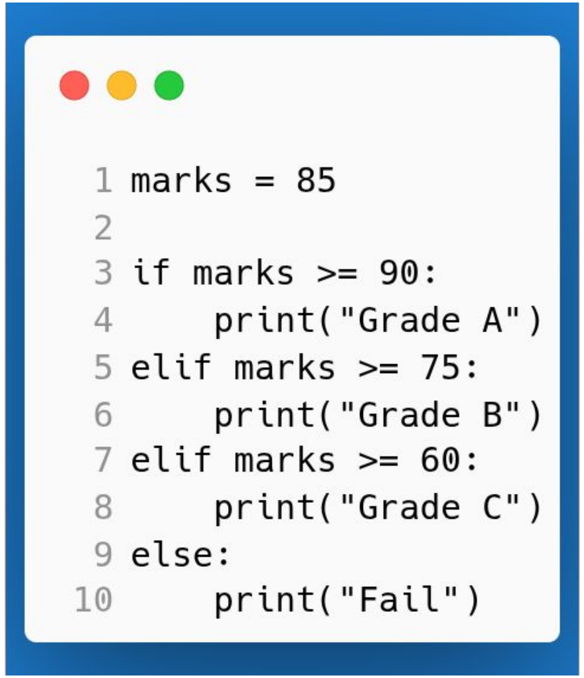
```python
s1 = "Hello"
s2 = "World"
print(s1 + " " + s2)  # Concatenation: Hello World
print(s1 * 3)          # Repetition: HelloHelloHello
```

Conditional Statements in Python

- ❖ Conditional statements are used to execute different blocks of code based on a condition.
- ❖ Python uses if, elif, and else for decision-making.

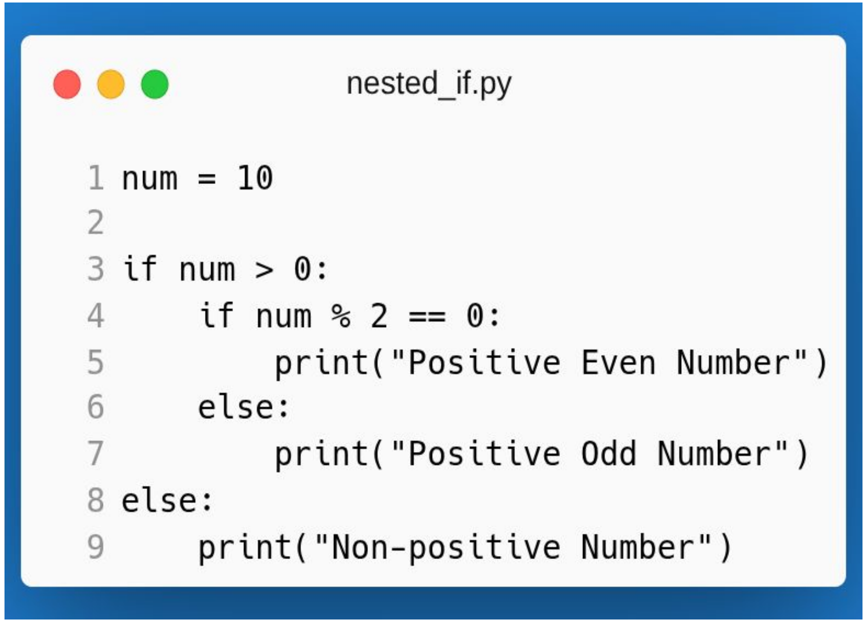


```python
age = 18
if age >= 18:
    print("You are an adult.")
```

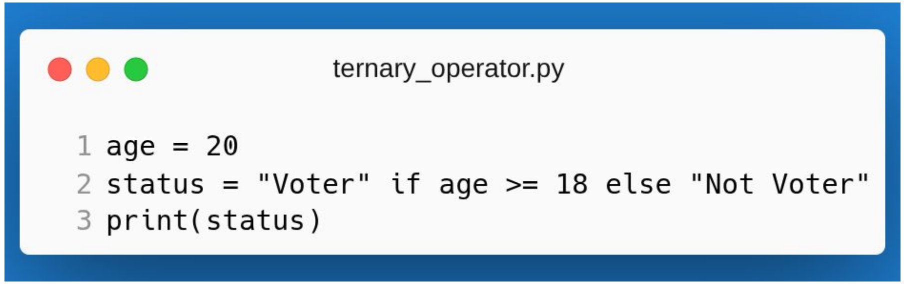


```python
marks = 85

if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
elif marks >= 60:
    print("Grade C")
else:
    print("Fail")
```

Conditional Statements in Python



```python
num = 10

if num > 0:
    if num % 2 == 0:
        print("Positive Even Number")
    else:
        print("Positive Odd Number")
else:
    print("Non-positive Number")
```



```python
age = 20
status = "Voter" if age >= 18 else "Not Voter"
print(status)
```

End of Lecture !