# WARM UP-Algorithm and Python

TA: Dongchen(Felix) HE

Contact: dc23.he@gmail.com

# Content

1. Introduce Yourself

2. Course Syllabus

3. Class & Homework Requirement

4. TA Session & Office Hour

5. Preview Materials

## 1. Introduction Part

Firsly, I'll give you guys a brief introduction of myself 🫣

Then, it's your show time 🥳

# Who I am

Dongchen HE, or you can just call Felix.

Bachelor degree of EE in Nanjing university

Ph.D. student in the Department of Computer Science & Engineering in CUHK

My research interests foucus on NLP(Have you guys ever heard of ChatGPT?) and AI4Healthcare(like AlphaFold).

## Most importantly: I'm your TA of the course Algorithm and Python.

# Who you are

## Feel free to give us a self-introduction~ 👍

| Name | School & Grade & Major | Current City | Programming Experience (Python) (Research Experience in this Course) | Desired Field of Study & Time | What I Hope to Gain from This Course |
|---|---|---|---|---|---|

## 2. Course Syllabus

- Specific course schedules for Chinese professors and foreign professors, please refer to the course syllabus sent by Remi.
- Our regular classes are held on Saturday and Sunday evenings.
- TA classes are held on Friday evenings.

# Course Summary

| Lecture | Objective | Description |
|---|---|---|
| 1 | Introduction to Python; Recurrent algorithms | - Implementation in Python of the naïve sorting algorithm and the merge sort algorithm. |
|  |  | - Complexity analysis of the naïve sorting algorithm and the merge sort algorithm. |
| 2 | Linear-time selection; Breadth-first and depth-first search | - Introduction to graphs, cuts, and the contraction algorithm. |
|  |  | - Random Contraction Algorithm |

# 3. Class & Homework Requirements

## Class & Homework Requirements

- During class:
  - Please ensure your camera is turned on.
  - Actively participate in class discussions and activities.
  - Attend classes punctually and regularly.
- For homework assignments:
  - Complete assignments on time.
  - The requirement for homework naming is: `HW1_FirstName_LastName_EnglishName` .

# 4. TA Session & Office Hour

- During TA sessions:
    - The TA will review the content covered in each class.
    - They will answer any questions or doubts that students may have.
- Outside of TA sessions:
    - If you have any questions or concerns, please summarize them beforehand.
    - Schedule a meeting during the designated office hours with the TA for further assistance in advance (Two days or longer).

# 5. Preview Materials

## History of Python

- Created by Guido van Rossum in 1989
- Named after the comedy group Monty Python
- First released in 1991, version 0.9.0
- Python 2.0 released in 2000
- Python 3.0 released in 2008 with major changes
- Python 3.x series is the current version

# Creation of Python

- Guido van Rossum
  - Creator of Python
  - Started development in 1989
  - Guided Python's design and development for over 30 years
- Design Principles
  - Readability
  - Simplicity
  - Code expressiveness
  - Emphasis on community and collaboration

- Python Enhancement Proposals (PEPs)
  - Formal process for proposing and discussing language improvements
  - Allows community involvement in shaping Python's future
- Python Software Foundation (PSF)
  - Non-profit organization
  - Supports Python's development and community
  - Manages resources and initiatives
- Python's Evolution
  - Python 2.x vs. Python 3.x
  - Python 2's end of life (EOL) on January 1, 2020
  - Transitioning to Python 3 for ongoing development

# Why Python is Important

- **Simplicity**: Python's easy-to-read syntax makes it accessible for beginners and experienced programmers alike. 🐍
- **Versatility**: Python is used in web development, data analysis, machine learning, scientific computing, and automation. 🌐📊🤖

- **Rich Ecosystem**: Python has a vast collection of libraries like NumPy, Pandas, and TensorFlow, expanding its functionality. 📚
- **Strong Community**: Python has a large and supportive community providing resources and assistance. 👥💪
- **Cross-Platform**: Python runs on Windows, macOS, and Linux, allowing code to be written once and run anywhere. 🖥️
- **Integration**: Python seamlessly integrates with other languages, leveraging existing codebases. 🔄

python
powered

- **Scalability**: Python handles both small and large-scale projects efficiently. ⚙️
- **Career Opportunities**: Python's popularity and versatility create abundant job prospects. 💼

# How to Learn Python

1. **Set Clear Goals**: Define what you want to achieve with Python. Whether it's web development, data analysis, or machine learning, having clear goals helps guide your learning path.
2. **Start with the Basics**: Begin by understanding the fundamentals of Python, including variables, data types, control flow, and functions. Online tutorials and beginner-friendly resources can help you grasp the basics.
3. **Practice Coding**: The more you code, the better you become. Practice writing Python code regularly to reinforce your understanding and gain hands-on experience.

4. **Utilize Online Resources**: Take advantage of the abundant online resources available. Websites like Codecademy, Coursera, and edX offer Python courses and tutorials for all skill levels.
5. **Work on Projects**: Apply your Python skills by working on small projects. It helps you consolidate your knowledge and gives you practical experience.
6. **Read Python Documentation**: Familiarize yourself with the official Python documentation. It serves as a comprehensive resource and reference for the language and its standard library.
7. **Explore Python Libraries**: Python has a vast ecosystem of libraries. Research and explore popular libraries like NumPy, Pandas, and Matplotlib, depending on your areas of interest.

8. **Keep Learning**: Python is a versatile language with ever-evolving features. Stay curious, continue learning, and stay updated with the latest developments in the Python community.

## Remember, learning Python is a journey, so be patient, persistent, and enjoy the process!

python
powered

# Basic Python

```python
# Hello, World!
print("Hello, World!")
```

```python
# Variables and Data Types
name = "Alice"
age = 25
is_student = True
```

```
# Arithmetic Operations
x = 10
y = 5
addition = x + y
subtraction = x - y
multiplication = x * y
division = x / y
```

```python
# Conditional Statements
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

```
# Loops
for i in range(5):
    print(i)
```

```
# Functions
def greet(name):
    print("Hello, " + name + "!")

greet("Bob")
```

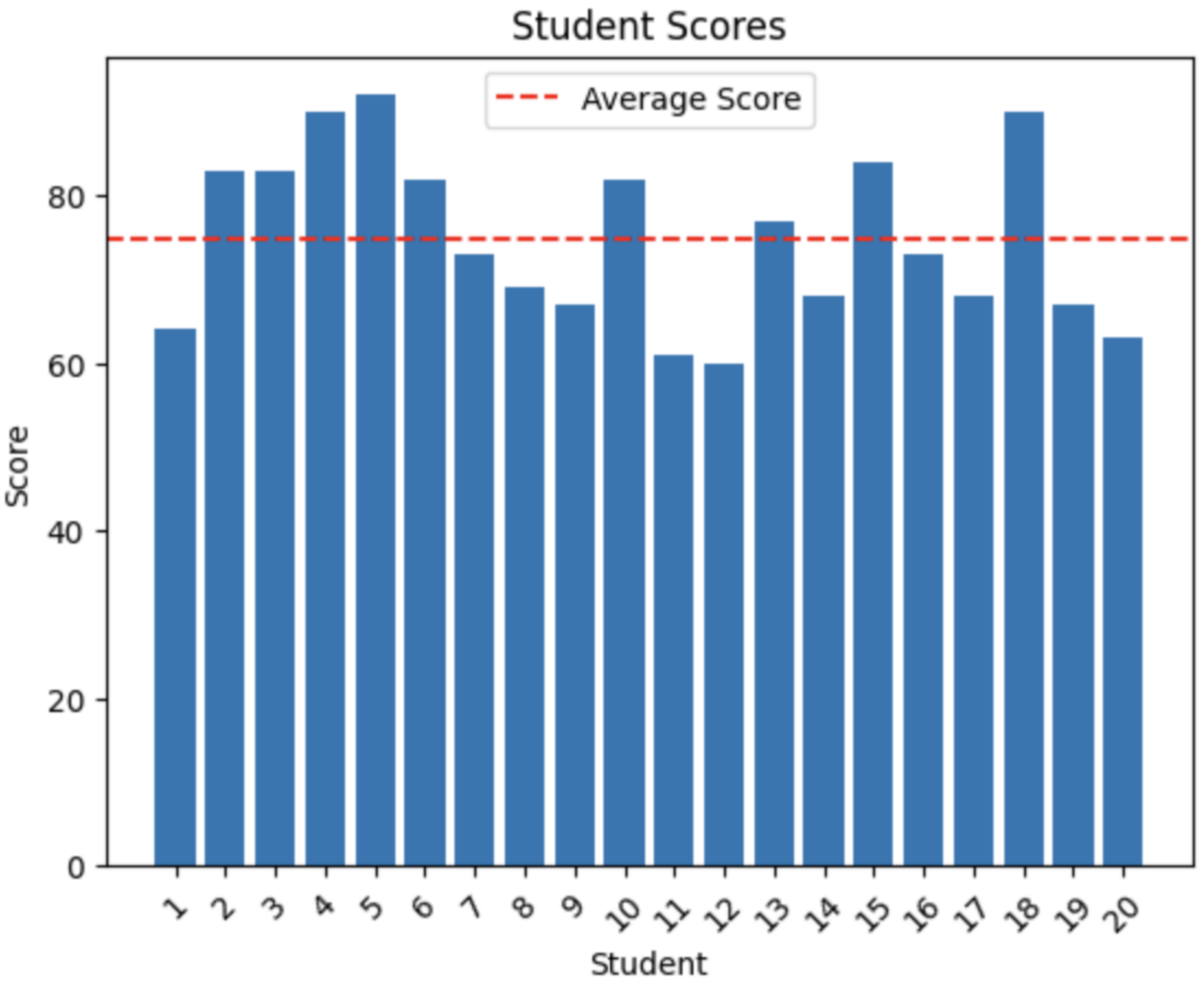
```python
# Lists
fruits = ["apple", "banana", "orange"]
print(fruits[0])  # Output: apple
```

```
# Dictionaries
person = {"name": "Alice", "age": 25, "is_student": True}
print(person["age"])  # Output: 25
```

# Commonly-Used Python Packages

- **NumPy**: A powerful package for scientific computing and working with arrays.
- **Pandas**: A library for data manipulation and analysis, providing data structures like DataFrames.
- **Matplotlib**: A popular plotting library for creating visualizations and charts.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(1)  # Set random seed for reproducibility
n_students = 50
names = ['Student ' + str(i) for i in range(1, n_students + 1)]
ages = np.random.randint(18, 23, size=n_students)
scores = np.random.randint(60, 100, size=n_students)

# Create DataFrame
data = pd.DataFrame({'Name': names, 'Age': ages, 'Score': scores})

# Calculate average score
average_score = data['Score'].mean()

# Create bar plot
plt.bar(data['Name'], data['Score'])
plt.axhline(average_score, color='red', linestyle='--', label='Average Score')
plt.xlabel('Student')
plt.ylabel('Score')
plt.title('Student Scores')
plt.xticks(rotation=45)
plt.legend()

# Save the image
plt.savefig('student_scores.png')

# Show the image
plt.show()
```



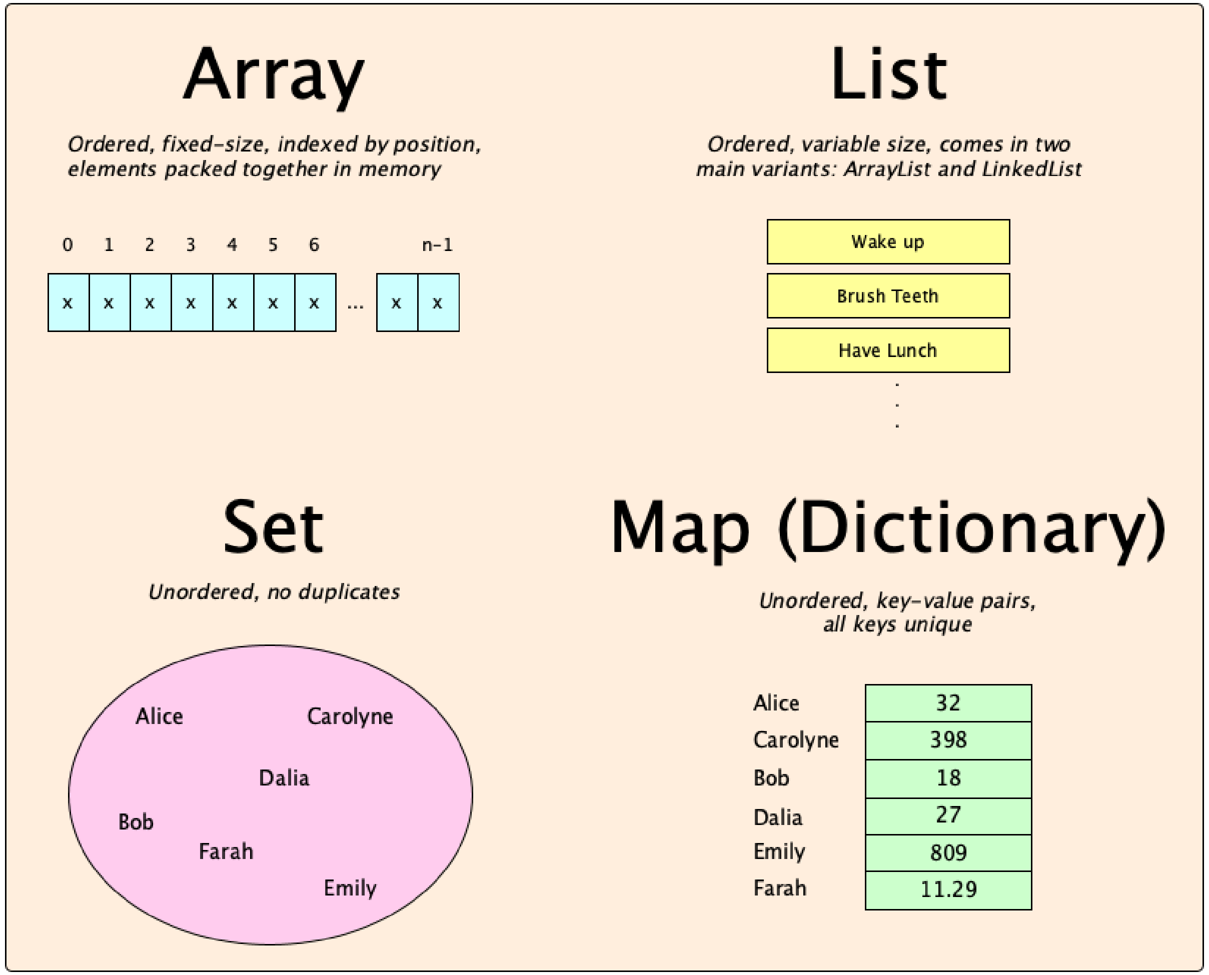
Let's look at the output of the code!

- **Scikit-learn**: A machine learning library that provides various algorithms and tools for data mining and analysis.
- **OpenCV**: An open-source computer vision library with various functions for image and video processing.
- **Pygame**: A library for game development, providing tools for graphics, sound, and user input handling.
- **PyTorch**: A deep learning framework that provides tensors and dynamic neural networks for efficient computation.
- **Seaborn**: A data visualization library based on Matplotlib, offering enhanced statistical graphics.
- **SciPy**: A library for scientific and technical computing, providing modules for optimization, linear algebra, integration, and more.

# Using Python to Solve Algorithm Problems

## Step 1: Understanding Algorithms

- Gain a solid understanding of algorithms, their components, and problem-solving techniques.



*Ordered, fixed-size, indexed by position, elements packed together in memory*

*Ordered, variable size, comes in two main variants: ArrayList and LinkedList*

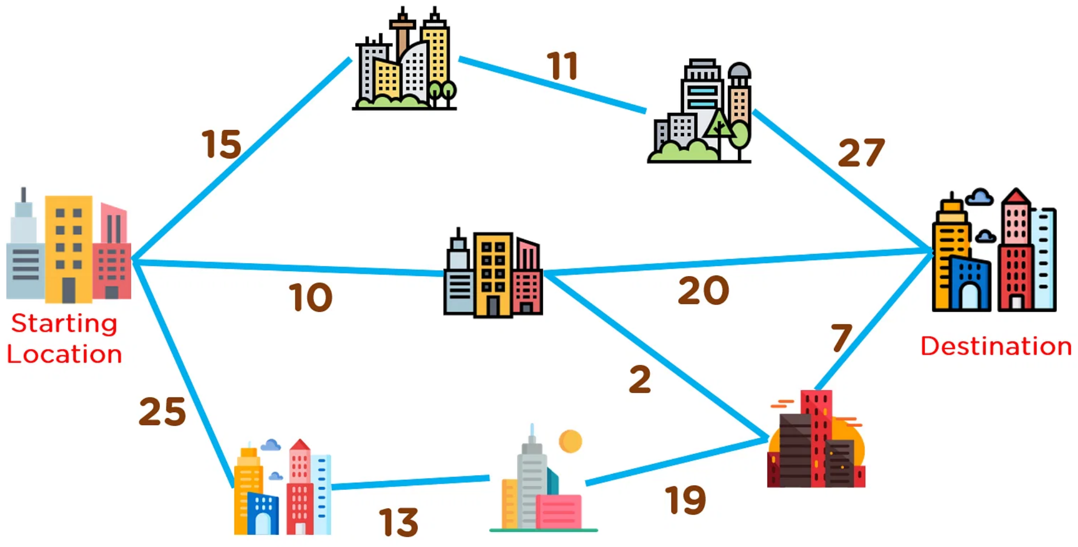
*Unordered, no duplicates*

*Unordered, key-value pairs, all keys unique*

| | |
|---|---|
| Alice | 32 |
| Carolyne | 398 |
| Bob | 18 |
| Dalia | 27 |
| Emily | 809 |
| Farah | 11.29 |

## Step 2: Data Structures

- Familiarize yourself with fundamental data structures:
    - Arrays
    - Linked Lists
    - Stacks
    - Queues
    - Trees



## Step 3: Algorithm Design

- Learn algorithm design techniques:
  - Divide and Conquer
  - Greedy Algorithms
  - Dynamic Programming
  - Backtracking

## Step 4: Implementing Algorithms in Python

- Translate algorithmic knowledge into Python code:
    - Define functions and classes
    - Example:

```
# Example: Binary Search
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1
```

## Step 5: Algorithm Analysis

- Evaluate algorithm efficiency:
    - Time Complexity
    - Space Complexity

## Step 6: Algorithm Libraries

- Utilize Python libraries for algorithmic problem-solving:
  - `math`
  - `itertools`
  - `collections`

# Step 7: Practice with Algorithmic Challenges

- Solve algorithmic problems on platforms like LeetCode or HackerRank.
- Example:
  - LeetCode Problem: "Two Sum"

  -
    ```python
    def two_sum(nums, target):
        num_dict = {}

        for i, num in enumerate(nums):
            complement = target - num
            if complement in num_dict:
                return [num_dict[complement], i]
            num_dict[num] = i

        return []
    ```

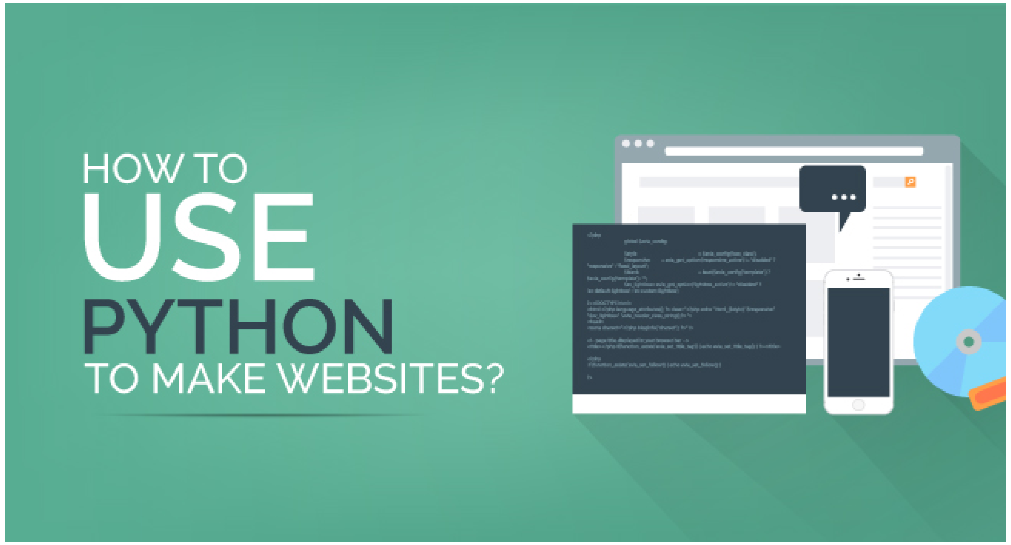
## Step 8: Continued Learning

- Stay updated with new algorithms and optimization techniques:
  - Read books, articles, and research papers on algorithms
  - Attend online courses and workshops

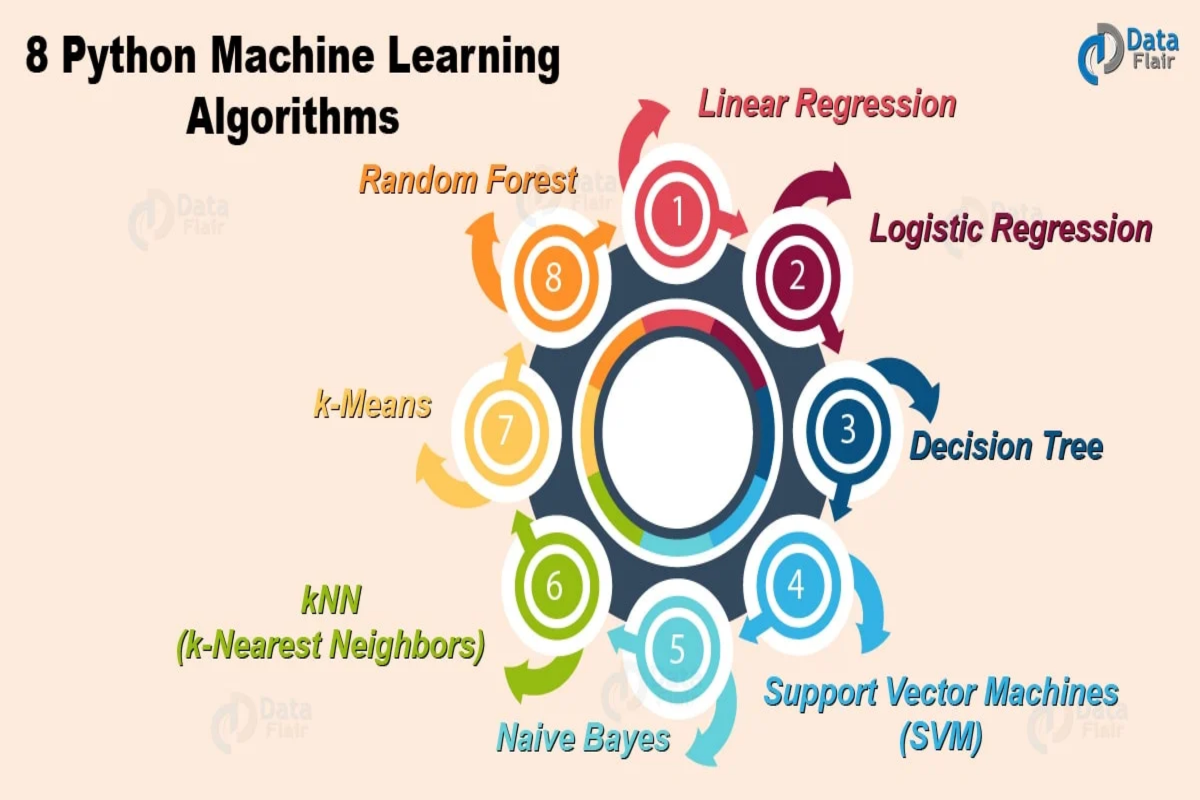
**Remember, solving algorithm problems requires understanding the core concepts and applying them in Python. Practice regularly, challenge yourself with diverse problems, and continuously enhance your problem-solving skills.**

# Applications of Python and Algorithms

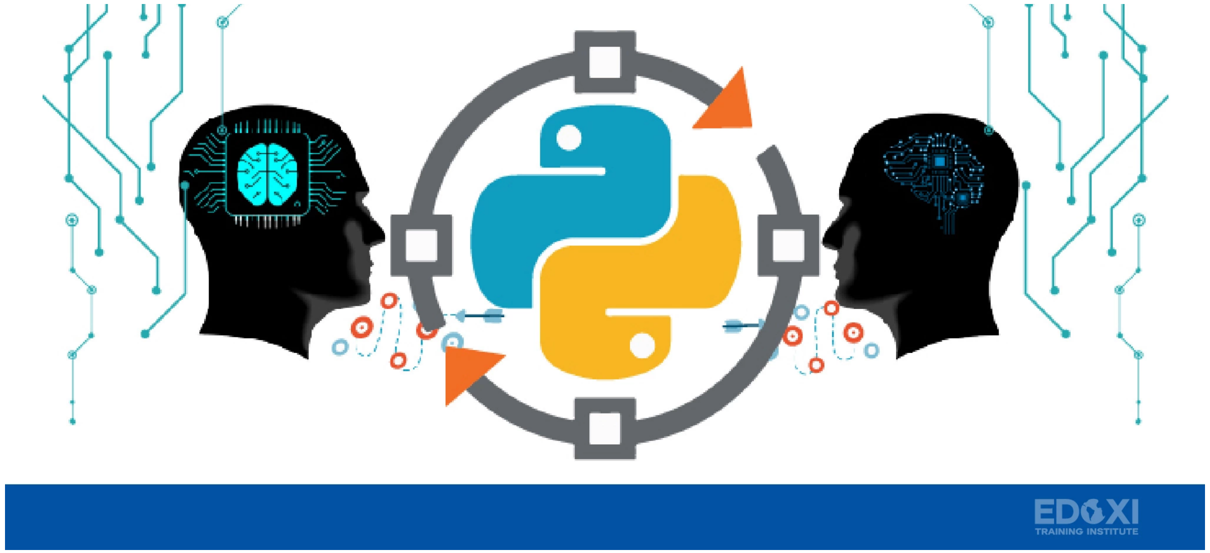
Python and algorithms have a wide range of applications in various domains. Let's explore some of the key areas where Python and algorithms are commonly used



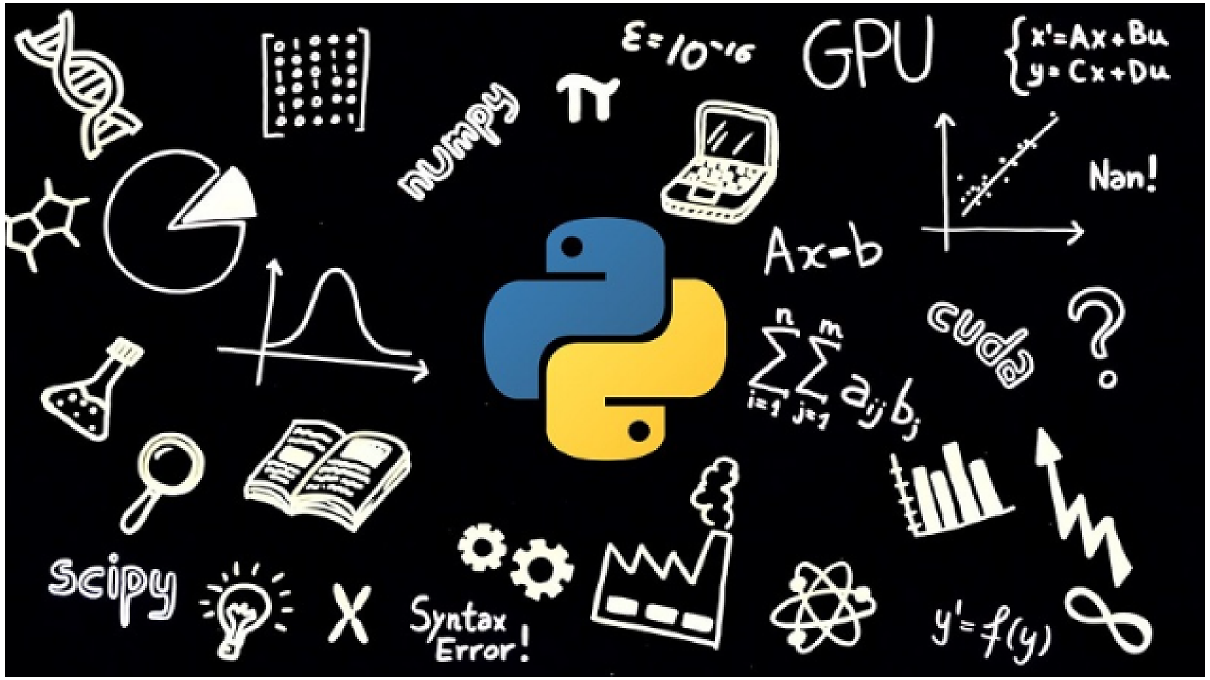
1. **Web Development**
   - Python frameworks like Django and Flask enable rapid web application development.
   - Algorithms are used for tasks like routing, caching, and optimizing web applications.



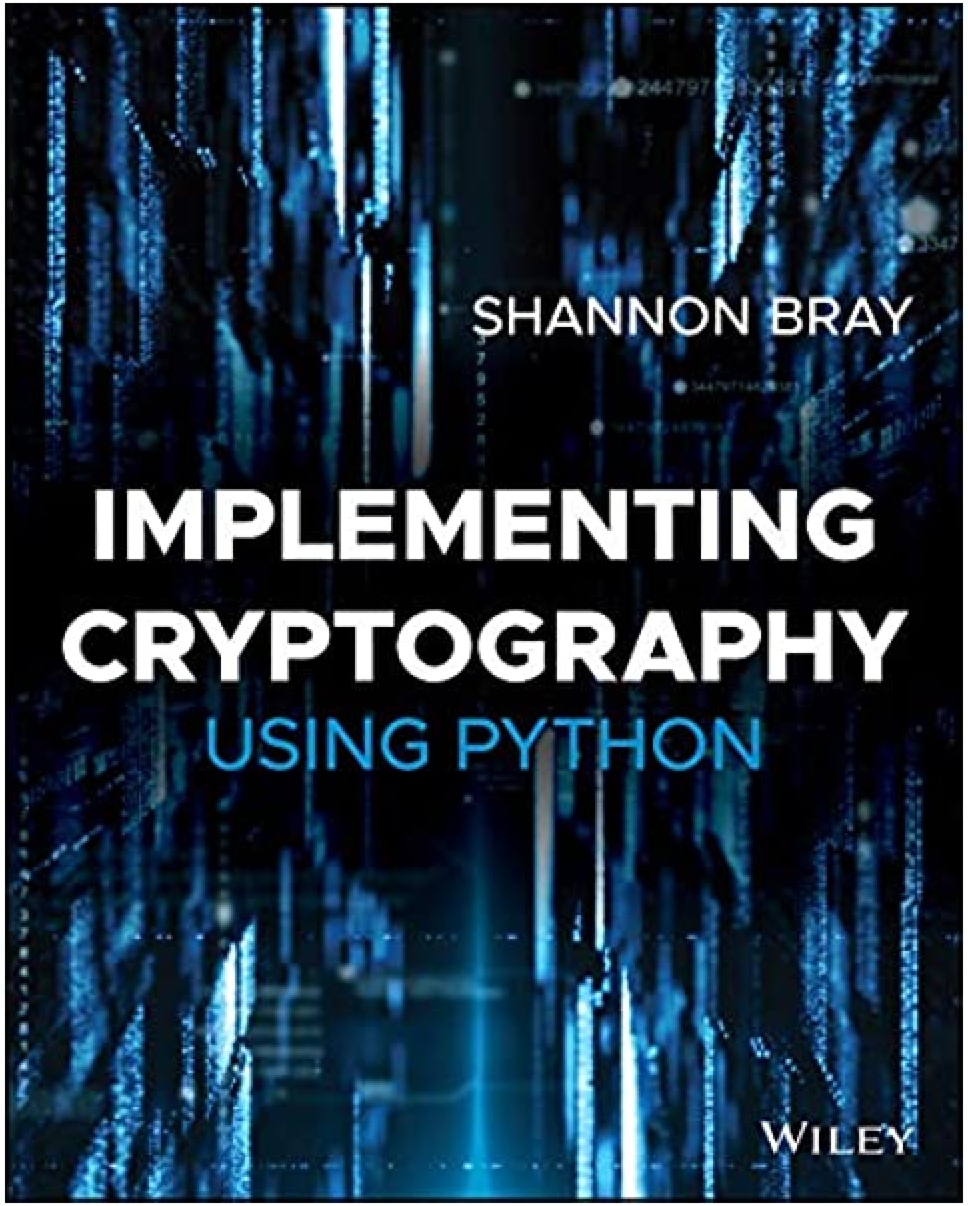
2. **Data Science and Machine Learning**
   - Python's rich ecosystem, including libraries like NumPy, Pandas, and Scikit-learn, makes it ideal for data analysis and machine learning.
   - Algorithms play a crucial role in tasks like data preprocessing, feature selection, and model training.



3. **Artificial Intelligence and Natural Language Processing**
    - Python libraries such as TensorFlow, PyTorch, and NLTK provide powerful tools for AI and NLP applications.
    - Algorithms are used for tasks like image recognition, speech processing, sentiment analysis, and chatbots.



4. **Scientific Computing**
   - Python's scientific libraries, such as SciPy and Matplotlib, are widely used in scientific research and simulations.
   - Algorithms are utilized for mathematical modeling, numerical computations, and data visualization.



5. **Cybersecurity and Cryptography**
    - Python is used for building security tools and analyzing vulnerabilities.
    - Algorithms are employed for encryption, decryption, and secure communication protocols.

6. **Financial Analysis and Trading**
   - Python's libraries, such as Pandas and NumPy, are extensively used in financial analysis and modeling.
   - Algorithms play a critical role in tasks like portfolio optimization, algorithmic trading, and risk management.

7. **Game Development**
   - Python's simplicity and libraries like Pygame make it suitable for game development.
   - Algorithms are utilized for game logic, pathfinding, and AI opponents.

These are just a few examples of the countless applications of Python and algorithms. The versatility of Python and the power of algorithms make them valuable tools across various industries and domains.

# Thank You!

hope to see you guys next time

Contact: PM via wechat or Email