Hack110 Interest Form!

When? Saturday, April 5th from 10 AM - 12 AM (Midnight)

Where? In Sitterson Lower Lobby

Who can join? Anyone in COMP 110! No prior experience required. Bring a partner or come as yourself (we'll have team-building activities if you want a partner)

Come for a fun day of coding, workshops and events (also **food will be provided**):

- Choose between web development or game development track
- Go to various **workshops & events** such as: Navigating the CS Major, Resume workshop, ice cream station, and kahoot trivia and MORE!
- Link: Interest Form Here! Or via the QR code →
- **Interest form will close Friday, February 28th at 11:59 pm**
    - Fill out this form to get **priority notice** of when we release the sign-up form.

Interest Form!

COMP110

CL16 – Practice with while Loops

Announcements

Quiz 01

- Great job! Median was 85%
- Please submit any regrade requests by **Friday (Feb 28) at 11:59pm**
- Question about something you missed? Please come see us in Office Hours/Tutoring!

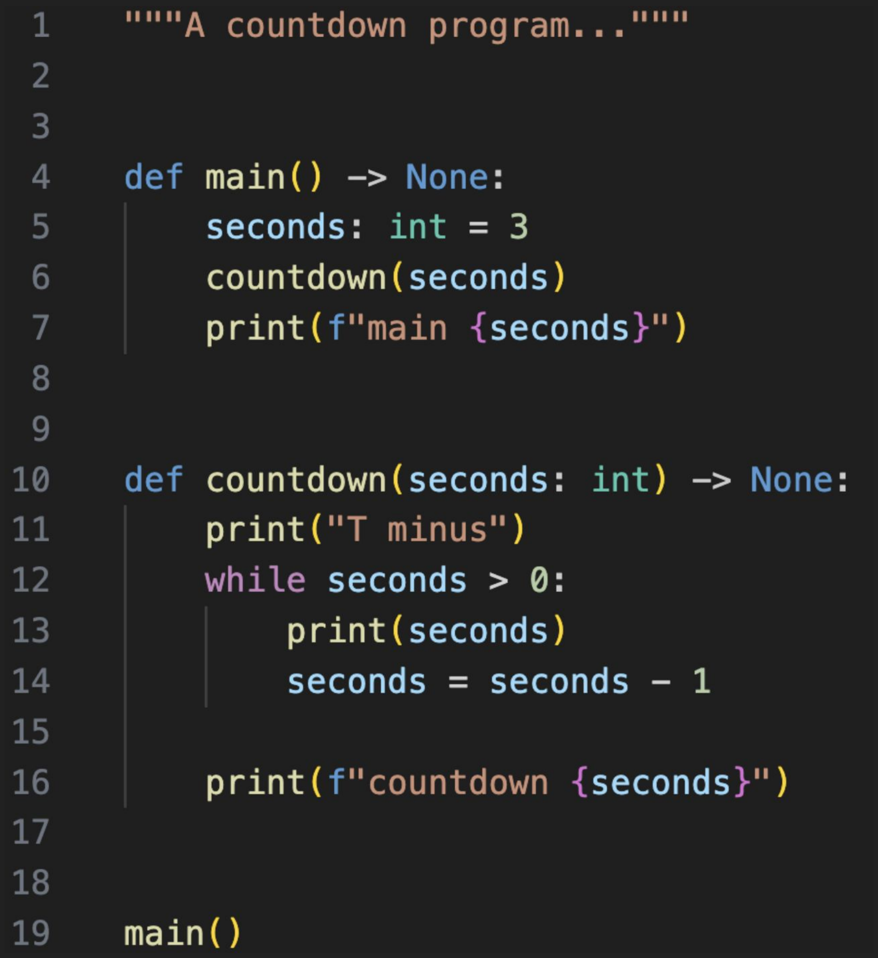
EX02 (Wordle) due Sunday, March 2 at 11:59pm

- You'll be writing 4 functions to make Wordle!

Quiz 02 next Friday (March 7)

- Question about what we've covered thus far? Please visit Office Hours/Tutoring!
- Practice quiz will be posted today
- Solutions video will be posted this week

Warm-Up: Memory Diagram



```python
"""A countdown program..."""


def main() -> None:
    seconds: int = 3
    countdown(seconds)
    print(f"main {seconds}")


def countdown(seconds: int) -> None:
    print("T minus")
    while seconds > 0:
        print(seconds)
        seconds = seconds - 1

    print(f"countdown {seconds}")


main()
```

```python
"""A countdown program..."""


def main() -> None:
    seconds: int = 3
    countdown(seconds)
    print(f"main {seconds}")


def countdown(seconds: int) -> None:
    print("T minus")
    while seconds > 0:
        print(seconds)
        seconds = seconds - 1

    print(f"countdown {seconds}")


main()
```

Relative Reassignment Operators

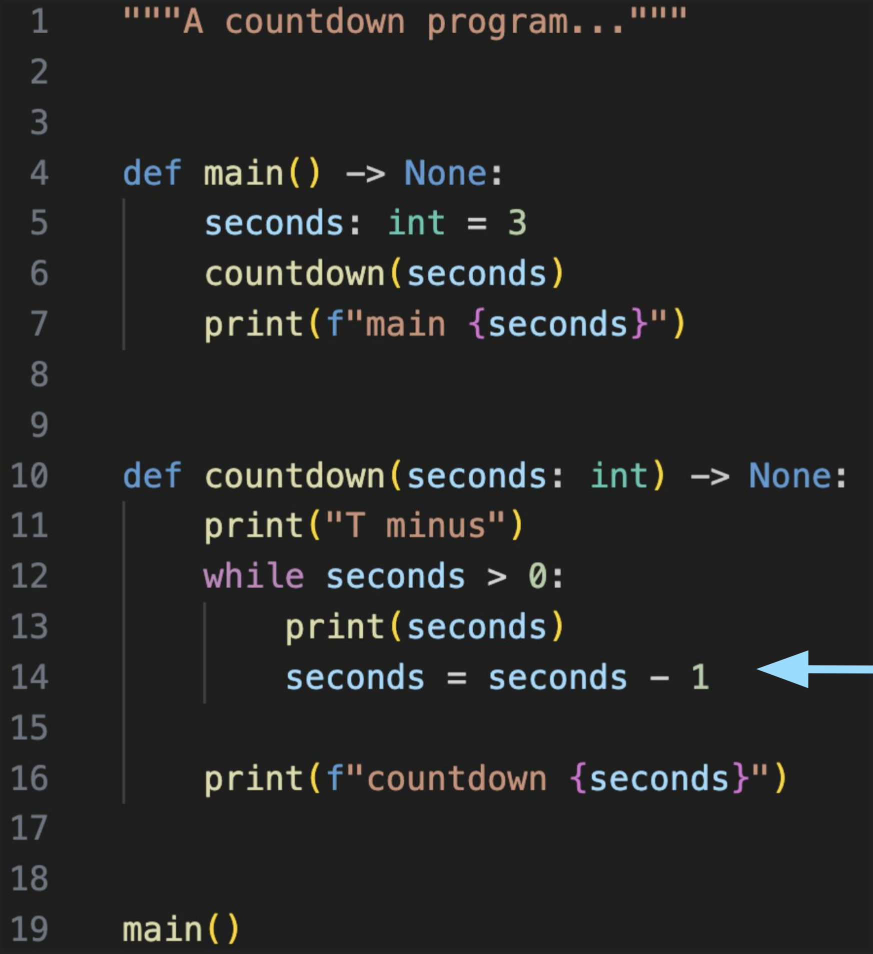
It's *very* common to need to update the value of a variable, relative to its current value, e.g.:

```
count: int = 1
count = count + 1
```

Relative reassignment operators offer a shorthand way of doing this!

```
count += 1
```

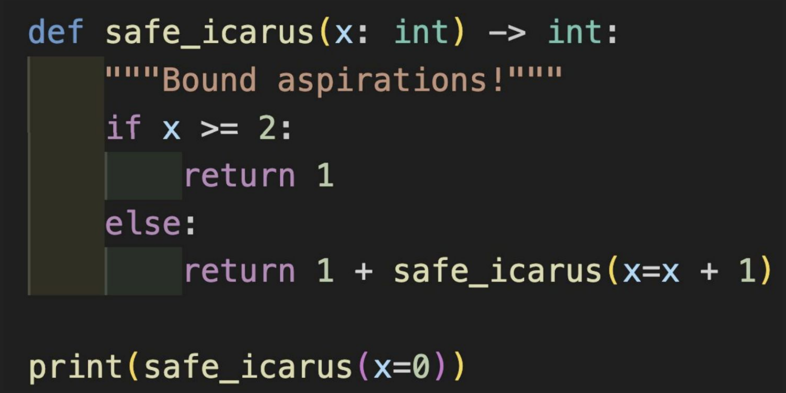
Relative Reassignment Operators



```
 1  """A countdown program..."""
 2
 3
 4  def main() -> None:
 5      seconds: int = 3
 6      countdown(seconds)
 7      print(f"main {seconds}")
 8
 9
10  def countdown(seconds: int) -> None:
11      print("T minus")
12      while seconds > 0:
13          print(seconds)
14          seconds = seconds - 1
15
16      print(f"countdown {seconds}")
17
18
19  main()
```

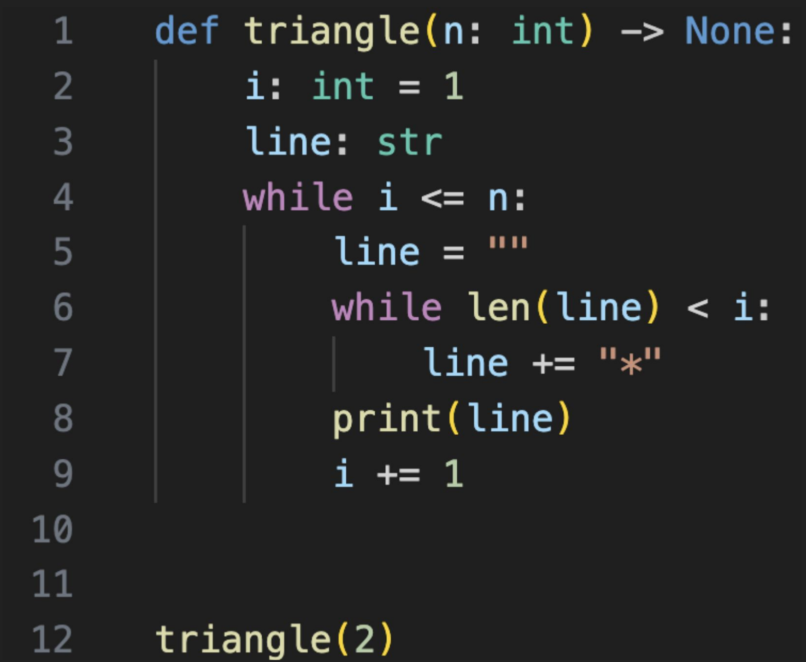
Try writing line 14 using a relative reassignment operator!

Your task: Convert this recursive function to one that uses a while loop!



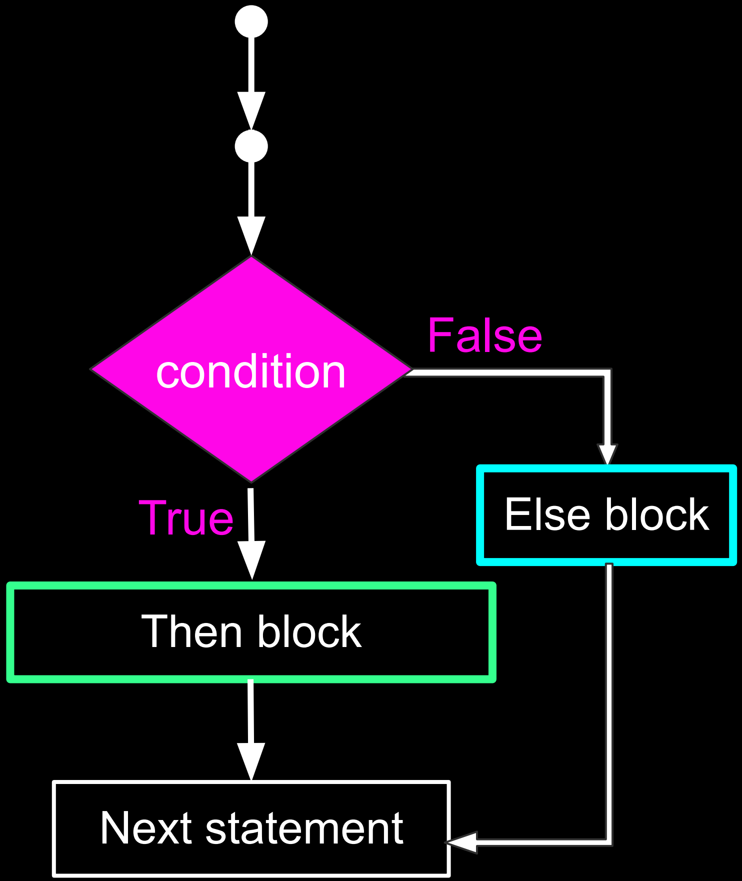
```python
def safe_icarus(x: int) -> int:
    """Bound aspirations!"""
    if x >= 2:
        return 1
    else:
        return 1 + safe_icarus(x=x + 1)

print(safe_icarus(x=0))
```

A nested while loop!



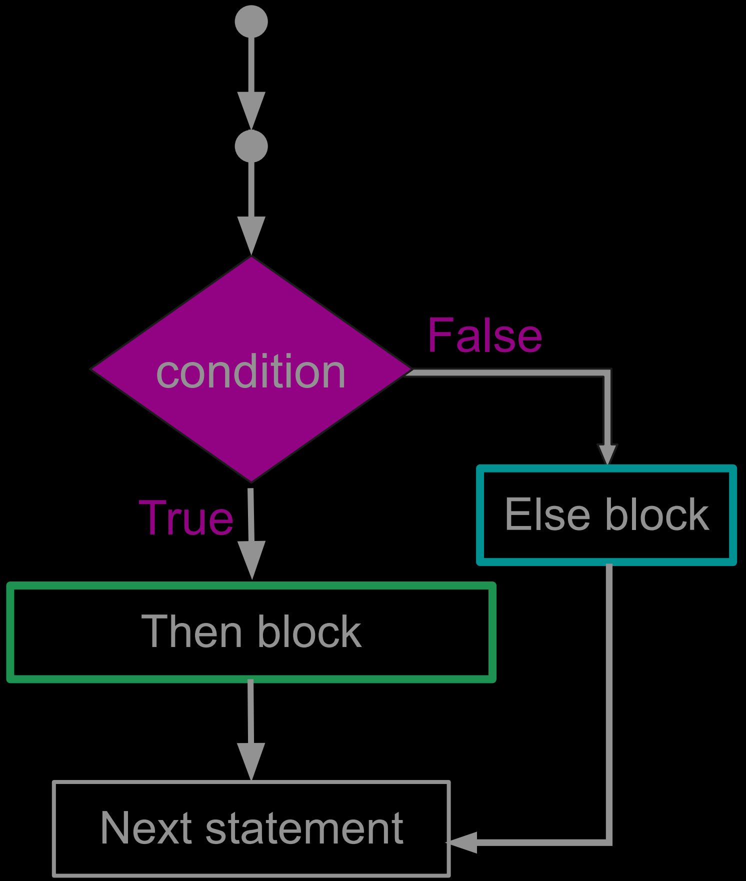
```python
def triangle(n: int) -> None:
    i: int = 1
    line: str
    while i <= n:
        line = ""
        while len(line) < i:
            line += "*"
        print(line)
        i += 1


triangle(2)
```

Recall: if-then-else / *Conditional* Statements

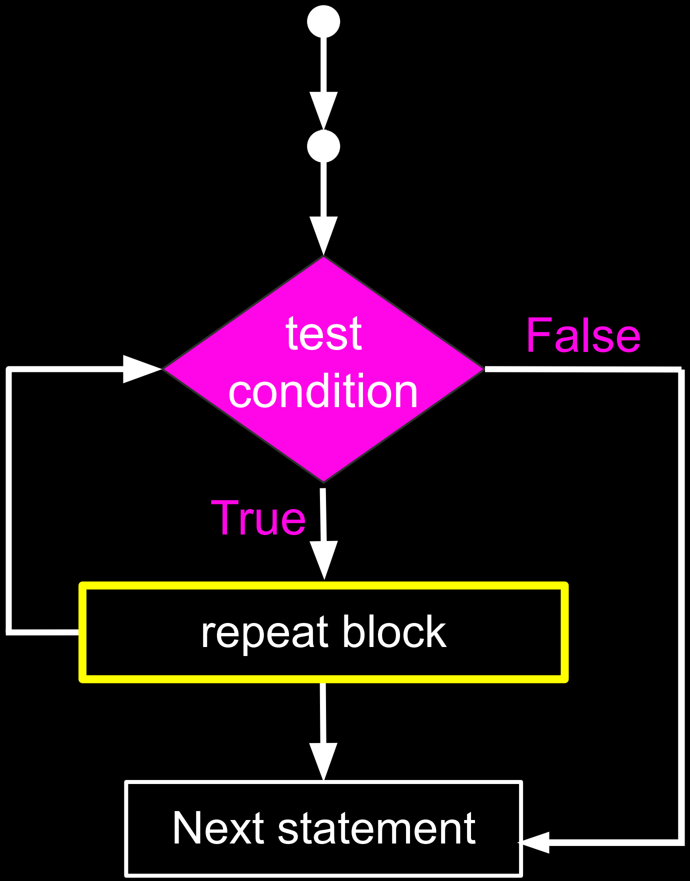


```
if <condition>:
    <then, execute these statements>
else:
    <execute these statements>
<rest of program>
```

`if-then-else` Statements

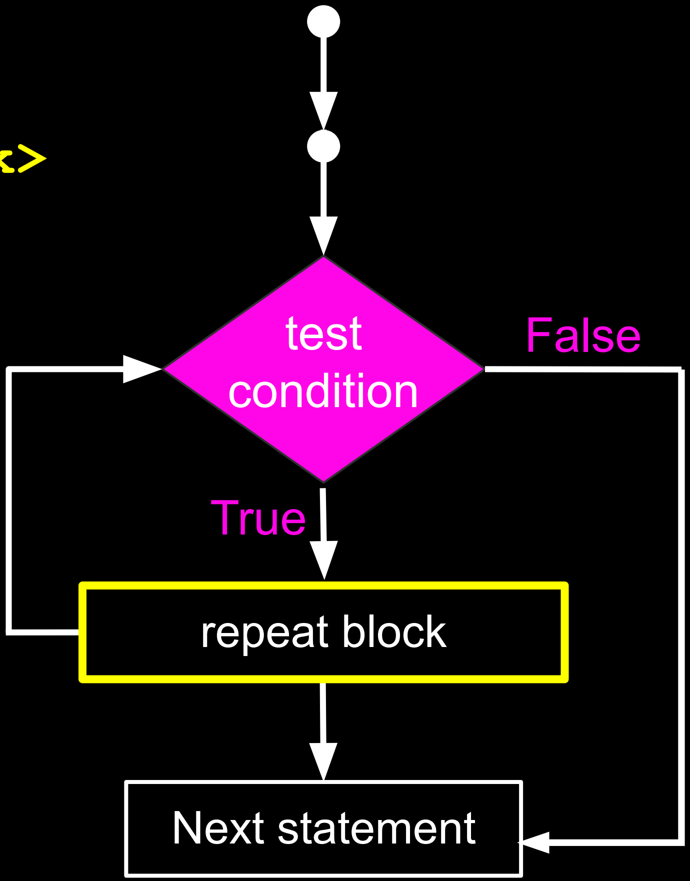


`while` Loop Statements



while Loop Statements

```
while <condition>:
    <execute indented repeat block>
<rest of program>
```

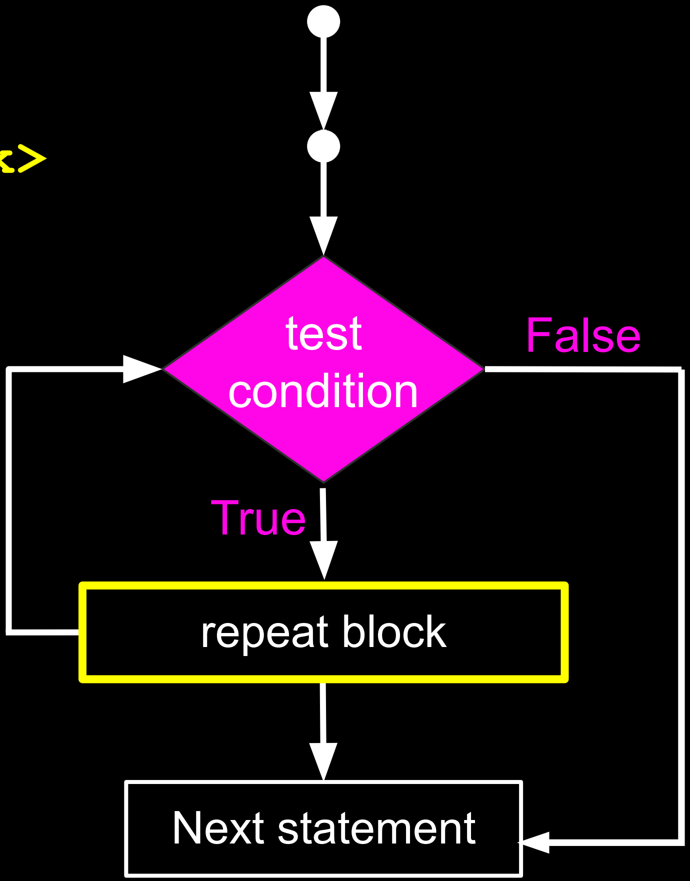


`while` Loop Statements

```
while <condition>:
    <execute indented repeat block>
<rest of program>
```

When we reach a while loop statement in code…

- While the **condition** evaluates to **True**:
  - Execute the **repeat block**
  - Jump back up to the test if the **condition** is still True. This process will repeat (“iterate”) until the condition is False. In which case…
- When the **condition** evaluates to **False**:
  - *Skip past* the **repeat block** and continue on to the next line of code at the same level of indentation as the `while` keyword



Let’s try writing a function, `count_to_n`, that will print values from 0 to n using a `while` loop!

Requirements:

Name: `count_to_n`
Parameter: `n`, an int
Return type: `None`

We’ll need:

- Local variable (to keep track of the count)
- `while` loop (to iterate through each value of count, from 0 to `n`)

Output:

```
Count is:  0
Count is:  1
Count is:  2
Count is:  3
Count is:  4
```

Let's try writing a function, `count_to_n`, that will print values from 0 to n using a `while` loop!

Requirements:

Name: `count_to_n`
Parameter: `n`, an int
Return type: `None`

We'll need:

- Local variable (to keep track of the count)
- `while` loop (to iterate through each value of count, from 0 to `n`)

Output:

```
Count is: 0
Count is: 1
Count is: 2
Count is: 3
Count is: 4
```

Challenge: Pause the video here and try writing this function definition by yourself!

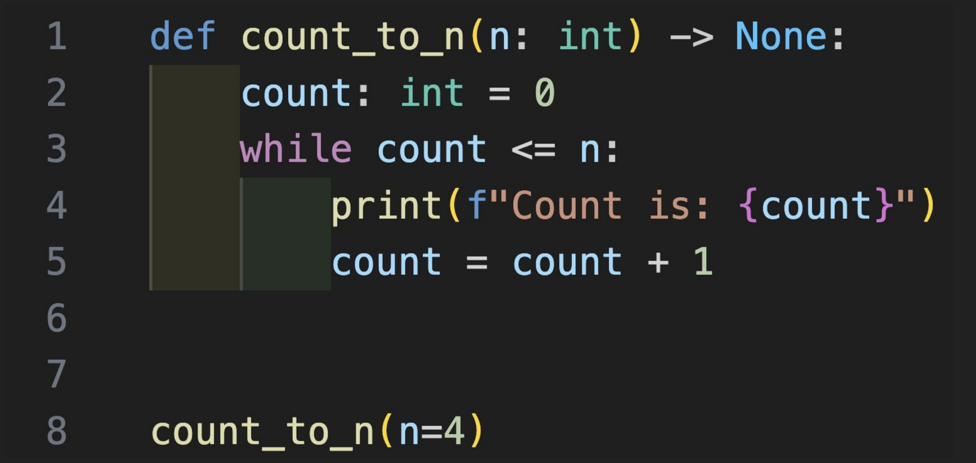
```python
def count_to_n(n: int) -> None:
    count: int = 0
    while count <= n:
        print(f"Count is: {count}")
        count = count + 1


count_to_n(n=4)
```

A common problem: the dreaded *infinite loop*

If a condition in a **`while`** loop never becomes False, the loop will continue indefinitely.

To prevent this:

- Ensure that your loop's condition will eventually be False!



```python
def count_to_n(n: int) -> None:
    count: int = 0
    while count <= n:
        print(f"Count is: {count}")
        count = count + 1


count_to_n(n=4)
```

A common problem: the dreaded *infinite loop*

If a condition in a **`while`** loop never becomes False, the loop will continue indefinitely.

To prevent this:

- Ensure that your loop's condition will eventually be False!

```
1   def count_to_n(n: int) -> None:
2       count: int = 0
3       while count <= n:
4           print(f"Count is: {count}")
5           count = count + 1
6
7
8   count_to_n(n=4)
```

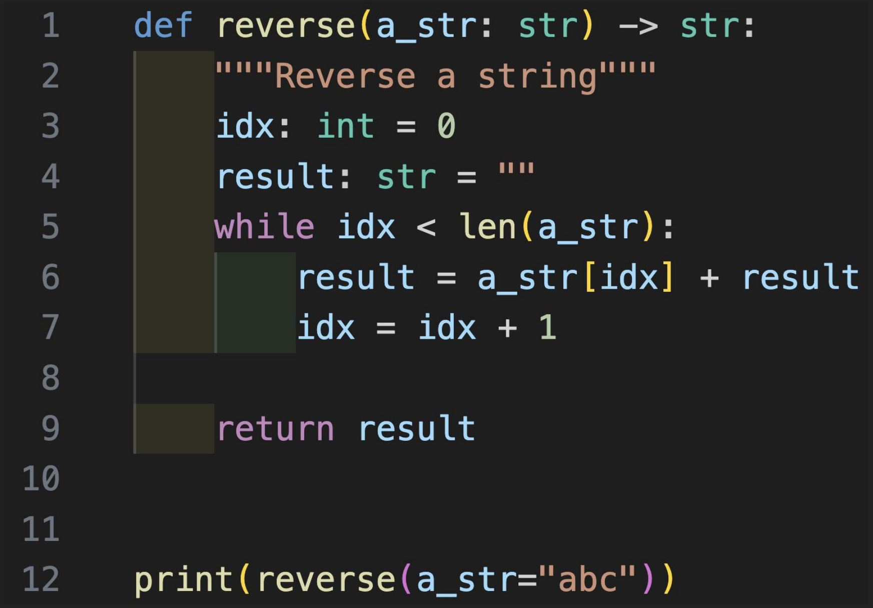
Which line of code in the code listing prevents an *infinite loop* from occurring? What would happen without it?

Common use cases of `while` loops

- **User input validation:** Prompt the user for a valid input until they give one to you!
  - *Think:* our word-guessing game example, or Wordle!
- **Game loops:** Keep a game running until some condition is met
  - Common examples: You run out of lives or attempts
- Iterating through values
  - Examples:
    - Counting from 0 to n ✅
    - Looping through every character in a string (via subscription notation)

Common use cases of `while` loops

- **User input validation:** Prompt the user for a valid input until they give one to you!
  - *Think:* our word-guessing game example, or Wordle!
- **Game loops:** Keep a game running until some condition is met
  - Common examples: You run out of lives or attempts
- Iterating through values
  - Examples:
    - Counting from 0 to n ✅
    - Looping through every character in a string (via subscription notation) ✨



```python
def reverse(a_str: str) -> str:
    """Reverse a string"""
    idx: int = 0
    result: str = ""
    while idx < len(a_str):
        result = a_str[idx] + result
        idx = idx + 1

    return result


print(reverse(a_str="abc"))
```