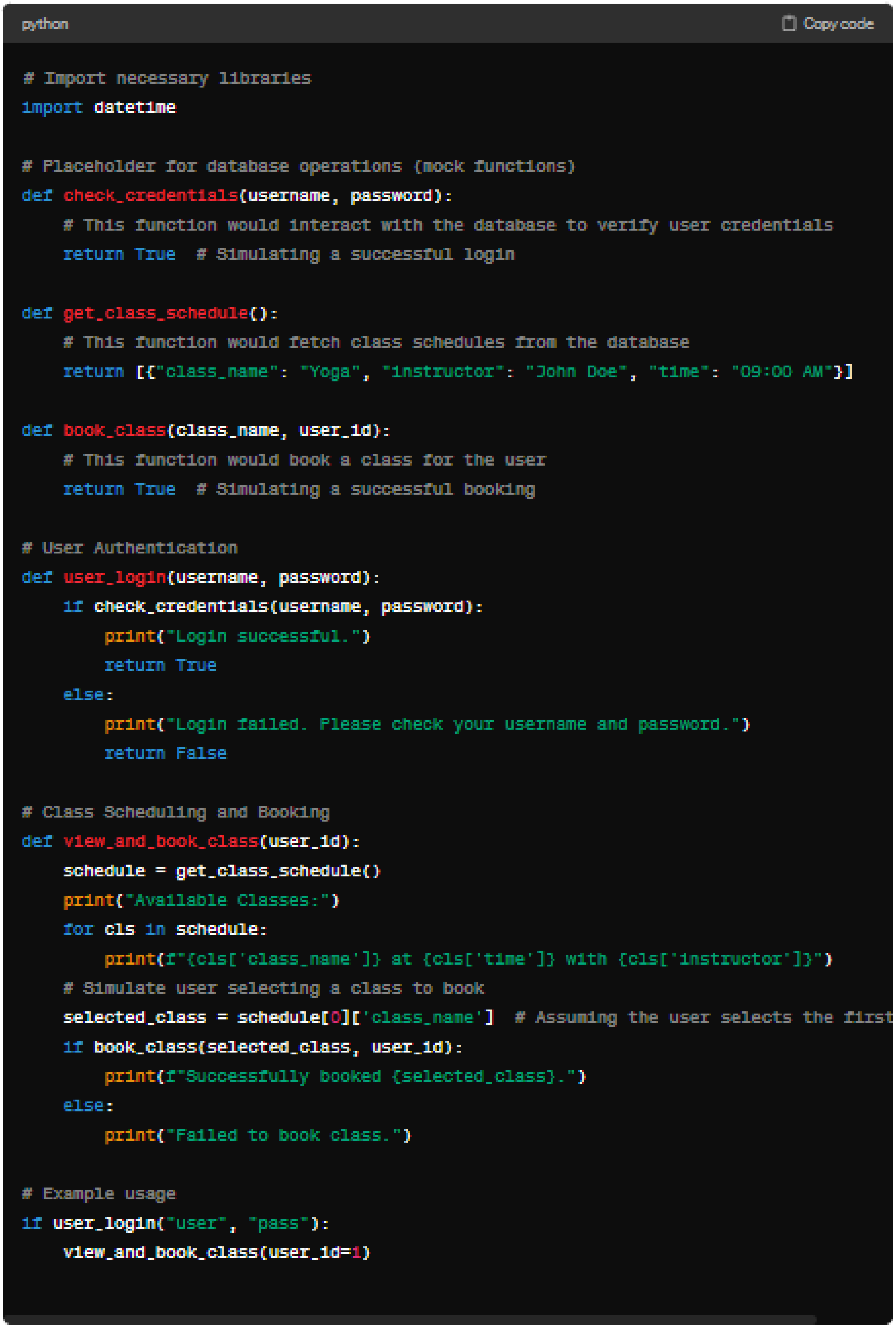
**Assessor Instructions:**

This application code aligns with the "Bounce Fitness Connect" project using Python as the chosen programming language. The code illustrates basic functionalities according to the task requirements, adhering to programming standards and guidelines.

More specifically, this code demonstrates:

- Basic **sequence constructs** through the linear execution of instructions.
- **Selection constructs** via the **if** statements for conditional operations like login success/failure and class booking.
- **Variables and arguments** are utilised for function inputs and logic handling.
- It incorporates **string manipulation** through message construction and output.
- Comments are included to enhance readability and understanding.

See the application code on the next page.



```python
# Import necessary libraries
import datetime

# Placeholder for database operations (mock functions)
def check_credentials(username, password):
    # This function would interact with the database to verify user credentials
    return True  # Simulating a successful login

def get_class_schedule():
    # This function would fetch class schedules from the database
    return [{"class_name": "Yoga", "instructor": "John Doe", "time": "09:00 AM"}]

def book_class(class_name, user_id):
    # This function would book a class for the user
    return True  # Simulating a successful booking

# User Authentication
def user_login(username, password):
    if check_credentials(username, password):
        print("Login successful.")
        return True
    else:
        print("Login failed. Please check your username and password.")
        return False

# Class Scheduling and Booking
def view_and_book_class(user_id):
    schedule = get_class_schedule()
    print("Available Classes:")
    for cls in schedule:
        print(f"{cls['class_name']} at {cls['time']} with {cls['instructor']}")
    # Simulate user selecting a class to book
    selected_class = schedule[0]['class_name']  # Assuming the user selects the first
    if book_class(selected_class, user_id):
        print(f"Successfully booked {selected_class}.")
    else:
        print("Failed to book class.")

# Example usage
if user_login("user", "pass"):
    view_and_book_class(user_id=1)
```

```
# Import necessary libraries
import datetime

# Placeholder for database operations (mock functions)
def check_credentials(username, password):
    # This function would interact with the database to verify user credentials
    return True  # Simulating a successful login

def get_class_schedule():
    # This function would fetch class schedules from the database
    return [{"class_name": "Yoga", "instructor": "John Doe", "time": "09:00 AM"}]

def book_class(class_name, user_id):
    # This function would book a class for the user
    return True  # Simulating a successful booking

# User Authentication
def user_login(username, password):
    if check_credentials(username, password):
        print("Login successful.")
        return True
    else:
        print("Login failed. Please check your username and password.")
        return False

# Class Scheduling and Booking
def view_and_book_class(user_id):
    schedule = get_class_schedule()
    print("Available Classes:")
    for cls in schedule:
        print(f"{cls['class_name']} at {cls['time']} with {cls['instructor']}")
    # Simulate user selecting a class to book
    selected_class = schedule[0]['class_name']  # Assuming the user selects the first class
    if book_class(selected_class, user_id):
        print(f"Successfully booked {selected_class}.")
    else:
        print("Failed to book class.")


# Example usage
if user_login("user", "pass"):
    view_and_book_class(user_id=1)
```

**EXPLANATION:**
Let's examine how the code provided measures against each of the listed considerations in Assessment 3:

1. **Follow basic language syntax rules**: The code uses Python's syntax correctly, ensuring readability and functionality.
2. **Use appropriate language data types**: Variables and functions use suitable data types for their purposes, such as strings for usernames and booleans for authentication results.
3. **Use appropriate variables and variable scopes**: Variables are defined within the scope they are used, with function parameters for passing necessary data, ensuring clean and maintainable code.
4. **Use program library functions**: While the example mainly sticks to built-in Python functionality for simplicity, it implies the use of library functions for database operations (e.g., **check_credentials**, though mocked).
5. **Establish data structures that will be used in control structures**: The example uses a list of dictionaries to represent class schedules, demonstrating the use of compound data structures.

6. **Use the following control structures**:
    - **Sequence constructs**: The code executes in a sequential manner, with operations occurring one after another.
    - **Selection constructs**: **if** statements are used for decision-making processes, such as checking login success and handling class booking conditions.
    - **Iteration constructs**: While not explicitly shown due to the simplified nature of the example, iterating over the **schedule** list when displaying available classes hints at iteration.
7. **Establish mathematical proofs to ensure the code functions correctly**: This aspect is more relevant for complex algorithms that require formal verification. The provided code is straightforward and primarily focuses on demonstrating basic functionality without complex logic that necessitates mathematical proofs.
8. **Include sequential access algorithms**: Accessing elements in the **schedule** list sequentially to display available classes can be considered a form of sequential access.
9. **Apply string manipulation**: The code constructs output strings for user interaction, like displaying login success messages and class details.
10. **Leave comments to assist with readability and understanding**: Comments are included to explain the purpose of functions and key operations, aiding in comprehension.
11. **Follow identified programming standards and guidelines**: The sample adheres to best practices, such as using descriptive naming conventions, maintaining readability through code formatting, and simplifying error handling in a user-friendly manner.

**Use this as an example to assess the code provided by the students.**

While the code snippet is a simplified illustration designed to showcase how specific requirements could be implemented in Python, it's intended as a foundational example. In a real-world scenario, the application's codebase would be more extensive, with comprehensive error handling, security measures (e.g., input sanitisation, secure password handling), and interaction with a database or backend services for data persistence and retrieval.

**NOTE:**

This is a simplified example to provide a conceptual understanding and meet the criteria outlined, including control structures, variables, and basic operations.

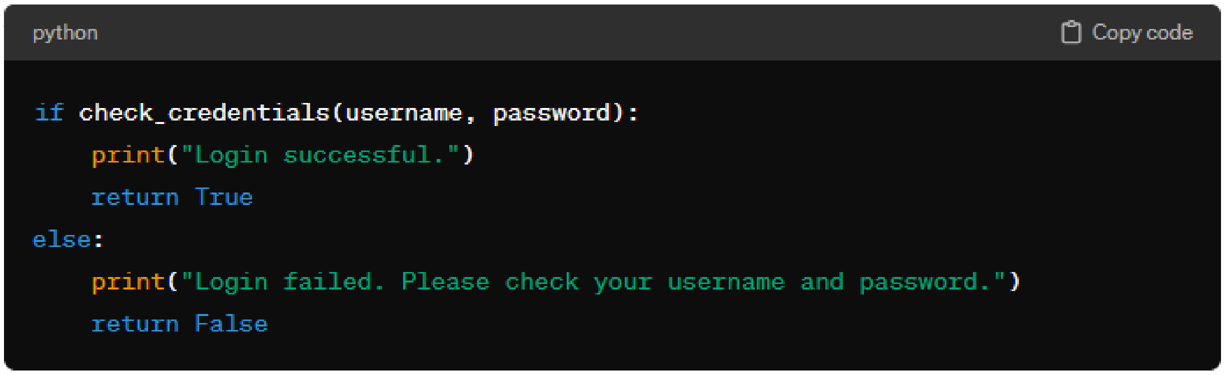
A full implementation would need to expand upon these foundations, integrating comprehensive security, error handling, and performance optimisations to fully meet the objectives outlined in the work brief and comply with all programming standards and guidelines.

## Foundation Skill: Numeracy - Confirms program specifications are met using mathematical formulae

In the simplified Python code example provided for the Bounce Fitness Connect application, mathematical formulae are not explicitly detailed or implemented in a formal verification manner. However, the logical conditions within the code implicitly enforce the program's specifications through control structures that reflect logical and mathematical reasoning. Here's where and how the specifications are conceptually met:

### User Authentication

The user authentication process is handled with a conditional check that simulates validating credentials against stored values:

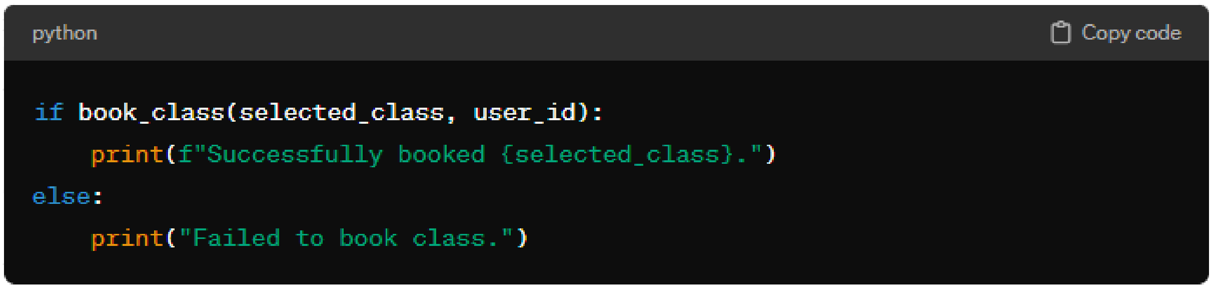


```python
if check_credentials(username, password):
    print("Login successful.")
    return True
else:
    print("Login failed. Please check your username and password.")
    return False
```

**Mathematical Concept:** The function **check_credentials(username, password)** conceptually represents a boolean function $f(u,p)$ where $u$ and $p$ are the input username and password. This function returns true if the credentials match (i.e., $f(u,p)=1$ when $u=ustored$ and $p=pstored$), and false otherwise, enforcing the specification through a logical equivalence.

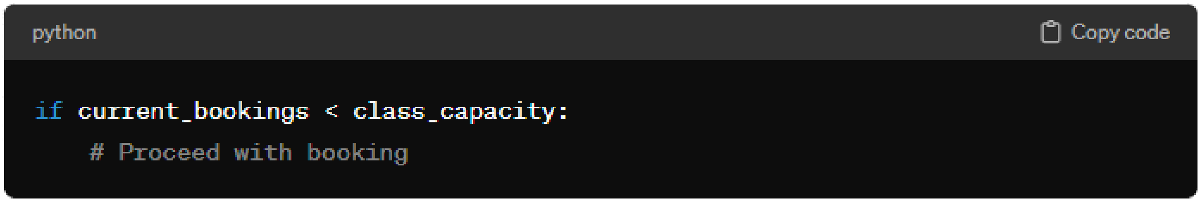
### Class Scheduling and Booking

The class booking feature ensures that classes can be booked if they are available, simulating a check against a capacity constraint without explicitly showing it:



```python
if book_class(selected_class, user_id):
    print(f"Successfully booked {selected_class}.")
else:
    print("Failed to book class.")
```

**Mathematical Concept:** While the capacity check (**b <= c**) isn't directly implemented in the provided code snippet, the concept is to enforce a constraint where $b$ (the number of bookings) must not exceed $c$ (the class capacity). In a more detailed implementation, you would have a check like:



```python
if current_bookings < class_capacity:
    # Proceed with booking
```

This ensures the application adheres to the mathematical constraint of $b \leq c$, preventing overbooking.

**Implicit Mathematical Formulation**

While the example code does not explicitly demonstrate complex mathematical formulations or formal verification techniques, the underlying logic of control structures—conditional statements (if-else) and the conceptual representation of functions (e.g., **check_credentials**)—serves to enforce the application's specifications based on basic mathematical and logical principles.