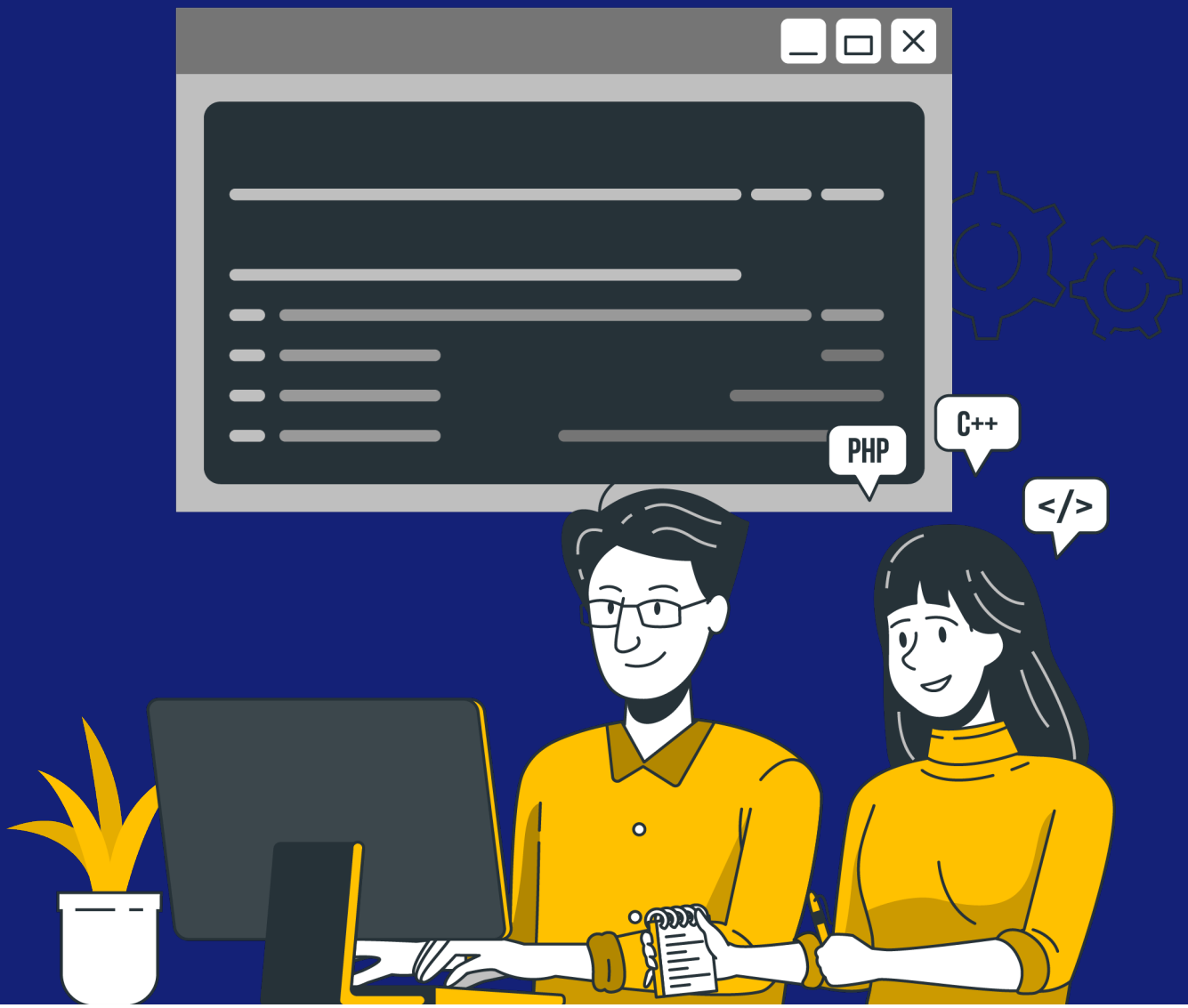
PW SKILLS

Lesson Plan

Best Practice Exception Handling



Exception handling is a critical aspect of writing robust and error-tolerant Python code. Here are some best practices for exception handling along with examples:

1. Specificity in Exception Handling:

Catch specific exceptions rather than using a broad `except` clause.

```Python
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

2. Use of `finally` Block:

Use the `finally` block to ensure that certain code runs whether an exception is raised or not.

```Python
try:
    file = open("example.txt", "r")
    # Some code here
except FileNotFoundError as e:
    print(f"File not found: {e}")
finally:
    file.close()
```

3. Logging Exceptions:

Log exceptions to provide detailed information for debugging.

```Python
import logging
try:
    result = int("abc")
except ValueError as e:
    logging.error(f"Error: {e}")
```

4. Raising Exceptions:

Raise exceptions to indicate errors or unexpected conditions.

```Python
def divide(x, y):
    if y == 0:
        raise ValueError("Cannot divide by zero")
    return x / y
try:
    result = divide(10, 0)
except ValueError as e:
    print(f"Error: {e}")
```

5. Handling Multiple Exceptions:

Handle multiple exceptions in separate except blocks.

```Python
try:
    value = int("abc")
    result = 10 / 0
except ValueError as ve:
    print(f"ValueError: {ve}")
except ZeroDivisionError as ze:
    print(f"ZeroDivisionError: {ze}")
```

6. Custom Exception Classes:

Create custom exception classes to represent specific error conditions.

```Python
class CustomError(Exception):
    pass

try:
    raise CustomError("This is a custom error")
except CustomError as ce:
    print(f"Custom Error: {ce}")
```

7. Avoiding Bare except:

Avoid using a bare except clause as it can catch unexpected exceptions.

```Python
try:
    # Some code that may raise different exceptions
except Exception as e:
    print(f"Caught an unexpected exception: {e}")
```

8. Exception Chaining:

Use from to chain exceptions, preserving the original exception context.

```Python
try:
    result = int("abc")
except ValueError as e:
    raise CustomError("Failed to convert to integer") from e
```