Python Tutorial for Beginners

This tutorial covers core Python concepts, object-oriented programming, and working with external files. Each section includes explanations, examples, flow diagrams, and practice questions.

---

1. Python Fundamentals

1.1 Variables and Data Types

Variables store data. Data types define the kind of data a variable holds.

```
name = "John"          # String
grade = 9              # Integer
marks = 85.5           # Float
is_passed = True       # Boolean
```

Flow Diagram:

```
Start -> Declare Variable -> Assign Value -> End
```

Practice:

- Create variables to store your name, age, and grade.
- Print their data types using `type()`.

1.2 Operators and Expressions

Operators perform operations on variables.

```
a = 10
b = 5
print(a + b)  # Addition
print(a > b)  # Comparison
```

Practice:

- Use all arithmetic operators.
- Try comparison and logical operators.

1.3 Conditional Statements

Make decisions using `if`, `elif`, and `else`.

```
age = 18
if age >= 18:
    print("Adult")
elif age > 13:
    print("Teen")
else:
    print("Child")
```

Flow Diagram:

```
Start -> Check Condition -> True? -> Execute Block -> End
```

Practice:

- Write a program to check if a number is positive, negative, or zero.

1.4 Loops

Repeat code using `for` and `while`.

```
# For loop
for i in range(5):
    print(i)

# While loop
count = 0
while count < 5:
    print(count)
    count += 1
```

Practice:

- Print numbers from 1 to 10 using both loops.

1.5 Functions and Recursion

Encapsulate logic in functions.

```python
def greet(name):
    print(f"Hello {name}")

greet("Alice")

# Recursion
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

Flow Diagram:

```
Start -> Call Function -> Execute Block -> Return Value -> End
```

Practice:

- Write a function to calculate the square of a number.
- Implement factorial using recursion.

1.6 Data Structures

List, Tuple, Dict, Set

```python
my_list = [1, 2, 3]
my_tuple = (4, 5, 6)
my_dict = {"name": "John", "age": 30}
my_set = {1, 2, 3, 2}
```

Practice:

- Add/remove elements from each data structure.
- Access elements using loops.

1.7 String Manipulation

```python
text = "hello"
print(text.upper())
print(text[::-1])  # Reverse
```

Practice:

- Check if a string is a palindrome.
- Count vowels in a string.

1.8 Exception Handling

```
try:
    x = int(input("Enter number: "))
    print(100 / x)
except ZeroDivisionError:
    print("Can't divide by 0")
except ValueError:
    print("Invalid input")
```

Practice:

- Handle errors for division and string conversion.

1.9 File I/O

```
# Writing
with open("sample.txt", "w") as f:
    f.write("Hello")

# Reading
with open("sample.txt", "r") as f:
    print(f.read())
```

Practice:

- Write user input to a file and read it.

---

2. Object-Oriented Programming (OOP)

2.1 Classes and Objects

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

```
    def greet(self):
        print(f"Hello {self.name}")

p = Person("John")
p.greet()
```

Flow Diagram:

```
Create Class -> Instantiate Object -> Call Method
```

2.2 Inheritance and Polymorphism

```
class Animal:
    def speak(self):
        print("Animal sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

obj = Dog()
obj.speak()
```

2.3 Encapsulation and Abstraction

```
class BankAccount:
    def __init__(self):
        self.__balance = 0  # private

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance
```

Practice:

- Create a `Vehicle` class and inherit `Car` and `Bike`.
- Add private attributes and access them using methods.

3. Working with External Files

3.1 CSV

```
import csv
with open("data.csv", "r") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

3.2 JSON

```
import json
data = {"name": "Alice", "age": 25}

with open("data.json", "w") as f:
    json.dump(data, f)

with open("data.json", "r") as f:
    print(json.load(f))
```

3.3 XML

```
import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')
root = tree.getroot()
for child in root:
    print(child.tag, child.text)
```

3.4 Config Files (INI, YAML)

```
# INI
from configparser import ConfigParser
parser = ConfigParser()
parser.read("config.ini")
print(parser.get("settings", "username"))

# YAML
import yaml
with open("config.yaml") as f:
```

```
    config = yaml.safe_load(f)
    print(config['username'])
```

3.5 Logging

```
import logging
logging.basicConfig(level=logging.INFO)
logging.info("This is an info message")
```

Practice:

- Parse JSON and print values.
- Read from CSV and write to JSON.
- Configure and use logging.

This guide provides a solid foundation to master Python for development and automation.