Python Notes

1. Introduction to Python

- Python is a **high-level, interpreted, general-purpose programming language**.
- Created by **Guido van Rossum** in 1991.
- Known for **readability, simplicity, and large community support**.
- Supports multiple programming paradigms: **procedural, object-oriented, and functional programming**.
- Widely used in **web development, data science, machine learning, automation, AI, scripting, etc.**

---

2. Variables

- Variables are used to store data values.
- Python is **dynamically typed** (no need to declare type explicitly).
- Example:
- x = 10   # integer
- name = "Sam" # string
- pi = 3.14   # float

---

3. Keywords

- Keywords are **reserved words** in Python.
- Examples: if, else, while, for, def, class, try, except, import, return, with.
- Python 3.x has around **35 keywords**.
- Example:
- if True:
-     print("Hello")

---

4. Data Types and Identifiers

- **Identifiers**: Names given to variables, functions, classes, etc. Must follow rules:
    - Start with a letter or underscore.

    - Cannot use keywords.
    - Case-sensitive.
- **Basic Data Types**:
    - int → integers
    - float → decimals
    - str → strings
    - bool → True/False
    - complex → complex numbers (e.g., 3+4j)

---

5. Data Structures in Python

- **List** → ordered, mutable, allows duplicates
- lst = [1, 2, 3, "Python"]
- **Tuple** → ordered, immutable
- tup = (10, 20, 30)
- **Set** → unordered, unique values
- st = {1, 2, 3, 3} # {1, 2, 3}
- **Dictionary** → key-value pairs
- d = {"name": "Sam", "age": 20}

---

6. Operators in Python

- **Arithmetic**: +, -, *, /, %, //, **
- **Comparison**: ==, !=, <, >, <=, >=
- **Logical**: and, or, not
- **Assignment**: =, +=, -=, etc.
- **Bitwise**: &, |, ^, <<, >>
- **Membership**: in, not in
- **Identity**: is, is not

---

7. Control Flow Statements

- **Conditional**:
- if x > 0:
-     print("Positive")
- elif x == 0:
-     print("Zero")
- else:
-     print("Negative")
- **Loops**:
    - for loop (iterates over sequence)
    - while loop (runs until condition false)
- **Loop control**: break, continue, pass

---

8. Input and Output

- Input:
- name = input("Enter your name: ")
- Output:
- print("Hello", name)
- print(f"Welcome {name}") # f-string formatting

---

9. Functions

- Defined using def keyword.
- Can take arguments and return values.
- def add(a, b):
-     return a + b
- print(add(5, 3))
- Supports default, keyword, and variable-length arguments.

---

10. Object-Oriented Programming (OOP) in Python

- Supports **classes and objects**.
- Key concepts:
    - **Class**: Blueprint for objects.
    - **Object**: Instance of class.
    - **Inheritance**: Reuse properties.
    - **Polymorphism**: Same method, different behavior.
    - **Encapsulation**: Hiding internal details.
- Example:

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
    def display(self):
        print(self.brand, self.model)

c = Car("Tesla", "Model S")
c.display()
```

---

11. Exception Handling

- Used to handle runtime errors.
- Keywords: try, except, finally, raise.
- Example:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Division by zero not allowed")
finally:
```

-     print("Execution completed")

---

12. Python Comprehensions

- Shorter syntax for creating sequences.
- **List comprehension**:
- squares = [x**2 for x in range(5)]
- **Set comprehension**:
- s = {x for x in range(5)}
- **Dictionary comprehension**:
- d = {x: x**2 for x in range(5)}

---

13. Decorators

- Special functions that modify other functions or methods.
- Commonly used in **logging, authentication, performance measurement**.
- Example:
- def decorator(func):
-     def wrapper():
-         print("Before function")
-         func()
-         print("After function")
-     return wrapper
-
- @decorator
- def greet():
-     print("Hello")
-
- greet()

---

14. Functional Programming Tools

- **Lambda functions**: Anonymous one-liner functions.
- square = lambda x: x**2
- print(square(5))
- **map()**: Apply function to iterable.
- nums = [1, 2, 3]
- result = list(map(lambda x: x*2, nums))
- **filter()**: Filter elements based on condition.
- evens = list(filter(lambda x: x%2==0, nums))
- **reduce()** (from functools): Apply function cumulatively.
- from functools import reduce

total = reduce(lambda a,b: a+b, nums)