🐍 One-Month Python Learning Plan (Revised)

Since you've already completed **Python Basics through Conditional Statements**, here's your structured 4-week plan with code snippets for each new concept. Final projects are moved to after the month completion.

📅 Week 1: Loops, Functions & Lambda

Days 1-2: Loops

Day 1: For and While Loops

For Loop:

```
# Iterating through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Using range()
for i in range(5):
    print(i)  # Prints 0, 1, 2, 3, 4
```

While Loop:

```
# Basic while loop
count = 0
while count < 5:
    print(f"Count is: {count}")
    count += 1
```

Practice Tasks:

- Print numbers 1-100
- Sum all numbers in a list

- Print even numbers from 1-50

Day 2: Loop Control & Nested Loops

Loop Control Statements:

```
# break - exits the loop entirely
for i in range(10):
    if i == 5:
        break
    print(i)  # Prints 0, 1, 2, 3, 4

# continue - skips current iteration
for i in range(5):
    if i == 2:
        continue
    print(i)  # Prints 0, 1, 3, 4

# pass - placeholder, does nothing
for i in range(3):
    pass  # Will implement later
```

Nested Loops:

```
# Multiplication table
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} x {j} = {i*j}")
    print("---")

# Pattern printing - right triangle
for i in range(1, 6):
    print("*" * i)
# Output:
# *
# **
# ***
# ****
# *****
```

Days 3-4: Advanced Data Structures

Day 3: Nested Lists (2D Arrays)

```
# Creating a 2D list (matrix)
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Accessing elements
print(matrix[0][0])  # 1 (first row, first column)
print(matrix[1][2])  # 6 (second row, third column)

# Looping through 2D list
for row in matrix:
    for element in row:
        print(element, end=" ")
    print()  # New line after each row

# Using indices
for i in range(len(matrix)):
    for j in range(len(matrix[i])):
        print(f"matrix[{i}][{j}] = {matrix[i][j]}")
```

Practice Tasks:

- Create a 3x3 matrix and find the sum of all elements
- Transpose a matrix (swap rows and columns)

Day 4: Nested Dictionaries

```
# Nested dictionary - student records
students = {
    "student1": {
        "name": "Alice",
        "age": 20,
        "grades": {"math": 90, "science": 85}
    },
    "student2": {
        "name": "Bob",
        "age": 22,
        "grades": {"math": 78, "science": 92}
```

```
    }
}

# Accessing nested values
print(students["student1"]["name"])  # Alice
print(students["student1"]["grades"]["math"])  # 90

# Looping through nested dictionary
for student_id, info in students.items():
    print(f"\n{student_id}:")
    for key, value in info.items():
        print(f"  {key}: {value}")

# Adding new nested data
students["student3"] = {
    "name": "Charlie",
    "age": 21,
    "grades": {"math": 88, "science": 90}
}
```

Days 5-6: Functions

Day 5: Function Basics

Defining and Calling Functions:

```
# Basic function
def greet():
    print("Hello, World!")

greet()  # Call the function

# Function with parameters
def greet_person(name):
    print(f"Hello, {name}!")

greet_person("Alice")

# Function with return statement
def add_numbers(a, b):
    return a + b
```

```
result = add_numbers(5, 3)
print(result)  # 8

# Default parameters
def greet_with_default(name="Guest"):
    return f"Welcome, {name}!"

print(greet_with_default())        # Welcome, Guest!
print(greet_with_default("John"))  # Welcome, John!

# Multiple return values
def get_stats(numbers):
    return min(numbers), max(numbers), sum(numbers)

minimum, maximum, total = get_stats([1, 2, 3, 4, 5])
print(f"Min: {minimum}, Max: {maximum}, Sum: {total}")
```

Day 6: Advanced Functions

Variable Scopes:

```
# Global vs Local scope
global_var = "I'm global"

def my_function():
    local_var = "I'm local"
    print(global_var)  # Can access global
    print(local_var)   # Can access local

my_function()
# print(local_var)  # Error! local_var not accessible here

# Modifying global variable
counter = 0

def increment():
    global counter  # Declare as global to modify
    counter += 1

increment()
print(counter)  # 1
```

Recursive Functions:

```
# Factorial using recursion
def factorial(n):
    if n == 0 or n == 1:  # Base case
        return 1
    else:
        return n * factorial(n - 1)  # Recursive case

print(factorial(5))  # 120 (5 * 4 * 3 * 2 * 1)

# Fibonacci sequence
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

# Print first 10 Fibonacci numbers
for i in range(10):
    print(fibonacci(i), end=" ")  # 0 1 1 2 3 5 8 13 21 34
```

Day 7: Lambda & Higher-Order Functions

Lambda Functions:

```
# Regular function vs Lambda
def square(x):
    return x ** 2

# Same as lambda
square_lambda = lambda x: x ** 2

print(square(5))        # 25
print(square_lambda(5)) # 25

# Lambda with multiple parameters
add = lambda a, b: a + b
print(add(3, 4))  # 7
```

Built-in Higher-Order Functions:

```
numbers = [1, 2, 3, 4, 5]

# map() - apply function to each element
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # [1, 4, 9, 16, 25]

# filter() - filter elements based on condition
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)  # [2, 4]

# zip() - combine multiple iterables
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
combined = list(zip(names, ages))
print(combined)  # [('Alice', 25), ('Bob', 30), ('Charlie', 35)]

# Using zip in a loop
for name, age in zip(names, ages):
    print(f"{name} is {age} years old")
```

🔨 **Mini Project: TO-DO CLI App**

📅 Week 2: Error Handling & OOP

Days 8-9: Error Handling

Day 8: Exception Handling

```
# Basic try-except
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")

# Multiple exceptions
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ValueError:
    print("Invalid input! Please enter a number.")
```

```
except ZeroDivisionError:
    print("Cannot divide by zero!")

# try-except-else-finally
try:
    file = open("test.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("File not found!")
else:
    print("File read successfully!")
    print(content)
finally:
    print("This always executes!")

# Catching all exceptions
try:
    # risky code
    x = 1 / 0
except Exception as e:
    print(f"An error occurred: {e}")
```

Day 9: Debugging Techniques

```
# Using print statements for debugging
def calculate_average(numbers):
    print(f"DEBUG: Input received: {numbers}")  # Debug line
    total = sum(numbers)
    print(f"DEBUG: Total = {total}")  # Debug line
    count = len(numbers)
    print(f"DEBUG: Count = {count}")  # Debug line
    average = total / count
    return average

# Using assert for debugging
def divide(a, b):
    assert b != 0, "Divisor cannot be zero!"
    return a / b

# Try-except for debugging
def safe_divide(a, b):
    try:
        return a / b
```

```
    except ZeroDivisionError as e:
        print(f"Error: {e}")
        return None
```

Days 10-12: OOP Fundamentals

Day 10: Classes & Objects

Object-Oriented Programming (OOP) in Python is based on the concept of "objects," which can contain data and code: data in the form of instance variables (often known as attributes or properties), and code in the form of methods .

```
# Defining a class
class Dog:
    # Class variable (shared by all instances)
    species = "Canis familiaris"

    # Constructor (initializer)
    def __init__(self, name, age):
        # Instance variables (unique to each instance)
        self.name = name
        self.age = age

    # Instance method
    def bark(self):
        return f"{self.name} says Woof!"

    def description(self):
        return f"{self.name} is {self.age} years old"

# Creating objects (instances)
dog1 = Dog("Buddy", 3)
dog2 = Dog("Max", 5)

# Accessing attributes and methods
print(dog1.name)          # Buddy
print(dog1.bark())        # Buddy says Woof!
print(dog2.description()) # Max is 5 years old
print(Dog.species)        # Canis familiaris (class variable)
```

Day 11: Methods & Constructors

```
class BankAccount:
    # Constructor with __init__
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance
        print(f"Account created for {owner}")

    # Instance methods
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return f"Deposited ${amount}. New balance: ${self.balance}"
        return "Invalid amount"

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            return f"Withdrew ${amount}. New balance: ${self.balance}"
        return "Insufficient funds or invalid amount"

    def get_balance(self):
        return f"{self.owner}'s balance: ${self.balance}"

# Using the class
account = BankAccount("Alice", 100)
print(account.deposit(50))    # Deposited $50. New balance: $150
print(account.withdraw(30))   # Withdrew $30. New balance: $120
print(account.get_balance())  # Alice's balance: $120
```

Day 12: Inheritance & Polymorphism

Master advanced Python topics by diving deep into object-oriented programming (OOP) concepts such as classes, methods, inheritance, polymorphism, and encapsulation .

```
# Inheritance - Parent class
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return "Some sound"
```

```python
# Child classes inheriting from Animal
class Dog(Animal):
    def speak(self):  # Method overriding
        return f"{self.name} says Woof!"

class Cat(Animal):
    def speak(self):  # Method overriding
        return f"{self.name} says Meow!"

# Polymorphism - same method, different behavior
animals = [Dog("Buddy"), Cat("Whiskers"), Dog("Max")]

for animal in animals:
    print(animal.speak())
# Output:
# Buddy says Woof!
# Whiskers says Meow!
# Max says Woof!

# Using super() to call parent method
class Bird(Animal):
    def __init__(self, name, can_fly=True):
        super().__init__(name)  # Call parent constructor
        self.can_fly = can_fly

    def speak(self):
        return f"{self.name} says Chirp!"

bird = Bird("Tweety")
print(bird.speak())  # Tweety says Chirp!
```

Days 13-14: OOP Advanced

Day 13: Encapsulation & Abstraction

```python
# Encapsulation - hiding internal data
class Employee:
    def __init__(self, name, salary):
        self.name = name           # Public attribute
        self._department = "IT"    # Protected (convention)
        self.__salary = salary     # Private attribute
```

```
    # Getter method
    def get_salary(self):
        return self.__salary

    # Setter method
    def set_salary(self, new_salary):
        if new_salary > 0:
            self.__salary = new_salary
        else:
            print("Invalid salary!")

emp = Employee("John", 50000)
print(emp.name)           # John (public - accessible)
print(emp._department)    # IT (protected - accessible but discouraged)
# print(emp.__salary)     # Error! Private attribute
print(emp.get_salary())   # 50000 (using getter)

# Abstraction using ABC (Abstract Base Class)
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

rect = Rectangle(5, 3)
print(f"Area: {rect.area()}")        # Area: 15
print(f"Perimeter: {rect.perimeter()}")  # Perimeter: 16
```

Day 14: Static & Class Methods

```
class MathOperations:
    pi = 3.14159  # Class variable

    def __init__(self, value):
        self.value = value

    # Instance method - needs self
    def square(self):
        return self.value ** 2

    # Static method - doesn't need self or cls
    @staticmethod
    def add(a, b):
        return a + b

    # Class method - needs cls (the class itself)
    @classmethod
    def circle_area(cls, radius):
        return cls.pi * radius ** 2

    @classmethod
    def from_string(cls, value_string):
        # Alternative constructor
        value = int(value_string)
        return cls(value)

# Using different method types
math_obj = MathOperations(5)
print(math_obj.square())              # 25 (instance method)
print(MathOperations.add(3, 4))       # 7 (static method)
print(MathOperations.circle_area(5))  # 78.53975 (class method)

# Using class method as alternative constructor
obj = MathOperations.from_string("10")
print(obj.value)  # 10
```

🔨 **Mini Project: Password Generator**

📅 Week 3: Advanced Python Concepts

Days 15-16: File Handling

Day 15: Read/Write Files

```
# Writing to a file
with open("example.txt", "w") as file:
    file.write("Hello, World!\n")
    file.write("This is line 2.\n")
    file.write("This is line 3.\n")

# Reading entire file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)

# Reading line by line
with open("example.txt", "r") as file:
    for line in file:
        print(line.strip())  # strip() removes newline

# Reading all lines into a list
with open("example.txt", "r") as file:
    lines = file.readlines()
    print(lines)  # ['Hello, World!\n', 'This is line 2.\n', ...]

# Appending to a file
with open("example.txt", "a") as file:
    file.write("This line was appended.\n")

# Reading and writing (r+)
with open("example.txt", "r+") as file:
    content = file.read()
    file.write("\nNew content added!")
```

Day 16: Working with CSV Files

```
import csv

# Writing CSV file
data = [
    ["Name", "Age", "City"],
    ["Alice", 25, "New York"],
```

```
    ["Bob", 30, "Los Angeles"],
    ["Charlie", 35, "Chicago"]
]

with open("people.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(data)

# Reading CSV file
with open("people.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)

# Using DictReader and DictWriter
with open("people.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(f"{row['Name']} is {row['Age']} years old")

# Writing with DictWriter
people = [
    {"Name": "David", "Age": 28, "City": "Boston"},
    {"Name": "Eve", "Age": 32, "City": "Seattle"}
]

with open("more_people.csv", "w", newline="") as file:
    fieldnames = ["Name", "Age", "City"]
    writer = csv.DictWriter(file, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(people)
```

Days 17-18: Modules & Packages

Day 17: Python Modules

```
# Built-in modules
import os
import datetime
import random
import math
```

```
# os module - file/directory operations
print(os.getcwd())           # Current working directory
print(os.listdir("."))       # List files in current directory
# os.mkdir("new_folder")     # Create new directory

# datetime module
now = datetime.datetime.now()
print(now)                                  # Current date and time
print(now.strftime("%Y-%m-%d %H:%M"))       # Formatted date

# random module
print(random.randint(1, 100))               # Random integer
print(random.choice(["a", "b", "c"]))       # Random choice
print(random.shuffle([1, 2, 3, 4, 5]))      # Shuffle list

# math module
print(math.sqrt(16))     # 4.0
print(math.pi)           # 3.141592653589793
print(math.ceil(4.2))    # 5
print(math.floor(4.8))   # 4

# Creating a custom module (save as mymodule.py)
# --- mymodule.py ---
# def greet(name):
#     return f"Hello, {name}!"
#
# PI = 3.14159

# Using custom module
# import mymodule
# print(mymodule.greet("Alice"))
# print(mymodule.PI)
```

Day 18: Virtual Environments & Pip

```
# Virtual Environment Commands (run in terminal)
"""
# Create virtual environment
python -m venv myenv

# Activate (Windows)
myenv\Scripts\activate
```

```
# Activate (Mac/Linux)
source myenv/bin/activate

# Deactivate
deactivate

# Install packages
pip install requests
pip install pandas numpy

# List installed packages
pip list

# Save dependencies
pip freeze > requirements.txt

# Install from requirements
pip install -r requirements.txt
"""

# Creating a package structure
"""
my_package/
    __init__.py
    module1.py
    module2.py
    subpackage/
        __init__.py
        module3.py
"""

# __init__.py example
# from .module1 import function1
# from .module2 import function2

# Using the package
# from my_package import function1
# from my_package.subpackage import module3
```

Days 19-20: Custom Exceptions & Regex

Day 19: Custom Exceptions

```
# Creating custom exceptions
class InsufficientFundsError(Exception):
    def __init__(self, balance, amount):
        self.balance = balance
        self.amount = amount
        self.message = f"Cannot withdraw ${amount}. Balance is only ${bal
        super().__init__(self.message)

class InvalidAgeError(Exception):
    pass

# Using custom exceptions
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def withdraw(self, amount):
        if amount > self.balance:
            raise InsufficientFundsError(self.balance, amount)
        self.balance -= amount
        return self.balance

# Handling custom exceptions
account = BankAccount(100)

try:
    account.withdraw(150)
except InsufficientFundsError as e:
    print(f"Error: {e.message}")
    print(f"You tried to withdraw: ${e.amount}")
    print(f"Your balance: ${e.balance}")

# Raising exceptions
def set_age(age):
    if age < 0:
        raise InvalidAgeError("Age cannot be negative!")
    if age > 150:
        raise InvalidAgeError("Age seems unrealistic!")
    return age
```

Day 20: Regular Expressions (Regex)

```python
import re

text = "Contact us at support@example.com or sales@company.org"

# Basic patterns
# search() - finds first match
match = re.search(r"\d+", "There are 25 apples and 30 oranges")
if match:
    print(match.group())  # 25

# findall() - finds all matches
numbers = re.findall(r"\d+", "There are 25 apples and 30 oranges")
print(numbers)  # ['25', '30']

# Email pattern
emails = re.findall(r"[\w\.-]+@[\w\.-]+", text)
print(emails)  # ['support@example.com', 'sales@company.org']

# sub() - replace pattern
cleaned = re.sub(r"\d+", "X", "Phone: 123-456-7890")
print(cleaned)  # Phone: X-X-X

# Common patterns
patterns = {
    "digits": r"\d+",           # One or more digits
    "words": r"\w+",            # One or more word characters
    "email": r"[\w\.-]+@[\w\.-]+\.\w+",
    "phone": r"\d{3}-\d{3}-\d{4}",
    "url": r"https?://[\w\./]+"
}

# Validation example
def validate_email(email):
    pattern = r"^[\w\.-]+@[\w\.-]+\.\w+$"
    return bool(re.match(pattern, email))

print(validate_email("test@example.com"))  # True
print(validate_email("invalid-email"))     # False
```

Day 21: SQLite3 with Python

```python
import sqlite3

# Connect to database (creates if doesn't exist)
conn = sqlite3.connect("example.db")
cursor = conn.cursor()

# Create table
cursor.execute("""
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        email TEXT UNIQUE,
        age INTEGER
    )
""")

# INSERT - Create
cursor.execute("""
    INSERT INTO users (name, email, age)
    VALUES (?, ?, ?)
""", ("Alice", "alice@example.com", 25))

# Insert multiple records
users = [
    ("Bob", "bob@example.com", 30),
    ("Charlie", "charlie@example.com", 35)
]
cursor.executemany("INSERT INTO users (name, email, age) VALUES (?, ?, ?)

# SELECT - Read
cursor.execute("SELECT * FROM users")
all_users = cursor.fetchall()
print(all_users)

# SELECT with condition
cursor.execute("SELECT * FROM users WHERE age > ?", (25,))
print(cursor.fetchall())

# UPDATE
cursor.execute("UPDATE users SET age = ? WHERE name = ?", (26, "Alice"))

# DELETE
cursor.execute("DELETE FROM users WHERE name = ?", ("Charlie",))
```

```
# Commit changes and close
conn.commit()
conn.close()
```

🔨 Mini Project: Alarm Clock App

📅 Week 4: Data Handling & Testing

Days 22-23: Web Scraping

Day 22: BeautifulSoup Basics

```
# Install: pip install beautifulsoup4 requests
from bs4 import BeautifulSoup
import requests

# Fetch webpage
url = "https://example.com"
response = requests.get(url)
html_content = response.text

# Parse HTML
soup = BeautifulSoup(html_content, "html.parser")

# Find elements
title = soup.find("title")
print(title.text)

# Find all elements
all_paragraphs = soup.find_all("p")
for p in all_paragraphs:
    print(p.text)

# Find by class
# items = soup.find_all("div", class_="item")

# Find by id
# header = soup.find(id="header")

# Get attributes
links = soup.find_all("a")
```

```
for link in links:
    href = link.get("href")
    text = link.text
    print(f"{text}: {href}")
```

Day 23: Practical Web Scraping

```
from bs4 import BeautifulSoup
import requests
import csv

def scrape_quotes():
    url = "http://quotes.toscrape.com"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")

    quotes_data = []

    quotes = soup.find_all("div", class_="quote")
    for quote in quotes:
        text = quote.find("span", class_="text").text
        author = quote.find("small", class_="author").text
        tags = [tag.text for tag in quote.find_all("a", class_="tag")]

        quotes_data.append({
            "quote": text,
            "author": author,
            "tags": ", ".join(tags)
        })

    return quotes_data

# Save to CSV
quotes = scrape_quotes()
with open("quotes.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.DictWriter(file, fieldnames=["quote", "author", "tags"])
    writer.writeheader()
    writer.writerows(quotes)

print(f"Scraped {len(quotes)} quotes!")
```

🔨 Mini Project: App for Fetching Data from a Website

Days 24-26: Pandas & Data Analysis

Day 24: Intro to Pandas

```
# Install: pip install pandas
import pandas as pd

# Creating DataFrames
data = {
    "Name": ["Alice", "Bob", "Charlie", "David"],
    "Age": [25, 30, 35, 28],
    "City": ["New York", "LA", "Chicago", "Boston"],
    "Salary": [50000, 60000, 70000, 55000]
}

df = pd.DataFrame(data)
print(df)

# Basic operations
print(df.head())        # First 5 rows
print(df.tail(2))       # Last 2 rows
print(df.shape)         # (rows, columns)
print(df.columns)       # Column names
print(df.dtypes)        # Data types
print(df.describe())    # Statistical summary

# Accessing data
print(df["Name"])           # Single column
print(df[["Name", "Age"]])  # Multiple columns
print(df.iloc[0])           # First row by index
print(df.loc[0, "Name"])    # Specific cell
```

Day 25: Reading/Writing Files with Pandas

```
import pandas as pd

# Reading CSV
df = pd.read_csv("data.csv")

# Reading Excel
# df = pd.read_excel("data.xlsx", sheet_name="Sheet1")
```

```
# Writing CSV
df.to_csv("output.csv", index=False)

# Writing Excel
# df.to_excel("output.xlsx", index=False)

# Reading with options
df = pd.read_csv(
    "data.csv",
    sep=",",              # Delimiter
    header=0,             # Row number for header
    usecols=["Name", "Age"],  # Specific columns
    nrows=100             # Number of rows to read
)

# Creating sample data and saving
sample_data = pd.DataFrame({
    "Product": ["Apple", "Banana", "Orange", "Mango"],
    "Price": [1.5, 0.5, 0.75, 2.0],
    "Quantity": [100, 150, 80, 60]
})
sample_data.to_csv("products.csv", index=False)
```

Day 26: Data Analysis

```
import pandas as pd

# Sample data
df = pd.DataFrame({
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve"],
    "Department": ["IT", "HR", "IT", "Finance", "HR"],
    "Salary": [50000, 45000, 60000, 55000, 48000],
    "Age": [25, 30, 35, 28, 32]
})

# Filtering
it_employees = df[df["Department"] == "IT"]
high_salary = df[df["Salary"] > 50000]
print(high_salary)

# Multiple conditions
filtered = df[(df["Department"] == "IT") & (df["Salary"] > 55000)]
```

```python
# Sorting
sorted_df = df.sort_values("Salary", ascending=False)
print(sorted_df)

# Grouping
dept_stats = df.groupby("Department").agg({
    "Salary": ["mean", "sum", "count"],
    "Age": "mean"
})
print(dept_stats)

# Adding new columns
df["Bonus"] = df["Salary"] * 0.1
df["Tax"] = df["Salary"] * 0.2

# Value counts
print(df["Department"].value_counts())
```

Days 27-28: Testing & Debugging

Day 27: Unit Testing

```python
# Save as calculator.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero!")
    return a / b

# Save as test_calculator.py
import unittest
from calculator import add, subtract, multiply, divide
```

```python
class TestCalculator(unittest.TestCase):

    def test_add(self):
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)

    def test_subtract(self):
        self.assertEqual(subtract(5, 3), 2)
        self.assertEqual(subtract(0, 5), -5)

    def test_multiply(self):
        self.assertEqual(multiply(3, 4), 12)
        self.assertEqual(multiply(-2, 3), -6)

    def test_divide(self):
        self.assertEqual(divide(10, 2), 5)
        self.assertAlmostEqual(divide(1, 3), 0.333, places=2)

    def test_divide_by_zero(self):
        with self.assertRaises(ValueError):
            divide(10, 0)

# Run tests
if __name__ == "__main__":
    unittest.main()

# Run from terminal: python -m unittest test_calculator.py
```

Day 28: Advanced Debugging

```python
# Using the logging module (better than print)
import logging

# Configure logging
logging.basicConfig(
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s"
)

def calculate_total(items):
    logging.debug(f"Input items: {items}")
```

```
    total = 0
    for item in items:
        logging.debug(f"Processing item: {item}")
        total += item["price"] * item["quantity"]
        logging.debug(f"Running total: {total}")

    logging.info(f"Final total calculated: {total}")
    return total

items = [
    {"name": "Apple", "price": 1.5, "quantity": 10},
    {"name": "Banana", "price": 0.5, "quantity": 20}
]

result = calculate_total(items)

# Using pdb (Python Debugger)
"""
import pdb

def buggy_function(x):
    pdb.set_trace()  # Debugger starts here
    result = x * 2
    result = result + 10
    return result

# pdb commands:
# n (next) - execute next line
# s (step) - step into function
# c (continue) - continue execution
# p variable - print variable value
# q (quit) - quit debugger
"""

# Using breakpoint() - Python 3.7+
def another_function(data):
    processed = []
    for item in data:
        # breakpoint()  # Uncomment to debug
        processed.append(item * 2)
    return processed

# Try-except for error tracking
```

```
import traceback

def risky_operation():
    try:
        result = 10 / 0
    except Exception as e:
        logging.error(f"Error occurred: {e}")
        logging.error(traceback.format_exc())
```

Days 29-30: Review & Practice

Day 29: Comprehensive Review

```
# Review Exercise: Build a mini application combining all concepts

class Student:
    """Class demonstrating OOP concepts"""

    all_students = []  # Class variable

    def __init__(self, name, age, grades=None):
        self.name = name
        self.age = age
        self._grades = grades or {}  # Encapsulation
        Student.all_students.append(self)

    def add_grade(self, subject, grade):
        self._grades[subject] = grade

    def get_average(self):
        if not self._grades:
            return 0
        return sum(self._grades.values()) / len(self._grades)

    @classmethod
    def get_all_students(cls):
        return cls.all_students

    @staticmethod
    def is_passing(grade):
        return grade >= 60
```

```
    def __str__(self):
        return f"Student({self.name}, Age: {self.age})"


def save_students_to_csv(students, filename):
    """File handling with CSV"""
    import csv

    with open(filename, "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerow(["Name", "Age", "Average Grade"])
        for student in students:
            writer.writerow([student.name, student.age, student.get_avera


def load_students_from_csv(filename):
    """Reading from CSV"""
    import csv

    students = []
    try:
        with open(filename, "r") as file:
            reader = csv.DictReader(file)
            for row in reader:
                students.append(row)
    except FileNotFoundError:
        print("File not found!")
    return students


# Using the classes and functions
s1 = Student("Alice", 20)
s1.add_grade("Math", 85)
s1.add_grade("Science", 90)

s2 = Student("Bob", 22)
s2.add_grade("Math", 75)
s2.add_grade("Science", 80)

print(f"{s1.name}'s average: {s1.get_average()}")
save_students_to_csv(Student.get_all_students(), "students.csv")
```

Day 30: Final Practice & Preparation

```python
# Practice combining multiple concepts

import sqlite3
import csv
import re
from datetime import datetime

class TaskManager:
    """A complete task management system"""

    def __init__(self, db_name="tasks.db"):
        self.conn = sqlite3.connect(db_name)
        self.cursor = self.conn.cursor()
        self._create_table()

    def _create_table(self):
        self.cursor.execute("""
            CREATE TABLE IF NOT EXISTS tasks (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                description TEXT,
                due_date TEXT,
                completed INTEGER DEFAULT 0,
                created_at TEXT
            )
        """)
        self.conn.commit()

    def add_task(self, title, description="", due_date=None):
        created_at = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.cursor.execute("""
            INSERT INTO tasks (title, description, due_date, created_at)
            VALUES (?, ?, ?, ?)
        """, (title, description, due_date, created_at))
        self.conn.commit()
        print(f"Task '{title}' added successfully!")

    def get_all_tasks(self):
        self.cursor.execute("SELECT * FROM tasks")
        return self.cursor.fetchall()

    def complete_task(self, task_id):
        self.cursor.execute(
            "UPDATE tasks SET completed = 1 WHERE id = ?",
```

```
            (task_id,)
        )
        self.conn.commit()

    def delete_task(self, task_id):
        self.cursor.execute("DELETE FROM tasks WHERE id = ?", (task_id,))
        self.conn.commit()

    def export_to_csv(self, filename="tasks_export.csv"):
        tasks = self.get_all_tasks()
        with open(filename, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerow(["ID", "Title", "Description", "Due Date", "C
            writer.writerows(tasks)
        print(f"Tasks exported to {filename}")

    def search_tasks(self, pattern):
        tasks = self.get_all_tasks()
        matched = [t for t in tasks if re.search(pattern, t[1], re.IGNORE
        return matched

    def close(self):
        self.conn.close()


# Usage example
if __name__ == "__main__":
    manager = TaskManager()

    manager.add_task("Learn Python", "Complete the course", "2024-12-31")
    manager.add_task("Build Projects", "Create 5 mini projects")

    print("\nAll Tasks:")
    for task in manager.get_all_tasks():
        print(task)

    manager.export_to_csv()
    manager.close()
```

📅 After Month Completion: Final Projects

Once you've completed the 30-day plan, tackle these projects to solidify your learning:

Project 1: TO-DO CLI App 📝

Skills Used: Functions, File Handling, Loops, Conditionals

```
# Features to implement:
# - Add, view, update, delete tasks
# - Save tasks to a file
# - Mark tasks as complete
# - Filter tasks by status
```

Project 2: Password Generator 🔐

Skills Used: Random module, Strings, Functions, OOP

```
# Features to implement:
# - Generate passwords of custom length
# - Include/exclude special characters, numbers
# - Password strength checker
# - Save passwords securely
```

Project 3: App for Fetching Data from a Website 🌐

Skills Used: Web Scraping, BeautifulSoup, Requests, CSV

```
# Features to implement:
# - Scrape data from a website
# - Parse and clean the data
# - Save to CSV or database
# - Handle errors gracefully
```

Project 4: Alarm Clock App ⏰

Skills Used: Datetime, Threading, OS module, OOP

```
# Features to implement:
# - Set multiple alarms
# - Play sound when alarm triggers
```

```
# - Snooze functionality
# - Save alarms to file
```

Project 5: PDF Merger App 📄

Skills Used: PyPDF2 library, File Handling, OOP, Error Handling

```
# Install: pip install PyPDF2

# Features to implement:
# - Merge multiple PDFs
# - Split PDF into pages
# - Extract text from PDF
# - Simple GUI (optional with tkinter)
```

📊 Daily Study Schedule

Time Block	Activity	Duration
Session 1	Learn new concepts (reading/videos)	1-1.5 hours
Session 2	Hands-on coding practice	1.5-2 hours
Session 3	Review & mini exercises	30-45 mins

Total: 3-4 hours daily

✅ Weekly Milestones Checklist

- ☐ **Week 1:** Master Loops, Functions, Lambda
- ☐ **Week 2:** Complete OOP concepts
- ☐ **Week 3:** File Handling, Modules, Regex, SQLite
- ☐ **Week 4:** Web Scraping, Pandas, Testing

💡 Pro Tips

1. **Type every code snippet** - Don't just read, practice!
2. **Experiment** - Modify examples and see what happens

3. **Build small projects** - Apply concepts immediately
4. **Use Git** - Track your progress and build a portfolio
5. **Take breaks** - Avoid burnout with the Pomodoro technique
6. **Ask questions** - Join Python communities when stuck