While Class Theory

Class Theory: Python `while` Loops

1. Introduction to Loops in Python

Python provides two primary loop constructs:

1. `while` loops
2. `for` loops

A `while` **loop** runs as long as a specified condition evaluates to `True`, making it ideal for situations where the exact number of iterations isn't known beforehand.

2. The `while` Statement

2.1. Basic Syntax

```
while condition:
    # code block
```

- As long as `condition` remains `True`, the code block executes repeatedly.
- When `condition` becomes `False`, the loop ends.

2.2. Example

```
i = 1
while i < 6:
    print(i)
    i += 1
```

- This loop prints `i` for values 1 through 5.
- The variable `i` is incremented each time to prevent an infinite loop.

Key Tip

Forgetting to update the variable used in the condition can lead to an **infinite loop** the code will keep running until forcibly stopped.

3. The `break` Statement

- The `break` statement **immediately exits** the loop, regardless of the while condition.

3.1. Example

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

- As soon as `i` reaches 3, the loop stops entirely no further iterations happen.

Use Case: Exiting early once a certain condition is met (e.g., found a target value).

4. The `continue` Statement

- The `continue` statement **skips the remainder of the current iteration** and continues with the next iteration of the loop.

4.1. Example

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

- When `i` is 3, `continue` prevents `print(i)` from executing in that iteration. The loop then moves on to check `i < 6` again.

Use Case: Skipping over certain values or conditions without breaking the entire loop.

5. `else` Clause in a `while` Loop

- Python allows an `else` block to follow a while loop.
- The `else` block runs **only if** the loop concludes **normally** (no `break`).

5.1. Example

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

- The `else` message prints once `i` reaches 6 and the condition `i < 6` becomes `False`.
- If a `break` statement had stopped the loop earlier, the `else` block would **not** execute.

6. Common Pitfalls

1. **Infinite Loops**
   - Occur when the loop's condition never becomes `False`. Always ensure the loop variable is updated or use a `break`.
2. **Misuse of `break`**
   - Exiting a loop prematurely might skip important code. Double-check logic.
3. **Skipping with `continue`**
   - Remember that `continue` only affects the current iteration it does not exit the loop.

7. Summary

- `while` **loops** repeatedly run a block of code while a condition remains `True`.
- `break` lets you exit the loop before the condition becomes `False`.
- `continue` skips to the next iteration without exiting the loop entirely.
- An `else` block on a `while` loop runs if the loop finishes without being interrupted by a `break`.

Understanding these features allows for flexible loop control especially helpful when you're unsure how many times you need to iterate, or when user/input-driven events dictate how long your program should keep looping.