Topic 1: Introduction to Python and Setting Up Your Environment

Why This Matters

Before writing code, you need to understand what Python is and how to run it. This is like learning how to turn on a car before driving.

Explanation

Python is a programming language. It allows you to give instructions to a computer in a way that is close to human language.
Python is used for websites, data analysis, automation, artificial intelligence, and more.

To use Python, you need:

1. **Python installed** on your computer
2. A place to **write code** (editor)
3. A way to **run the code**

Recommended setup for beginners:

- Install Python from python.org
- Use **VS Code** or **PyCharm**
- Make sure Python is added to PATH

You can check installation by typing:

```
python --version
```

Code Examples

```
# This is a Python comment
# Comments explain code and are ignored by Python

# Python runs code from top to bottom
print("Python is ready")
```

Practice Exercises

1. Open a Python editor.
2. Run a Python file with one line of code.
3. Write a comment explaining what Python is.

Solutions

Solution to Exercise 1

```
print("Hello Python")
```

Solution to Exercise 2

```
print("This file is running")
```

Solution to Exercise 3

```
# Python is a programming language used to give instructions to computers
print("Learning Python")
```

Topic 2: Print Statement and Basic Output

Why This Matters

Printing lets you see results. Without it, you would not know what your program is doing.

Explanation

`print()` tells Python to show something on the screen.
It is like speaking out loud.

Code Examples

```
print("Hello, world")
```

```
print(10)
```

```
print("Age:", 25)
```

Practice Exercises

1. Print your name.
2. Print your age.
3. Print a sentence using print().

Solutions

```
print("John")
```

```
print(30)
```

```
print("I am learning Python")
```

Topic 3: Variables and Data Types (int, float, str, bool)

Why This Matters

Variables store information. This allows programs to remember things.

Explanation

A variable is a labeled box holding data.

Types:

- `int` → whole numbers
- `float` → decimal numbers
- `str` → text
- `bool` → True or False

Code Examples

```
age = 25
print(age)

price = 19.99
print(price)

name = "Alice"
is_student = True
```

Practice Exercises

1. Create a variable for your name.
2. Create a variable for your height.
3. Create a True/False variable.

Solutions

```
name = "Sarah"

height = 1.75

is_logged_in = False
```

**Topic 4: Basic Operations (+, -, *, /, //, %,)

Why This Matters

Programs often calculate values.

Explanation

Python can do math just like a calculator.

Code Examples

```
print(10 + 5)
print(10 / 3)

print(10 // 3)

print(2 ** 3)
```

Practice Exercises

1. Add two numbers.
2. Divide two numbers.
3. Find the remainder of 17 ÷ 5.

Solutions

```
print(4 + 6)

print(20 / 4)

print(17 % 5)
```

Topic 5: Getting User Input with `input()`

Why This Matters

Input allows users to interact with your program.

Explanation

`input()` pauses the program and waits for the user to type.

Code Examples

```
name = input("Enter your name: ")
print(name)

age = int(input("Enter age: "))
print(age)
```

Practice Exercises

1. Ask for a name and print it.
2. Ask for a number and print it.
3. Ask for two numbers and add them.

Solutions

```
name = input("Name: ")
print("Hello", name)

number = int(input("Number: "))
```

```
print(number)

a = int(input("First: "))
b = int(input("Second: "))
print(a + b)
```

Topic 6: Strings and String Methods

Why This Matters

Text handling is everywhere in programs.

Explanation

Strings are text inside quotes.
Methods are built-in string actions.

Code Examples

```
text = "hello"
print(text.upper())

print("Python".lower())

print("Hello World".replace("World", "Python"))
```

Practice Exercises

1. Convert text to uppercase.
2. Count letters in a string.
3. Replace a word in a sentence.

Solutions

```
print("hi".upper())

print(len("Python"))

print("I like Java".replace("Java", "Python"))
```

Topic 7: Conditionals (if, elif, else)

Why This Matters

Conditionals let programs make decisions.

Explanation

If something is true, do this. Otherwise, do something else.

Code Examples

```
age = 18
if age >= 18:
    print("Adult")

if age < 18:
    print("Minor")
else:
    print("Adult")
```

Practice Exercises

1. Check if a number is positive.
2. Check if a person can vote.
3. Grade system using elif.

Solutions

```
num = 5
if num > 0:
    print("Positive")

age = 20
if age >= 18:
    print("Can vote")

score = 75
if score >= 90:
    print("A")
elif score >= 60:
    print("B")
else:
    print("C")
```

Topic 8: Loops – while loop

Why This Matters

Loops repeat tasks automatically.

Explanation

A while loop runs while a condition is true.

Code Examples

```
count = 1
while count <= 5:
    print(count)
    count += 1
```

Practice Exercises

1. Print numbers 1–5.
2. Countdown from 10.
3. Ask until user types "yes".

Solutions

```
i = 1
while i <= 5:
    print(i)
    i += 1

i = 10
while i > 0:
    print(i)
    i -= 1

answer = ""
while answer != "yes":
    answer = input("Type yes: ")
```

Topic 9: Loops – for loop and range()

Why This Matters

For-loops are simpler for counting and lists.

Explanation

A for-loop repeats for each item.

Code Examples

```
for i in range(5):
    print(i)

for i in range(1, 6):
    print(i)
```

Practice Exercises

1. Print numbers 1–10.
2. Print even numbers.
3. Sum numbers 1–100.

Solutions

```
for i in range(1, 11):
    print(i)

for i in range(2, 11, 2):
    print(i)

total = 0
for i in range(1, 101):
    total += i
print(total)
```

Topic 10: Lists

Why This Matters

Lists store multiple values in one place.

Explanation

Lists are like shopping lists.

Code Examples

```
numbers = [1, 2, 3]
print(numbers)

numbers.append(4)
```

Practice Exercises

1. Create a list of fruits.
2. Add a fruit.
3. Print each fruit.

Solutions

```
fruits = ["apple", "banana"]

fruits.append("orange")

for fruit in fruits:
    print(fruit)
```

Topic 11: Tuples and Sets

Why This Matters

They store data differently from lists.

Explanation

- Tuple: cannot change

- Set: no duplicates

Code Examples

```
point = (10, 20)

colors = {"red", "blue", "red"}
print(colors)
```

Practice Exercises

1. Create a tuple.
2. Create a set.
3. Remove duplicates from a list using set.

Solutions

```
coords = (5, 6)

nums = {1, 2, 2, 3}

unique = list(set([1, 1, 2, 3]))
```

Topic 12: Dictionaries

Why This Matters

Dictionaries store labeled data.

Explanation

They work like a phone book.

Code Examples

```
person = {"name": "John", "age": 30}

print(person["name"])
```

Practice Exercises

1. Create a dictionary.
2. Add a key.
3. Print values.

Solutions

```
car = {"brand": "Toyota"}

car["year"] = 2022
```

```
for value in car.values():
    print(value)
```

Topic 13: Functions – Defining and Calling

Why This Matters

Functions reuse code.

Explanation

A function is a reusable action.

Code Examples

```
def greet():
    print("Hello")

greet()
```

Practice Exercises

1. Create a function.
2. Call it.
3. Print a message.

Solutions

```
def say_hi():
    print("Hi")
say_hi()
```

Topic 14: Functions – Parameters and Return

Why This Matters

Functions become flexible.

Explanation

Parameters are inputs.
Return sends back results.

Code Examples

```
def add(a, b):
    return a + b
```

Practice Exercises

1. Add two numbers.
2. Multiply two numbers.
3. Return a full name.

Solutions

```
def add(a, b):
    return a + b

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

def full_name(first, last):
    return first + " " + last
```

Topic 15: Modules and Importing

Why This Matters

Modules organize code.

Explanation

Modules are code files.

Code Examples

```
import math
print(math.sqrt(16))
```

Practice Exercises

1. Import math.
2. Use sqrt.
3. Use random.

Solutions

```
import math

print(math.sqrt(25))

import random
print(random.randint(1, 10))
```

Topic 16: Basic File Handling

Why This Matters

Files store data permanently.

Explanation

You can write to and read from files.

Code Examples

```
file = open("data.txt", "w")
file.write("Hello")
file.close()

file = open("data.txt", "r")
print(file.read())
file.close()
```

Practice Exercises

1. Write text to file.
2. Read file.
3. Append text.

Solutions

```
file = open("test.txt", "w")
file.write("Hi")
file.close()

file = open("test.txt", "r")
print(file.read())
file.close()

file = open("test.txt", "a")
file.write("\nMore text")
file.close()
```

Topic 17: Error Handling with try-except

Why This Matters

Programs should not crash.

Explanation

Errors happen. Try-except handles them.

Code Examples

```
try:
    print(10 / 0)
except:
    print("Error occurred")
```

Practice Exercises

1. Handle division error.
2. Handle input error.
3. Print custom message.

Solutions

```
try:
    x = int(input("Number: "))
except:
    print("Invalid input")
```

Topic 18: Introduction to Simple Projects

Why This Matters

Projects combine everything.

Explanation

Small projects build confidence.

Code Examples

```
def calculator():
    a = int(input("A: "))
    b = int(input("B: "))
    print(a + b)
calculator()
```

Practice Exercises

1. Number guessing game.
2. Simple calculator.
3. To-do list (list-based).

Solutions

```
import random
number = random.randint(1, 5)
guess = int(input("Guess: "))
if guess == number:
    print("Correct")
```

Final Words

Python is learned through practice, not memorization. Write code every day, even if it is small. Break things, fix them, and keep going. Start with tiny programs and gradually build bigger ones. Every professional developer started exactly where you are now. Keep learning, keep building, and stay consistent.