looping

Summary

LOOPS in PYTHON

while

```
while <boolean expression>:
    Suite
```

for

```
for element in collection:
    Suite
```

Keywords in Loop Structure

`else`, `break`, `continue`, `pass`

Range function

```
range([start], end[, step])
```

```
range(end)
```

```
range(start, end)
```

Nested Loop

A loop inside another loop

slido

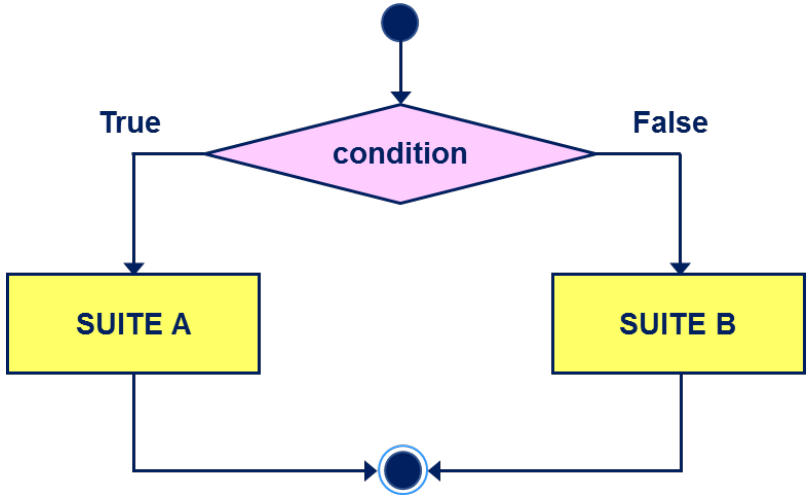
Please download and install the Slido app on all computers you use

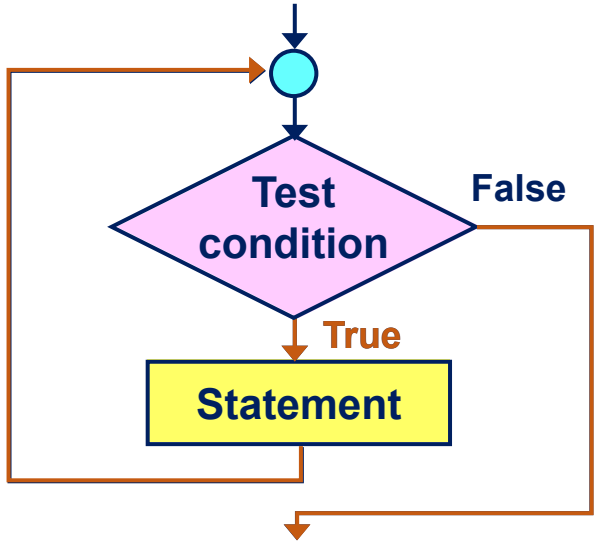
What is the key difference between branching and looping in programming?

ⓘ Start presenting to display the poll results on this slide.

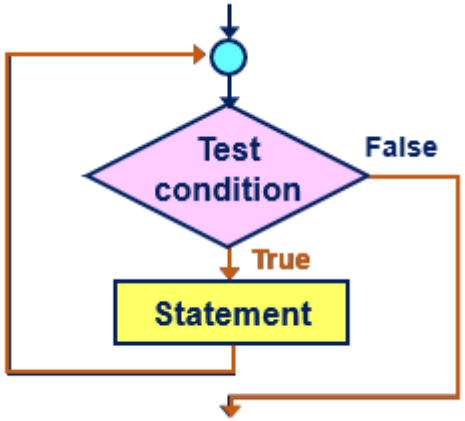
Loops in Python: `while`

- The **`while`** statement allows repetition a suite of Python codes as long as a condition (Boolean expression) is True.
- It is structurally similar to an **`if`** statement but repeats the block until the condition becomes False.





- When the condition becomes False, repetition ends and control moves on to the code following the repetition.



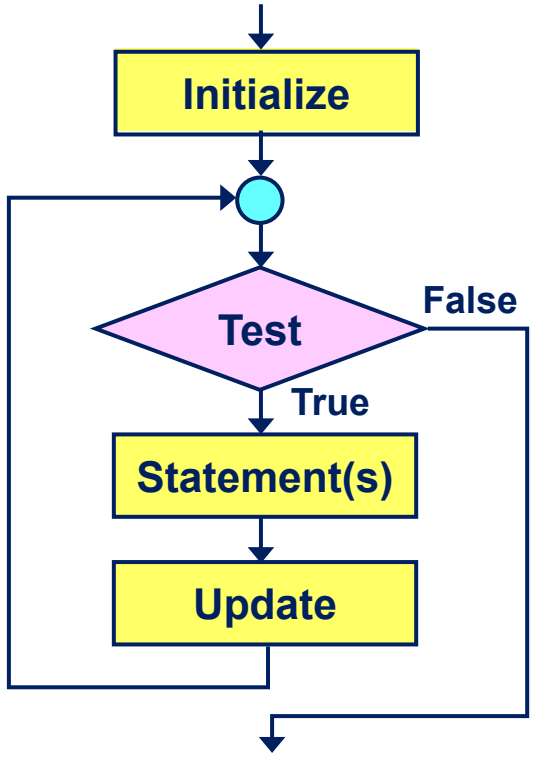
✅ If you see a **path that goes backward** and a **condition being checked repeatedly**, that means **repetition (a loop)** is present.
❌ If every step leads only forward with no rechecking of conditions, then **there is no loop** — it's just **sequential execution**.

Discussion Q2

General Execution of a Loop

Four Steps

1. Initialize	Loop control variable
2. Test	Continue the loop or not?
3. Loop body	Main computation being repeated
4. Update	Modify the value of the loop control variable so that next time when we test, we may exit the loop



- Sometimes a loop may not have all of them. E.g., *Infinite loop* (Test condition is always true).
- A one-time execution of a loop body is referred to as an **iteration** of the loop.

Loops in Python: `while` - Syntax

The whole **while** structure

```
while <boolean expression>:
    Suite (one or more indented statements)
```

indentation

It must use a **colon** followed by a proper **indentation**

the conditional repetition of a while loop and the dynamic stopping point is based on **when the condition is satisfied**.

Real-life Analogy

You ask students to stand and answer one by one.
If a student answers wrong, you call the next one.
The moment one of them answers correctly, you say,“Well done! Everyone can sit down now.”
The process stops as soon as the condition (no correct answer) becomes false.

```
student_index = 0
while not_correct:
    student = class_list[student_index]
    student.try_answer()
    if student.is_correct():
        not_correct = False
    else:
        student_index += 1
```

- `while not_correct:` → “As long as no one has answered correctly”
- `class_list[student_index]` → The next student in the class list
- `try_answer()` → The student gives their answer
- `if is_correct():` → You check if the answer is correct
- `not_correct = False` → If yes, you stop the loop — everyone can sit
- `else student_index += 1` → If not, move to the next student

the conditional repetition of a while loop and the dynamic stopping point — not based on how many students, but when the condition is satisfied.

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
amount = 10
count  = 0
while amount > 2:
    count += 1
    if count % 2 == 0:
        amount -= 2
    elif count % 3 == 0:
        amount -= 3
    else:
        amount -=4

print("count={}, amount={}.".format(count, amount))
```

count=3, amount=1.

Discussion Q3

Summary

LOOPS in PYTHON

while

```
while <boolean expression>:
    Suite
```

for

```
for element in collection:
    Suite
```

Keywords in Loop Structure

`else`, `break`, `continue`, `pass`

Range function

```
range([start], end[, step])
```

```
range(end)
```

```
range(start, end)
```

Nested Loop

A loop inside another loop

The `for` Loop: a Count-Controlled Loop

- Count-Controlled loop: iterates a specific number of times
  - Use a `for` statement to write count-controlled loop
    - Designed to work with sequence of data items
      - Iterates once for each item in the sequence
    - General format:

```
for variable in [val1, val2, etc]:
    statements
```

Here <variable> is a variable that is used for iterating over a <sequence>. On every iteration it takes the next value from <sequence> until the end of sequence is reached.

Loops in Python: `for` - Syntax

```
for iterating_var in <sequence>:
    Suite (one or more indented statements)
```

indentation

It must use a **colon** followed by proper **indentation**.

- Has a header and an associated suite.
- Keywords: **`for`** and **`in`**.
- The keyword **`in`** precedes the sequence.
- The variable iterating_var is a variable associated with the **`for`** loop that is assigned the value of an element in the sequence.
  - The variable iterating_var is assigned a different element during each pass of the **`for`** loop.
  - Eventually, iterating_var will be assigned to each element in the sequence.

Analogy: Taking Attendance in Class

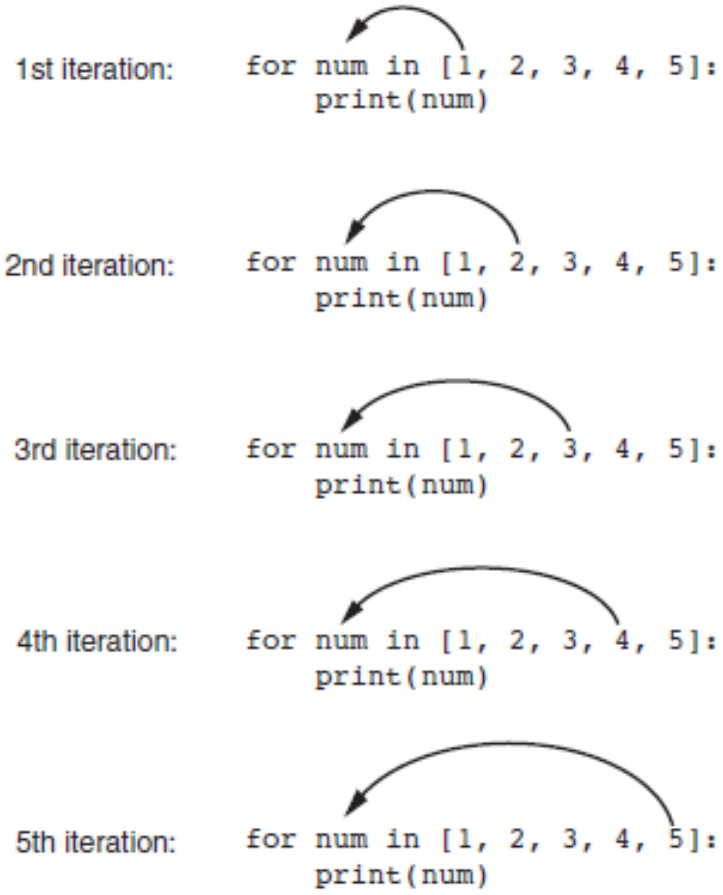
You (the teacher) go through each name in the class list (the sequence).
For every student name you call out (iterating variable),
you perform one or more tasks — like saying “Present” or checking homework (the indented suite).
You repeat this until you've gone through the entire list.

```
for student in class_list:
    print("Present")
```

- **for** → Like saying "For each student..."
- **iterating_var** → This is a temporary name you use one at a time (e.g., student)
- **in <sequence>** → The class list — it’s the group you’re going through one by one
- **:** → Tells the computer to now perform the following action(s)
- **suite (one or more indented statements)** → This is what you do **for each student** — for example, mark them "Present", ask them a question, etc.

This analogy emphasizes the fixed repetition of a for loop and the predetermined stopping point — it goes through a known list of students one by one, and stops automatically after the last student has been checked.

Figure 4-4 The `for` loop



The variable iterating_var is assigned a different element during each pass of the for loop. Eventually, iterating_var will be assigned to each element in the sequence.

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```python
for i in ["Hello", "Hi"]:
  print("Python", end ="*")
```

Python*Python*

slido

Please download and install the Slido app on all computers you use

Which of the following statements accurately describes the difference between a while loop and a for loop in Python?

ⓘ Start presenting to display the poll results on this slide.

Discussion Q4

Summary

LOOPS in PYTHON

while

```
while <boolean expression>:
    Suite
```

for

```
for element in collection:
    Suite
```

Keywords in Loop Structure

`else`, `break`, `continue`, `pass`

Range function

```
range([start], end[, step])
range(end)
range(start, end)
```

Nested Loop

A loop inside another loop

Analogy: Handing Out Exam Papers Row by Row

Imagine you're a teacher with **30 students** seated in rows. You need to **hand out exam papers** to students in **seats 1 to 30**.
You don't want to manually call out every seat number. Instead, you use a **systematic pattern**:
"Start at seat 1, go up by 1 each time, and stop after seat 30."

That's exactly how range(1, 31) works in Python:

Python: The Range Function

`range()`

- It is a useful built-in function in Python.
- It generates a list of integers from start up to end (but excluding end) with step-size step.

Difference between each number in the sequence. (default value is 1)

```
range([start], end[, step])
```

Starting number of the sequence. (default value is 0)

Generate numbers up to, but not including, this number.

Note: All parameters must be integers, positive or negative.

```python
for seat in range(1, 31):
    hand_out_paper_to(seat)
```

🧠 Breakdown of `range(start, stop, step)`

Part	Analogy	Python
`start`	First seat number	e.g., 1
`stop`	Stop **before** this number	e.g., 31
`step`	How many seats to skip	Default is 1

- range(1, 6) →
- Seats 1 to 5 → Hand out to first 5 students.
- range(1, 10, 2) →
- Every second seat → 1, 3, 5, 7, 9
- range(10, 0, -1) →
- Countdown → 10, 9, ..., 1

What is the output of the following code?

```
print(sum(range(6)))
```

print(sum(range(6))) → 15

```
range(6)
```

Means: Start from `0`, stop **before** `6`.

👉 So it gives:

```
0, 1, 2, 3, 4, 5
```

🔴 **Notice:** The number **6 is NOT included.**

✅ Summary for Students:

> **In `range(x)`, the number `x` is not included!**
> It's like stopping **just before** the final number.

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
num = 10
for i in range (9, 3, -3):
    print("{}+{}={}".format(num,i,num+i),end=', ')
```

10+9=19, 10+6=16,

Summary

LOOPS in PYTHON

while

```
while <boolean expression>:
    Suite
```

for

```
for element in collection:
    Suite
```

Keywords in Loop Structure

`else`, `break`, `continue`, `pass`

Range function

```
range([start], end[, step])
```

```
range(end)
```

```
range(start, end)
```

Nested Loop

A loop inside another loop

Loops in Python: `while-else` (Syntax)

The whole **`while`** structure

```
while boolean expression:
        handle_true()
else:
        handle_false()
rest of the programme
```

Condition is false now, handle and go on with the rest of the program

- **`while`** loop, can have an associated **`else`** statement
- **`else`** statement is executed when the loop finishes under **normal** conditions
  - the last thing the loop does as it exits

Real-Life Analogy: Finding a Seat in a Movie Theater

Imagine you're entering a movie theater looking for an **empty seat**. You start from the first row and move seat by seat.

- If you **find an empty seat**, you sit down and stop looking.
- If you go through **every seat and find none**, you **leave disappointed**.

```python
row = 0
max_rows = 10

while row < max_rows:
    if seat_is_empty(row):
        print(f"Found an empty seat at row {row}. Sitting down.")
        break
    row += 1
else:
    print("No empty seat found. Leaving the theater.")
```

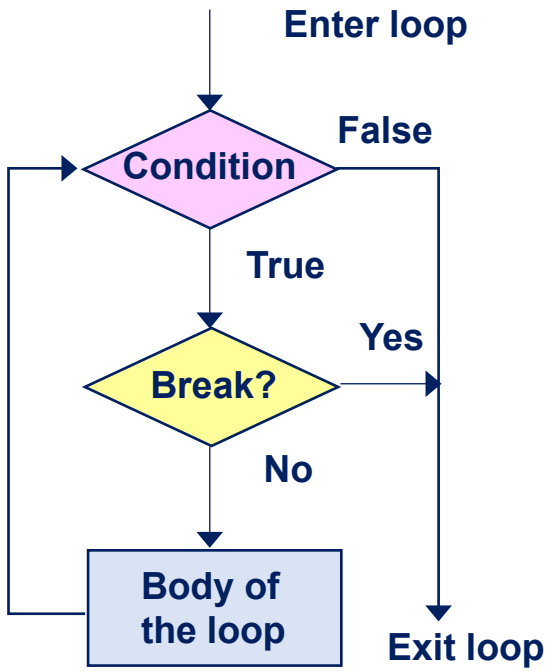
📢 Key Teaching Point:

> The `**else**` block in a `while` loop runs **only if the loop finishes normally** (without a `break`).
> It's like a **backup message**: "If no success inside the loop, do this instead."

Break Statement

- The **`break`** statement can be used to immediately exit the execution of the current loop and skip past all the remaining parts of the loop suite.
- The **`break`** statement is useful for stopping computation when the "answer" has been found or when continuing the computation is otherwise useless.

Flowchart of a break



Working of break in a while loop

```
while test expression:
    body of while
    if condition
        break
    body of while

statements
```

```python
row = 0
max_rows = 10


while row < max_rows:
    if seat_is_empty(row):
        print(f"Found an empty seat at row {row}. Sitting down.")
        break
    row += 1
else:
    print("No empty seat found. Leaving the theater.")
```

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

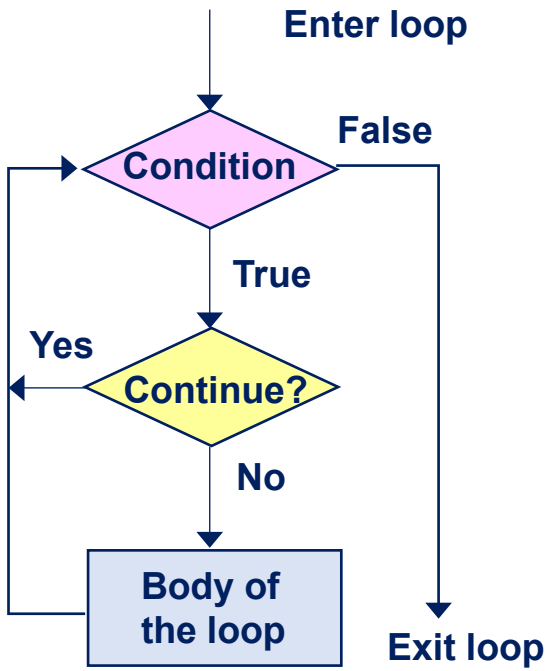
```python
value = 1
print("before ", value, end=", ")

while value <=3:
    value +=1
    if value == 2:
        break
    print("while", value, end =", ")
else:
    print("else", value, end =", ")

print("after ", value, end =", ")
```

before 1, after 2,

Continue Statement

- Skip some portion of the **`while`** suite we are executing and have control flow back to the beginning of the **`while`** loop.
- Exit early from this iteration of the loop (not the loop itself), and keep executing the **`while`** loop.
- The **`continue`** statement continues with the next iteration of the loop.

Flowchart of a continue



Working of continue in a while loop

```
while test expression:
    body of while
    if condition
        continue
    body of while

statements
```

Updated Real-Life Analogy: Finding a Seat in a Movie Theater

You're still looking for an empty seat in a movie theater.

- You go row by row.
- If the seat is **empty and clean**, you sit down → **stop** (`break`)
- If the seat is **empty but dirty**, you **skip it and keep looking** → `continue`
- If **no seat** is available after checking all rows, you **leave** → `else` **block**

```python
row = 0
max_rows = 10

while row < max_rows:
    if seat_is_empty(row):
        if seat_is_dirty(row):
            print(f"Seat at row {row} is dirty. Skipping.")
            row += 1
            continue
        print(f"Found a clean, empty seat at row {row}. Sitting down.")
        break
    row += 1
else:
    print("No suitable seat found. Leaving the theater.")
```

🔁 Control Flow Summary:

- `continue` : "Skip this one and move to the next."
- `break` : "This one is good! Stop the loop."
- `else` : "Checked all seats and none were suitable."

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
value = 1
print("before ", value, end=", ")

while value <=3:
    value +=1
    if value == 2:
        continue
    print("while", value, end =", ")
else:
    print("else", value, end =", ")

print("after ", value, end =", ")
```

before 1, while 3, while 4, else 4, after 4,

for-else-break-continue

```
for target in object:
    # statement suite1
    if boolean expression1:
    break  # Exit loop now; skip else
    if boolean expression2:
    continue  # Go to top of loop now
else:
    # statement suite2
```

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
for num in range(4):
    if num == 4:
        break
    else:
        print(num, end =" ")
else:
    print("else")
```

0 1 2 3 else

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
for num in range(10, -2, -2):
    if num == 8:
        continue
    if num == 2:
        break
    print(num, end =", ")
else:
    print("else")
```

10, 6, 4,

Summary

LOOPS in PYTHON

while

```
while <boolean expression>:
    Suite
```

for

```
for element in collection:
    Suite
```

Keywords in Loop Structure

`else`, `break`, `continue`, `pass`

Range function

```
range([start], end[, step])
range(end)
range(start, end)
```

Nested Loop

A loop inside another loop

Discussion Q3 & Q5

Real-Life Analogy: A Clock (Hours and Minutes)

A **nested loop** is a loop **inside another loop** — the **outer loop** runs first, and for **each time it runs**, the **inner loop runs completely**.

✅ Analogy: How a Clock Works

Outer Loop → **Hour hand (0 to 11 or 1 to 12)**

Inner Loop → **Minute hand (0 to 59)**

> For **each hour**, the **minute hand completes a full circle** from 0 to 59.

```python
for hour in range(1, 13):            # Outer loop (hour)
    for minute in range(60):         # Inner loop (minute)
        print(f"{hour}:{minute:02d}")
```

👩‍🏫 Summary for Students:

A **nested loop** is like a **clock**:

- "For each hour, the minute hand goes through every minute."
- "Only when the minute loop finishes does the hour go up by 1."
- "This helps us repeat smaller tasks (minutes) multiple times inside bigger tasks (hours)."

Nested Loops

Nested Loop a loop inside another loop

- Just as it is possible to have if statements nested within other if statements, a loop may appear inside another loop.
  - An outer loop may enclose an inner loop.

What can be nested?

- Nest as many levels of loops as the system allows.
- Nest different types of loops.

slido

Please download and install the Slido app on all computers you use

Which of the following statements about nested for loops is true in Python?

ⓘ Start presenting to display the poll results on this slide.

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```python
numbers = [10, 20]
items = ["Cat", "Puppy"]

for x in numbers:
  for y in items:
    print(x, y, end =", ")
```

10 Cat, 10 Puppy, 20 Cat, 20 Puppy,

slido

Please download and install the Slido app on all computers you use

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
count = 0
while count < 2:
    count += 1
    for k in range(2):
        if count == k:
            continue
        print(f"count={count}, k={k}", end ="# ")
```

count=1, k=0# count=2, k=0# count=2, k=1#

slido

What is the output of the following code?

ⓘ Start presenting to display the poll results on this slide.

```
for name in "Python":
    count = 1
    if name  == 'y':
     continue
    if name == "o":
     break
    while count < 3:
        if name == 'h':
         break
        print(name, end=' ')
        count = count + 1
```

P P t t

LAB EXERCISE

TA duty: Common agenda for a lab session

- The common agenda for a lab session is as follows (using Week 5 as an example):
- Review the suggested code for Week 4's lab exercise, which is Lab 3.
- Give a briefing on the Week 5's lab exercise.
- While students are working on the Week 5's exercise, provide help and guidance as needed. Patrol the class to identify students who may require assistance, as some may be struggling but hesitant to voice out.

Bad input

- In general, we have assumed that the input we receive is correct (from a file, from the user).
- This is almost never true. There is always the chance that the input could be wrong.
- *"Writing Secure Code," by Howard and LeBlanc*

Input Validation Loops

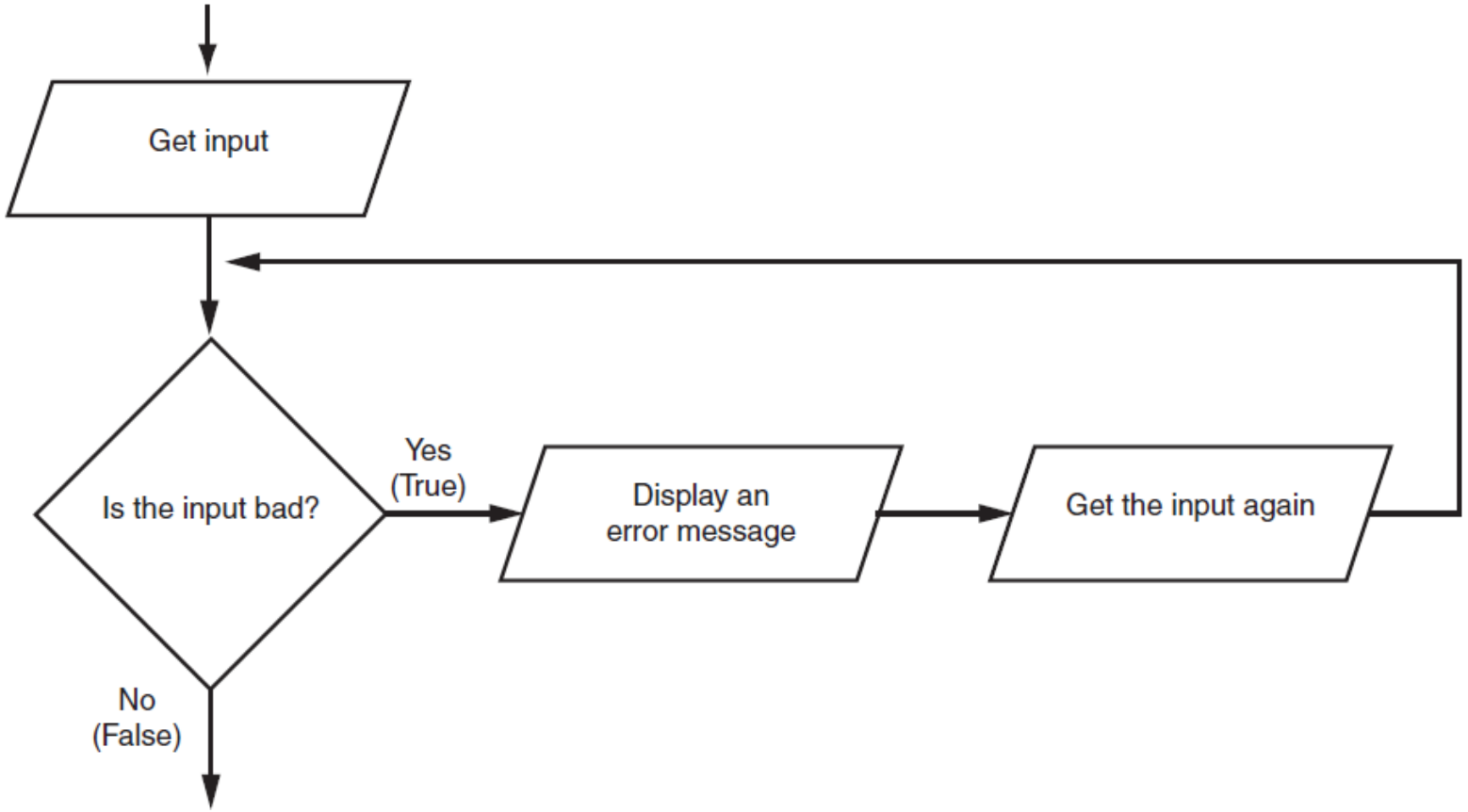
- Input validation: inspecting input before it is processed by the program
  - If input is invalid, prompt user to enter correct data
  - Commonly accomplished using a `while` loop which repeats as long as the input is bad
    - If input is bad, display error message and receive another set of data
    - If input is good, continue to process the input

Lab exercise: Input Validation

- # TODO : 2.While loop to repeatedly ask for valid attack coordinates
- # Add you code of TODO 2 here

Input Validation Loops (cont'd.)

Figure 4-7 Logic containing an input validation loop



Exceptions: Our programs should be able to handle this.

Exception: error that occurs while a program is running

- Usually causes program to abruptly halt

Traceback: error message that gives information regarding line numbers that caused the exception

- Indicates the type of exception and brief description of the error that caused exception to be raised