😊 While Loops

A **while loop** in Python is used to execute a block of code repeatedly **as long as a specified condition remains True**. It is useful when the number of iterations is not known beforehand and depends on a condition.

How the While Loop Works

1. The loop **checks the condition** before executing the code.
2. If the condition is **True**, the loop executes the block of code.
3. After each iteration, the condition is **re-evaluated**.
4. The loop continues **until the condition becomes False**.

Use Cases of While Loop

1. **User Input Validation**
   - Example: Continuously asking for input until the user enters valid data.
2. **Waiting for a Condition to Be Met**
   - Example: Running a process until a specific event occurs, like waiting for a server response.
3. **Processing Data Until Completion**
   - Example: Reading a file until the end is reached.
4. **Game Loops**
   - Example: Running a game loop until the player chooses to exit.
5. **Counting and Iterations**
   - Example: Implementing counters, like printing numbers from 1 to 10 dynamically.

Benefits of While Loop

1. **Flexible Iteration**
   - Used when the number of repetitions is unknown in advance.
2. **Efficient for Dynamic Conditions**
   - Can handle real-time conditions like waiting for user input or sensor data.
3. **Avoids Unnecessary Iterations**
   - Stops execution as soon as the condition is False.
4. **Useful for Infinite Loops**

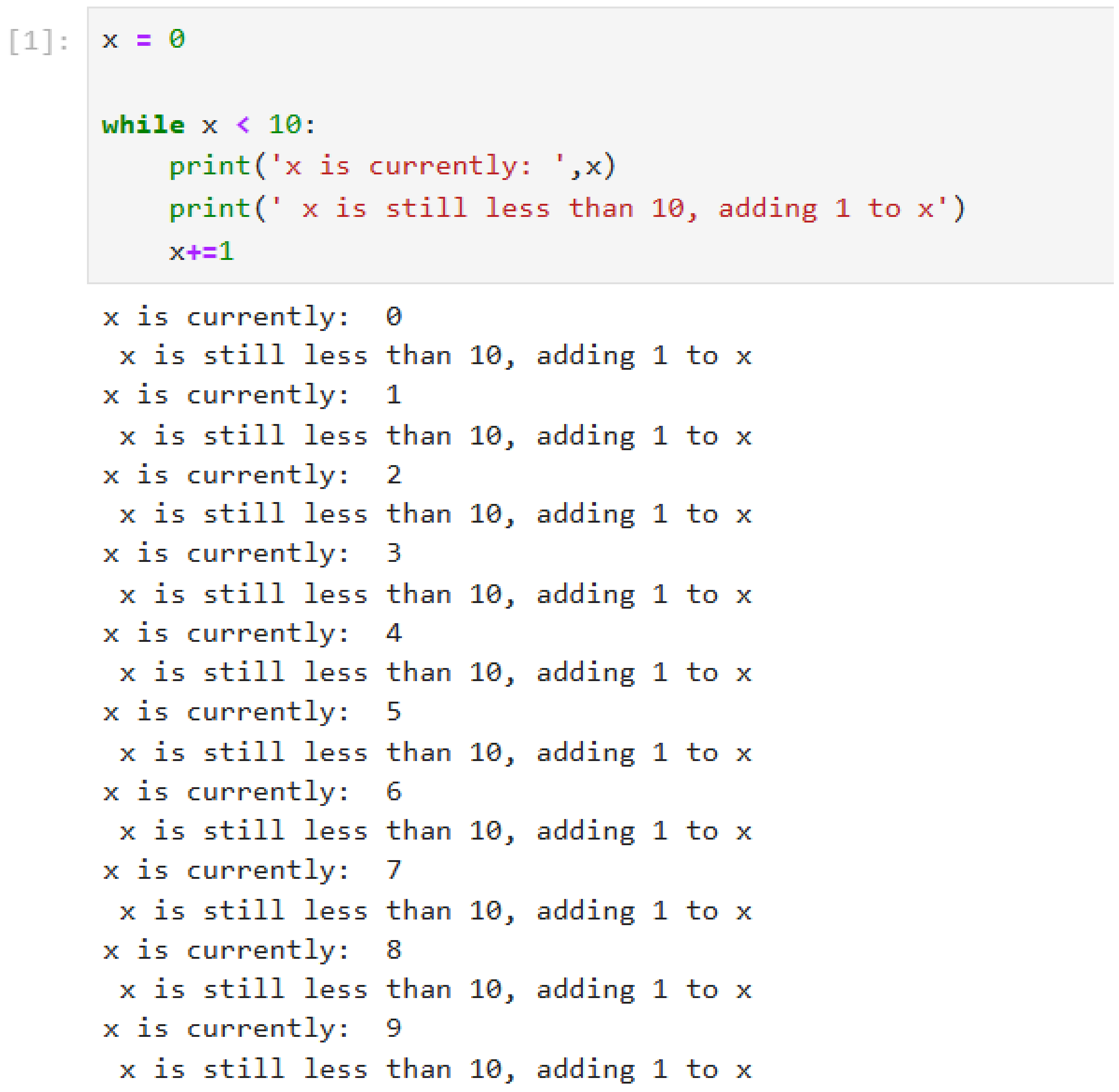
    - Can keep a program running until manually stopped (e.g., real-time monitoring).

😄 To begin with the Lab

1. The while statement in Python is one of the most general ways to perform iteration. A while statement will repeatedly execute a single statement or group of statements as long as the condition is true. The reason it is called a 'loop' is because the code statements are looped through over and over again until the condition is no longer met. The general format of a while loop is:

   ```
   while test:
      code statements
   else:
      final code statements
   ```

2. Did you notice how many times the print statements occurred and how the while loop kept going until the True condition was met, which occurred once x==10. It's important to note that once this occurred, the code stopped.

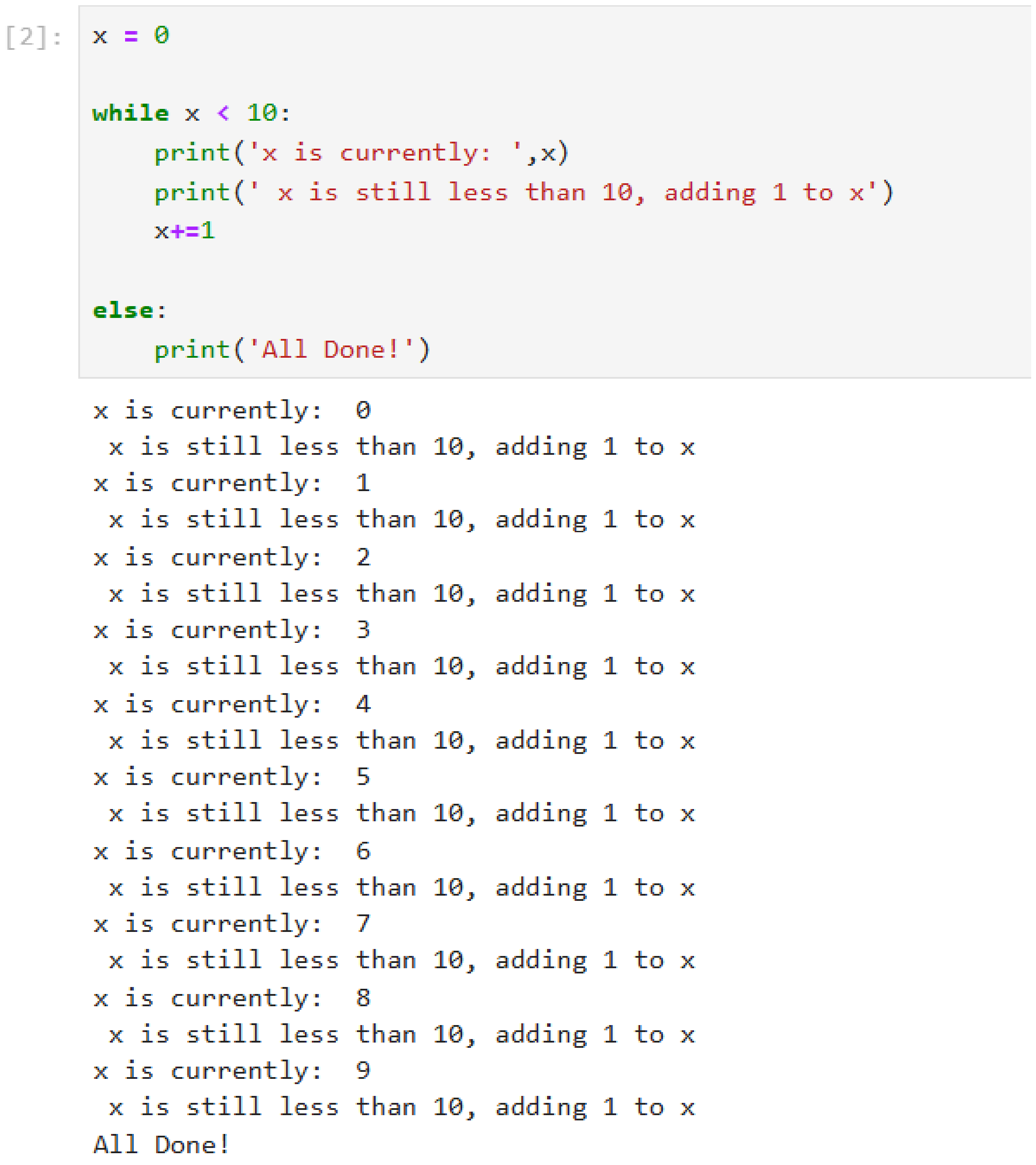


```
[1]: x = 0

     while x < 10:
         print('x is currently: ',x)
         print(' x is still less than 10, adding 1 to x')
         x+=1
```

```
x is currently:  0
 x is still less than 10, adding 1 to x
x is currently:  1
 x is still less than 10, adding 1 to x
x is currently:  2
 x is still less than 10, adding 1 to x
x is currently:  3
 x is still less than 10, adding 1 to x
x is currently:  4
 x is still less than 10, adding 1 to x
x is currently:  5
 x is still less than 10, adding 1 to x
x is currently:  6
 x is still less than 10, adding 1 to x
x is currently:  7
 x is still less than 10, adding 1 to x
x is currently:  8
 x is still less than 10, adding 1 to x
x is currently:  9
 x is still less than 10, adding 1 to x
```

3. In the code below, the **while loop** runs as long as x < 10. Once the loop **condition becomes False**, the else block executes. So, this time we added the else statement with the while loop.



```
[2]: x = 0

     while x < 10:
         print('x is currently: ',x)
         print(' x is still less than 10, adding 1 to x')
         x+=1

     else:
         print('All Done!')
```

```
x is currently:  0
 x is still less than 10, adding 1 to x
x is currently:  1
 x is still less than 10, adding 1 to x
x is currently:  2
 x is still less than 10, adding 1 to x
x is currently:  3
 x is still less than 10, adding 1 to x
x is currently:  4
 x is still less than 10, adding 1 to x
x is currently:  5
 x is still less than 10, adding 1 to x
x is currently:  6
 x is still less than 10, adding 1 to x
x is currently:  7
 x is still less than 10, adding 1 to x
x is currently:  8
 x is still less than 10, adding 1 to x
x is currently:  9
 x is still less than 10, adding 1 to x
All Done!
```

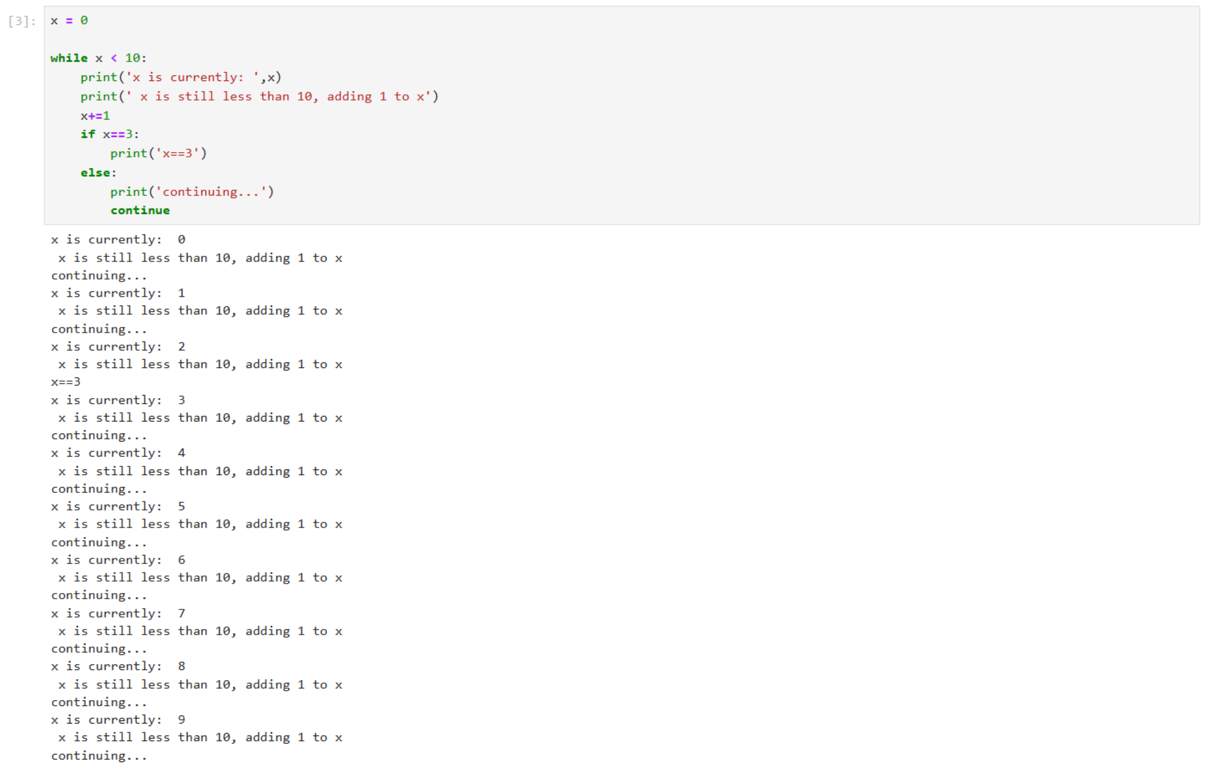
4. We can use break, continue, and pass statements in our loops to add additional functionality for various cases. The three statements are defined by:

   break: Breaks out of the current closest enclosing loop.
   continue: Goes to the top of the closest enclosing loop.
   pass: Does nothing at all.

5. When considering break and continue statements, the while loop's general structure resembles this:

```
while test:
    code statement
    if test:
        break
    if test:
        continue
else:
```

6. This **while loop** runs as long as x < 10, and it includes an if-else condition inside the loop.
7. This loop prints x, increases it by 1, and checks if x == 3. If true, it prints "x==3". Otherwise, it prints "continuing..." and skips to the next loop cycle. The continue here is unnecessary since the loop would naturally continue to the next iteration anyway.



```
[3]: x = 0

     while x < 10:
         print('x is currently: ',x)
         print(' x is still less than 10, adding 1 to x')
         x+=1
         if x==3:
             print('x==3')
         else:
             print('continuing...')
             continue
```

```
x is currently:  0
 x is still less than 10, adding 1 to x
continuing...
x is currently:  1
 x is still less than 10, adding 1 to x
continuing...
x is currently:  2
 x is still less than 10, adding 1 to x
x==3
x is currently:  3
 x is still less than 10, adding 1 to x
continuing...
x is currently:  4
 x is still less than 10, adding 1 to x
continuing...
x is currently:  5
 x is still less than 10, adding 1 to x
continuing...
x is currently:  6
 x is still less than 10, adding 1 to x
continuing...
x is currently:  7
 x is still less than 10, adding 1 to x
continuing...
x is currently:  8
 x is still less than 10, adding 1 to x
continuing...
x is currently:  9
 x is still less than 10, adding 1 to x
continuing...
```

8. The loop starts at x = 0 and keeps running while x < 10. It prints x, adds 1, and checks if x is 3. If x == 3, it prints a message and **stops** (break). Otherwise, it prints **"continuing..."** and moves to the next loop cycle.

[4]:

```python
x = 0

while x < 10:
    print('x is currently: ',x)
    print(' x is still less than 10, adding 1 to x')
    x+=1
    if x==3:
        print('Breaking because x==3')
        break
    else:
        print('continuing...')
        continue
```

```
x is currently:  0
 x is still less than 10, adding 1 to x
continuing...
x is currently:  1
 x is still less than 10, adding 1 to x
continuing...
x is currently:  2
 x is still less than 10, adding 1 to x
Breaking because x==3
```