Python Notes Book

A concise guide to understanding and using Python for general-purpose programming.

Table of Contents

Introduction to Python

Python is a high-level, interpreted, and general-purpose programming language known for its readability and versatility. Created by Guido van Rossum, it supports multiple paradigms like procedural, object-oriented, and functional programming.

- **Key Features**:
    - Simple and readable syntax
    - Extensive standard library
    - Cross-platform compatibility
    - Strong community and ecosystem (e.g., PyPI)
- **Use Case**: Web development, data analysis, automation, machine learning, and more.

Core Concepts

Variables and Data Types

Python is dynamically typed; no need to declare variable types.

```
name = "Alice"  # String
age = 25        # Integer
price = 19.99   # Float
is_active = True  # Boolean
```

Control Flow

Use `if`, `for`, and `while` for decision-making and looping.

```
# If statement
if age >= 18:
    print("Adult")
else:
    print("Minor")

# For loop
for i in range(5):
    print(i)  # Outputs 0 to 4
```

Exception Handling

Handle errors gracefully with `try-except`.

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

Data Structures

Python provides built-in data structures for efficient data management.

Lists

Ordered, mutable collections.

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")  # Add item
print(fruits[1])  # Outputs: banana
```

Dictionaries

Key-value pairs.

```
person = {"name": "Alice", "age": 25}
print(person["name"])  # Outputs: Alice
```

Tuples and Sets

- **Tuples**: Immutable sequences (`t = (1, 2, 3)`).
- **Sets**: Unordered, unique elements (`s = {1, 2, 3}`).

Functions and Modules

Functions

Define reusable code blocks with `def`.

```
def greet(name):
    return f"Hello, {name}!"

print(greet("Alice"))  # Outputs: Hello, Alice!
```

Modules

Organize code into files and import them.

```
# math_utils.py
def add(a, b):
    return a + b

# main.py
import math_utils
print(math_utils.add(2, 3))  # Outputs: 5
```

Lambda Functions

Anonymous, one-line functions.

```
double = lambda x: x * 2
print(double(5))  # Outputs: 10
```

Object-Oriented Programming

Python supports OOP with classes and objects.

Class Example

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

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

dog = Dog("Buddy")
print(dog.bark())  # Outputs: Buddy says Woof!
```

- **Inheritance**: Extend classes to reuse code.
- **Encapsulation**: Use _ or __ for private attributes.

Best Practices

1. **Follow PEP 8**: Adhere to Python’s style guide for readable code.
2. **Use Virtual Environments**: Isolate project dependencies (`python -m venv env`).
3. **Write Docstrings**: Document functions and classes for clarity.
4. **Avoid Global Variables**: Prefer passing parameters to functions.
5. **Use List Comprehensions**: For concise and readable loops.

6. `squares = [x**2 for x in range(5)]  # [0, 1, 4, 9, 16]`
7. **Test Code**: Use `unittest` or `pytest` for reliable code.

Example: Simple Task Manager

Below is a simple command-line task manager using Python.

```
# task_manager.py
tasks = []

def add_task(task):
    tasks.append({"task": task, "completed": False})
    print(f"Added: {task}")

def view_tasks():
    if not tasks:
        print("No tasks!")
        return
    for i, task in enumerate(tasks, 1):
        status = "✓" if task["completed"] else " "
        print(f"{i}. [{status}] {task['task']}")

def complete_task(index):
    if 1 <= index <= len(tasks):
        tasks[index-1]["completed"] = True
        print(f"Completed: {tasks[index-1]['task']}")
    else:
        print("Invalid task number!")

# Main loop
while True:
    print("\n1. Add Task\n2. View Tasks\n3. Complete Task\n4. Exit")
    choice = input("Choose an option (1-4): ")

    if choice == "1":
        task = input("Enter task: ")
        add_task(task)
    elif choice == "2":
        view_tasks()
    elif choice == "3":
        view_tasks()
        try:
            index = int(input("Enter task number to complete: "))
            complete_task(index)
        except ValueError:
            print("Please enter a valid number!")
    elif choice == "4":
        print("Goodbye!")
        break
    else:
        print("Invalid option!")
```

Steps to Use

1. Save the code as `task_manager.py`.

2. Run it: `python task_manager.py`.
3. Follow the menu to add, view, or complete tasks.

Additional Resources

- **Official Docs**: python.org
- **Tutorials**: Real Python, Corey Schafer’s YouTube channel
- **Community**: Python Discord, Stack Overflow