Lesson Plan: Python List Comprehensions

Grade Level: 11th & 12th Grade (Computer Science)

Time Allotment: 45 Minutes

Michigan State Standards:

- CSTA 3A-DA: Data Analysis and Representation
- CSTA 3B-DA: Creating Procedural Abstractions

Key Terms and Concepts:

- List Comprehension: A concise way to create a new list based on an existing list or iterable
- Loop: Repeated execution of a block of code
- Iterable: An object that can be iterated over (e.g., list, string, range)
- Conditional Statement: An expression that controls the flow of code based on a boolean condition

Lesson Outline

Mini Lesson (8-10 minutes)

- **Warm-up (2 minutes):** Briefly review the concept of loops (for loops) and how they can be used to create new lists based on existing ones.
- **Activity (8 minutes):** Divide the class into pairs. Present a scenario: "Imagine you have a list of exam scores and want to create a new list with only the scores above 80." Students brainstorm different ways to achieve this using a loop (possibly leading to an append operation within the loop).

Discussion Question 1 (2 minutes): What are some advantages and disadvantages of using a loop for this task? (Possible Answers: Advantages: Flexibility, readability for complex tasks. Disadvantages: Can be verbose for simple tasks.)

Guided Practice Activities (20 minutes)

- **Introduction to List Comprehensions (5 minutes):** Introduce list comprehensions as a concise way to create new lists in Python. Explain the basic syntax using an example: new_list = [expression for item in iterable].
- **Building Blocks (10 minutes):** Guide students through building simple list comprehensions step-by-step. Start with a base case like numbers = [1, 2, 3, 4, 5] and explore different expressions:
    - Squaring each number: squared_numbers = [number * number for number in numbers]

    - Filtering even numbers: even_numbers = [number for number in numbers if number % 2 == 0]
- **Challenge (5 minutes):** Present a slightly more complex scenario: "Create a list containing only the names longer than 5 characters from a list of student names." Students attempt the list comprehension on their own, then discuss the solution as a class.

Station Rotations (10 minutes) - Dr. Catlin Tucker's Model

Station 1: Comprehension Combos (3 minutes)

- Students are presented with various existing lists and functions (e.g., abs(), len()) and asked to create list comprehensions that combine them for specific tasks (e.g., absolute values of list elements, filtering names with specific lengths).

Station 2: List Comprehension CSI (3 minutes)

- Provide students with a coded snippet containing a list comprehension with a deliberate error. They need to identify and fix the error (e.g., missing colon, syntax mistake).

Station 3: Comprehension Challenge (4 minutes)

- Present a real-world scenario requiring data manipulation (e.g., extracting stock prices above a certain threshold from a financial dataset). Students brainstorm how they could solve it using list comprehensions.

Activities (5 minutes)

- **Quick Quiz (2 minutes):** A short quiz with multiple-choice questions reinforces key concepts of list comprehension syntax and functionalities.
- **Think-Pair-Share (3 minutes):** Pose a discussion question: "When would you choose a list comprehension over a traditional loop?" Students reflect individually (think), discuss with a partner (pair), then share their thoughts with the class (share).

Independent Practice (10 minutes)

- Students complete a worksheet with various exercises involving creating list comprehensions for different tasks (e.g., manipulating numerical data, filtering strings).

Assessment:

- Review completed worksheets and class participation during discussions and activities.
- Consider an exit ticket: Students write a short reflection on the advantages and disadvantages of using list comprehensions.

Differentiation:

- **Advanced Students:** Challenge them with nested list comprehensions or manipulating complex data structures like dictionaries.
- **Struggling Students:** Provide additional support during guided practice and station rotations. Offer alternative exercises or scaffolding during independent practice.

Discussion Questions:

1. We explored how to create new lists using traditional loops. How are list comprehensions different? (Possible Answer: List comprehensions offer a more concise and readable way to achieve the same outcome in a single line of code.)
2. Why might list comprehensions be considered more "Pythonic" than traditional loops? (Possible Answer: Easier to read and they do essentially the same thing as loops (but with less code).

```python
numbers = [1, 2, 3, 4, 5]
squared_numbers = []
for number in numbers:
  squared_numbers.append(number * number)


numbers = [1, 2, 3, 4, 5]
squared_numbers = [number * number for number in numbers]
```

The list comprehension achieves the same result in a single line, making the code more concise and arguably more Pythonic.

However, it's important to note that there are still situations where traditional loops might be preferable. For example, if you need more complex logic within the loop or need to modify existing elements in the list, a loop might be more suitable.