Assignment 1

❖ For each code snippet below Identify Error, Rewrite and Explain code.

1)
```
def add_num(a,b)
        return a+b
print(add_num(5,10))
```

Identify : In this code the SyntaxError: expected ':' error is accure.

Rewrite :
```
def add_num(a,b):
        return a+b
print(add_num(5,10))
```

Explain : The code works for to perform addition of two number using add_num() function that has two argument. The error is arise because of (:) colon. That is use to define the next code is in indentation.

2)
```
name= 'Alice
print("Hello, "+name)
```

Identify : Missing Closing Quote in name= 'Alice

Rewrite :
```
name= 'Alice'

print("Hello, "+name)
```

Explain : String literals in Python must begin and end with the same quote character. By adding the missing (‘) after Alice, the code compiles. Then string concatenation with + produces the output Hello, Alice.

3)
```
for i in range(5):
        print("Number:", i)
```

Identify : No Error

Rewrite : Same Code

Explain : In the given code we print number from 0 to 4 using for loop.

```
Number: 0

Number: 1

Number: 2

Number: 3

Number: 4
```

4) my_list = [1, 2, 3, 4, 5]

```
print("The fifth element is: " + my_list[5])
```

Identify : IndexError: list index out of range

Rewrite :

```
my_list = [1, 2, 3, 4, 5]

print("The fifth element is: " + str(my_list[4]))
```

Explain : Lists in Python use 0-based indexing, so the fifth element is at index 4, not 5. Also, concatenating an integer directly to a string causes a type error, so str() is used for conversion.

5)

```
def greet(name):
   print("Hello " + name)
greet("Bob")
```

Identify : No Error

Rewrite : Same Code

Explain : The function is properly defined and called with "Bob" as the argument. It prints the greeting as intended.

6)

```
age = input("Enter your age: ")
if age >= 18:
   print("You are eligible to vote.")
else:
   print("You are not eligible to vote.")
```

Identify : **Error:** TypeError: '>=' not supported between instances of 'str' and 'int'

Rewrite :

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

if age >= 18:
        print("You are eligible to vote.")
else:
        print("You are not eligible to vote.")
```

Explain : The input() function returns a string, so you must convert it to an integer with int() before comparing it to 18.

7)
```
def multiply(a, b):
            result = a * b
return result
print(multiply(4, 5))
```

Identify : No Error

Rewrite : Same Code

Explain : This function multiplies two numbers and returns the result, which is q correctly printed.

8)
```
count = 10
while count > 0
   print(count)
   count -= 1
print("Countdown complete!")
```

Identify : SyntaxError: expected ':'

Rewrite :
```
count = 10
while count > 0:
            print(count)
            count -= 1
print("Countdown complete!")
```

Explain : while count > 0: keeps looping as long as count remains positive.

Python Overview

Presenter : Pandya Shaunak Rajnikant

Date : 15th July 2025

What Is Python?

- Python is a high-level, interpreted programming language designed for readability and rapid development.
- Clean, expressive syntax minimizes boilerplate
- Supports procedural, object-oriented, and functional paradigms
- Extensive “batteries-included” standard library
- Open-source with a vibrant global community

Origins & Evolution

- Python’s journey began in the late 1980s under Guido van Rossum at CWI in the Netherlands.
- 1989: Guido starts the Python project to improve on ABC language
- 1991: First public release (Python 0.9.0) with functions, modules, exceptions
- 2000: Python 2.0 adds list comprehensions and garbage collection
- 2008: Python 3.0 introduces breaking changes for cleaner syntax
- 2023: Python 3.11/3.12 focus on performance enhancements and typing

Why Python Gained Popularity

- Several factors fueled Python's rapid adoption across domains:
- Readable code accelerates learning and maintenance
- Rich standard library spans web, data, networking, automation
- Strong foothold in data science, machine learning, web frameworks
- Beginner-friendly while scalable for large, complex systems

Python Functions

- Functions are reusable blocks of code that encapsulate logic and improve organization.
- Defined with the def keyword
- Can accept positional, keyword, default, *args, and **kwargs parameters
- Return any Python object (or None by default)
- Docstrings document purpose and usage

Advanced Function Concepts

- Unlock more flexibility and power with advanced patterns:
- Default arguments simplify common use cases
- *args and **kwargs collect variable argument lists
- Lambda functions for concise, anonymous expressions
- Decorators for modifying or extending behavior without altering the original function

Python Modules

Modules are single .py files that group related code into namespaces.
•Import with import module_name or from module_name import name
•Types of modules:
•Built-in (e.g., os, sys)
•Standard Library (e.g., json, dataclasses)
•Third-Party (e.g., requests, numpy)
•Custom (project-specific utilities)

Module Type	Example	Purpose
Built-in	os	System interactions
Standard Lib	json	JSON parsing & serialization
Third-Party	requests	HTTP client library
Custom	mymodule	Your application’s helpers

Creating & Using Custom Modules

- Steps to build and utilize your own Python modules:
- Create a file named mymodule.py alongside your scripts
- Define functions, classes, or constants inside it
- In another script, import:

Packages & Best Practices

- Organize code at scale and follow conventions for maintainability:
- Packages are directories with an __init__.py file, enabling hierarchical namespaces
- Use virtual environments (venv, conda) to isolate dependencies
- Manage libraries via pip and requirements.txt
- Adhere to PEP 8 for naming, formatting, and code style
- Keep functions small and focused; name modules clearly

Thank You