Python Notes

1. Introduction to Python

Python is an easy-to-learn, powerful, and versatile programming language. It is known for its simplicity and readability. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

---

2. Python Basics

2.1 Python Syntax

- **Variables**: Used to store values.

  ```
  python
  Copy code
  x = 10  # Integer
  name = "John"  # String
  is_active = True  # Boolean
  ```

- **Data Types**: Python has several built-in data types, including integers, floats, strings, lists, tuples, sets, and dictionaries.

2.2 Data Types and Variables

- **String**: Sequence of characters enclosed in single or double quotes.

  ```
  python
  Copy code
  name = "Alice"
  greeting = 'Hello, world!'
  ```

- **List**: Ordered collection of items.

  ```
  python
  Copy code
  fruits = ['apple', 'banana', 'cherry']
  ```

- **Tuple**: Immutable ordered collection of items.

  ```
  python
  Copy code
  coordinates = (10, 20)
  ```

- **Set**: Unordered collection of unique items.

  ```
  python
  Copy code
  colors = {'red', 'green', 'blue'}
  ```

- **Dictionary**: Key-value pairs.

  ```
  python
  Copy code
  person = {'name': 'John', 'age': 30}
  ```

2.3 Operators

- **Arithmetic Operators**: `+`, `-`, `*`, `/`, `//`, `%`, `**`

  ```
  python
  Copy code
  x = 10
  y = 3
  result = x + y  # 13
  ```

- **Comparison Operators**: `==`, `!=`, `>`, `<`, `>=`, `<=`

  ```
  python
  Copy code
  x = 10
  y = 5
  result = x > y  # True
  ```

- **Logical Operators**: `and`, `or`, `not`

  ```
  python
  Copy code
  x = True
  y = False
  result = x and y  # False
  ```

- **Assignment Operators**: `=`, `+=`, `-=`, `*=`, `/=`

  ```
  python
  Copy code
  x = 5
  x += 3  # x = 8
  ```

---

3. Control Flow

3.1 If-Else Statements

```
python
Copy code
age = 18
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

3.2 For Loops

```
python
Copy code
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

3.3 While Loops

```
python
Copy code
count = 0
while count < 5:
    print(count)
    count += 1
```

3.4 Break and Continue

- **break**: Exits the loop.
- **continue**: Skips the current iteration and moves to the next one.

```
python
Copy code
for i in range(5):
    if i == 3:
        break
    print(i)  # Output: 0 1 2
```

---

4. Functions

4.1 Defining Functions

```
python
Copy code
def greet(name):
    return f"Hello, {name}!"
print(greet("Alice"))  # Output: Hello, Alice!
```

4.2 Arguments and Return Values

```
python
Copy code
def add(a, b):
    return a + b
result = add(2, 3)  # 5
```

4.3 Lambda Functions

A small anonymous function defined using the `lambda` keyword.

```
python
Copy code
square = lambda x: x * x
print(square(4))  # Output: 16
```

---

5. Object-Oriented Programming (OOP)

5.1 Classes and Objects

- **Creating a Class**

```
python
Copy code
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def drive(self):
        print(f"The {self.brand} {self.model} is driving!")

car = Car("Toyota", "Corolla")
car.drive()  # Output: The Toyota Corolla is driving!
```

- **Attributes and Methods**: Attributes are variables inside a class, and methods are functions that belong to a class.

5.2 Inheritance

Inheritance allows one class to inherit the properties and methods of another class.

```
python
Copy code
class ElectricCar(Car):
    def __init__(self, brand, model, battery_size):
        super().__init__(brand, model)
        self.battery_size = battery_size

    def charge(self):
        print("Charging the electric car!")

electric_car = ElectricCar("Tesla", "Model S", 100)
electric_car.charge()  # Output: Charging the electric car!
```

5.3 Polymorphism

Polymorphism allows different classes to define methods with the same name but different behaviors.

```
python
Copy code
class Dog:
    def sound(self):
        return "Bark"

class Cat:
    def sound(self):
        return "Meow"

def animal_sound(animal):
    print(animal.sound())

dog = Dog()
cat = Cat()
animal_sound(dog)  # Output: Bark
animal_sound(cat)  # Output: Meow
```

---

6. Error Handling

6.1 Try-Except Block

Handle exceptions in Python using the `try` and `except` block.

```
python
Copy code
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

6.2 Finally Block

The `finally` block is executed no matter what, even if an exception is raised.

```
python
```

```
Copy code
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print("Execution completed")
```

---

7. Working with Files

7.1 Reading Files

```
python
Copy code
with open('file.txt', 'r') as file:
    content = file.read()
    print(content)
```

7.2 Writing Files

```
python
Copy code
with open('file.txt', 'w') as file:
    file.write("Hello, World!")
```

---

8. Libraries and Modules

8.1 Importing Modules

You can import built-in or external modules to extend Python's functionality.

```
python
Copy code
import math
print(math.sqrt(16))  # Output: 4.0
```

8.2 Creating Your Own Module

Create a file `my_module.py` and define functions or variables inside it. You can import and use it in another file.

```
python
Copy code
# my_module.py
def greet():
    return "Hello, Python!"
```

```
python
Copy code
# main.py
import my_module
print(my_module.greet())  # Output: Hello, Python!
```

---

9. Advanced Topics

9.1 List Comprehensions

A compact way to create lists.

```
python
Copy code
squares = [x ** 2 for x in range(5)]
print(squares)  # Output: [0, 1, 4, 9, 16]
```

9.2 Generators

Generators are functions that return an iterable set of items, one at a time.

```
python
Copy code
def count_up_to(limit):
    count = 1
    while count <= limit:
        yield count
        count += 1

counter = count_up_to(5)
for number in counter:
    print(number)
```

9.3 Decorators

A decorator is a function that takes another function as input and extends its behavior.

```
python
Copy code
def decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper

@decorator
def say_hello():
    print("Hello!")

say_hello()
```

---

10. Conclusion

Python is a versatile language that supports multiple programming paradigms, making it ideal for both beginners and advanced developers. Mastering its syntax, data structures, and advanced features like OOP, decorators, and generators will enable you to write efficient and maintainable code.