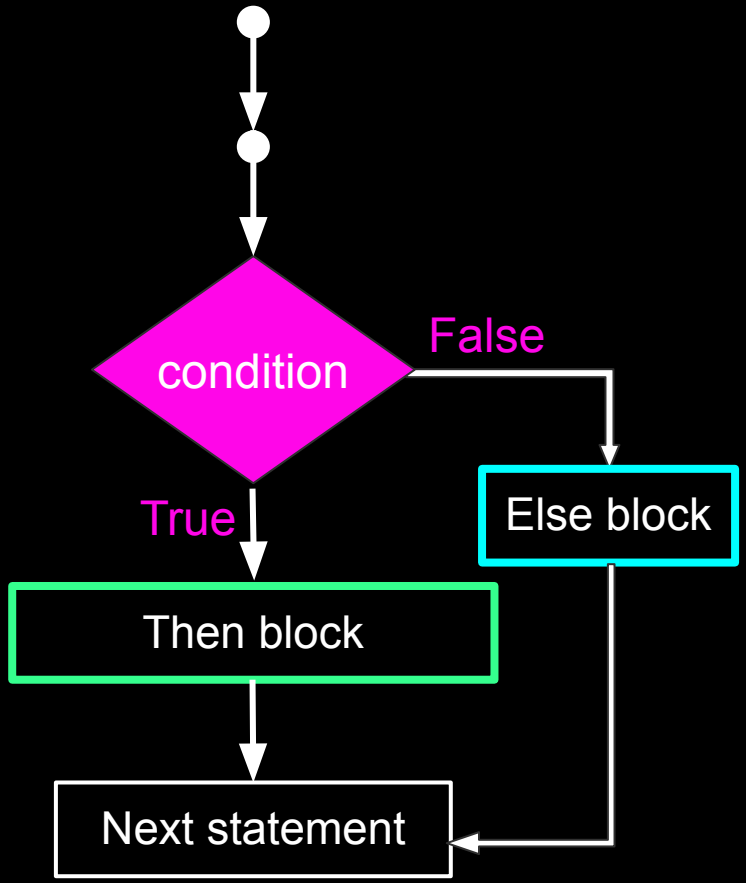
`while` loops + relative reassignment operators

Announcements

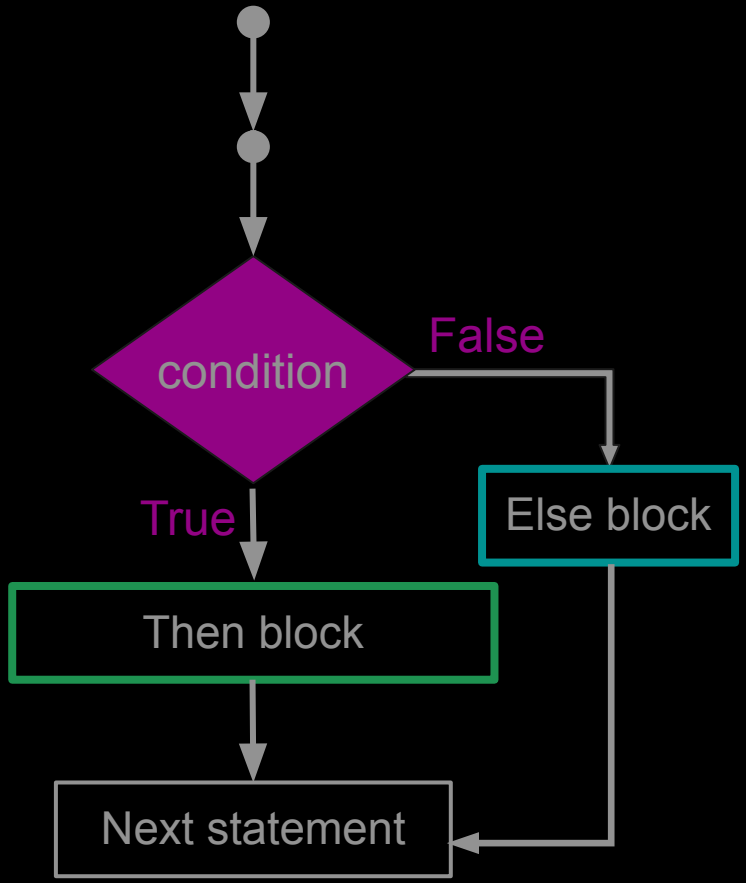
- **LS09: while Loops** due tonight at 11:59pm
  - If you'd like, read this optional reading about while loops!

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

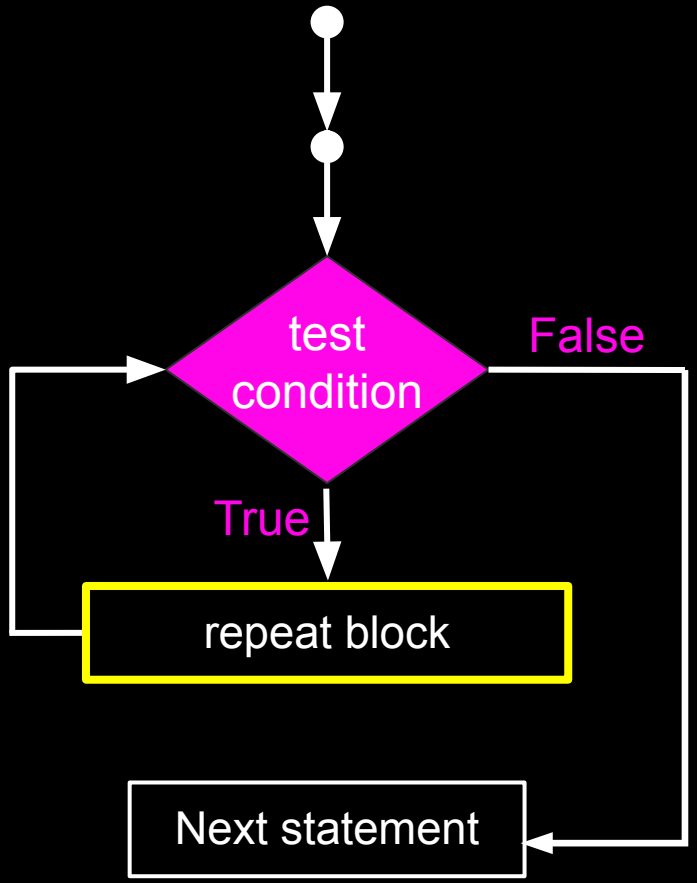



```
if <condition>:
    <then, execute these statements>
else:
    <execute these statements>
<rest of program>
```

`if-then-else` Statements

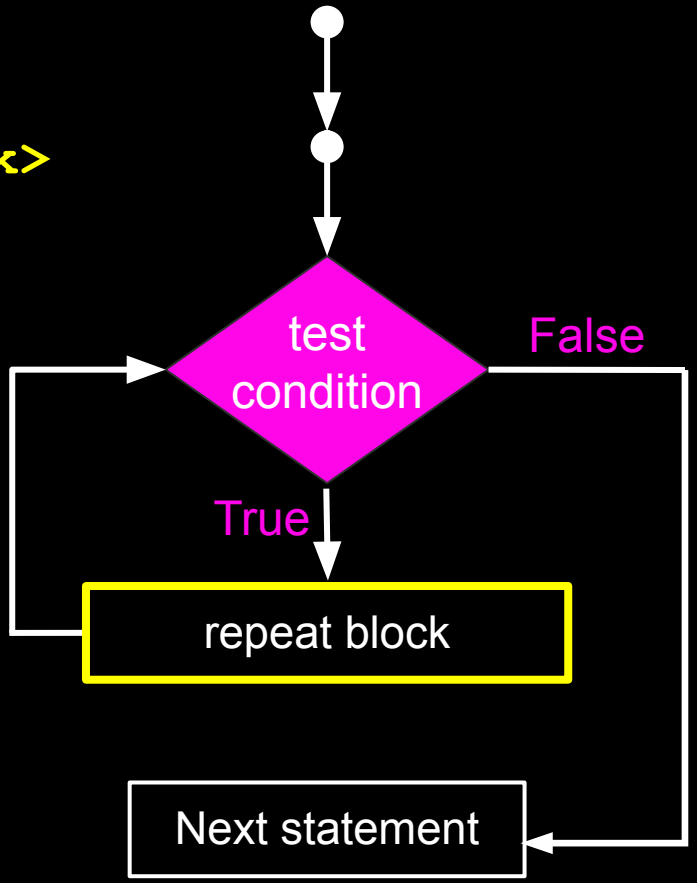


`while` Loop Statements



`while` Loop Statements

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

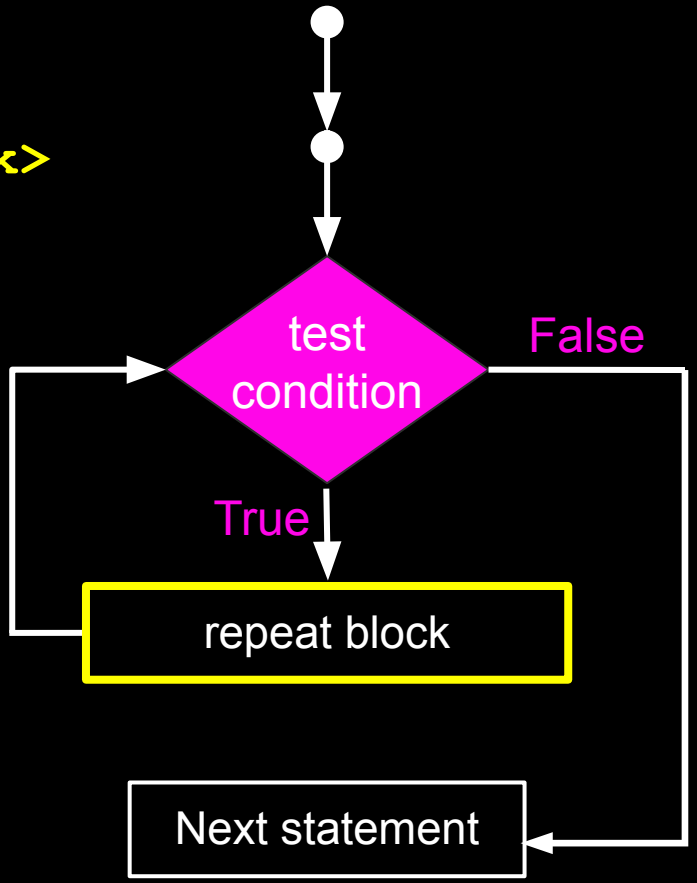


`while` Loop Statements

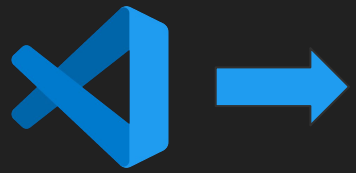
```
while <condition>:
    <execute indented repeat block>
<rest of program>
```

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
```

```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 evaluate to 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)
```

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 evaluate to 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?

Relative Reassignment Operators

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

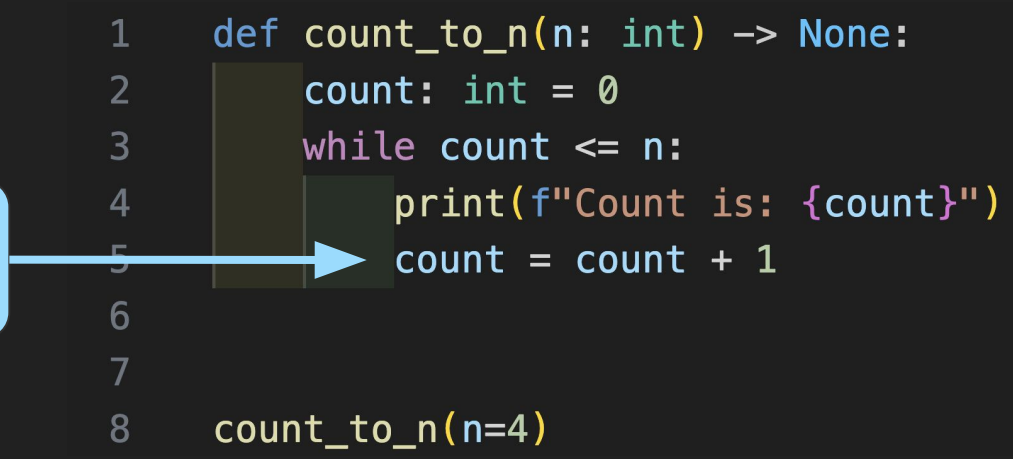
```
count: int = 1
```

```
count = count + 1
```

Relative reassignment operators offer a shorthand way of doing this!

```
count += 1
```

Try re-writing line 5 such that a **relative reassignment operator** will be used!



```
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)
```

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"))
```

Weekly Tutoring + Office Hours

Office Hours (Sitterson Hall (SN) 008):

- Mondays–Fridays: 11am-5pm
- Sundays: 1-5pm

Tutoring (see CSXL site for locations):

- Mondays, Wednesdays, Thursdays: 5-7pm