While Loop

Table of contents

01

While Loop Basics

Overview, Syntax, Flowchart, Example

while Loop

- A **loop** is a code block that executes specific instructions repeatedly.
- Python provides **two** main types of loops for iterative tasks: ***for loop*** and ***while loop.***
- The **while loop** in Python is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The loop continues until the condition is **False**.
- It's especially useful when the number of iterations is **not known** in advance, as opposed to the for loop which is often used when you know how many times the loop should run.

while Loop (Syntax)

Python

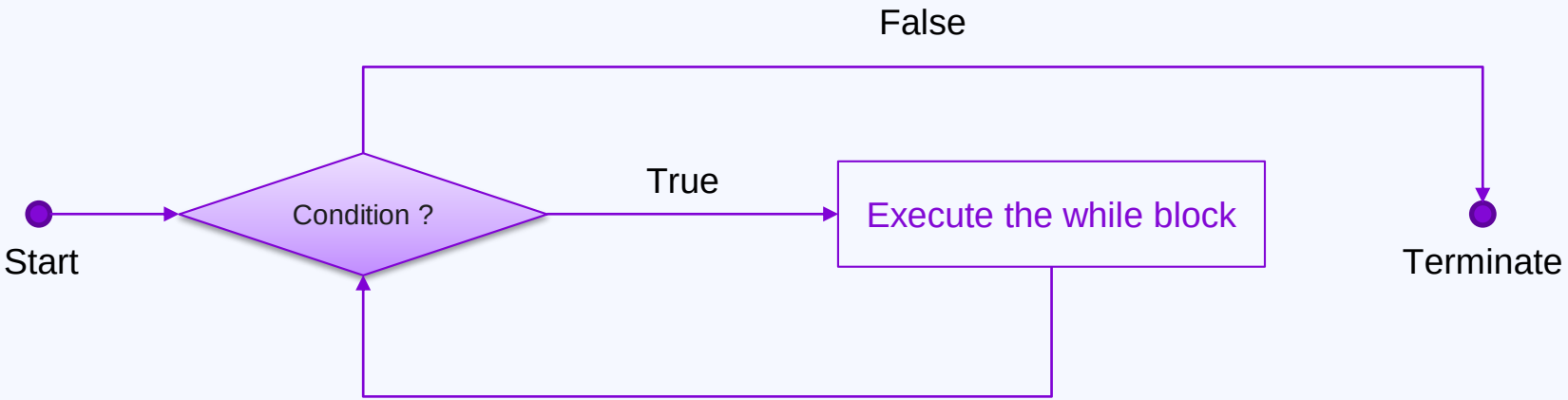
```
while condition:
    # Loop Body
```

Condition: The loop runs as long as this condition is True.

Loop Body: The indented block of code under the while statement that gets executed repeatedly.

Update/Control Statement: To avoid an *infinite loop*, you should modify the variable being checked in the condition, otherwise, the loop *will* run forever.

while Loop (flowchart)



while Loop (Example)

Python	Output

`counter = 1`
 	
`while counter <= 5:`
`    print(counter)`
`    counter += 1`
	1
	2
	3
	4
	5

while Loop (Loop Control)

Loop control is primarily managed through **three** statements:

Break:
Immediately exits the loop.
Usage: Exit the loop early after reaching some value or satisfying some condition.

Continue:
Skips the current iteration and moves directly to the next.
Usage: For ignoring certain condition/ value and move on.

Pass or … :
A placeholder statement that does nothing
Usage: while development to avoid errors temporarily.

while Control Example - break

Python	Output

`counter = 1`
 	
`while counter <= 5:`
`    if counter == 3:`
`        break`
`    print(counter)`
`    counter += 1`
	1
	2

while Control Example - continue

Python	Output

counter = 1
while counter <= 5:
    if counter == 3:
        counter += 1
        continue
    print(counter)
    counter += 1
	1
	2
	4
	5

Infinite Loop

- An **infinite loop** occurs when the condition never becomes False.
- This can happen if the control variable is not updated correctly.
- Press “**Ctrl + C**” to forcefully end infinite loop.

Example:

Python

```
while True:
    print("This will print forever")
```

???

Is this an infinite loop?

Python	Output

`counter = 1`
`while counter <= 5:`
`    if counter == 3:`
`        continue`
`    print(counter)`
`    counter += 1`	

???

Is this an infinite loop? → YES

Python	Output

`counter = 1`
`while counter <= 5:`
`    if counter == 3:`
`        continue`
`    print(counter)`
`    counter += 1`
	1
	2
	
	Infinite Loop

02

while-else loop

Overview, Syntax, Flowchart, Example

while-else loop

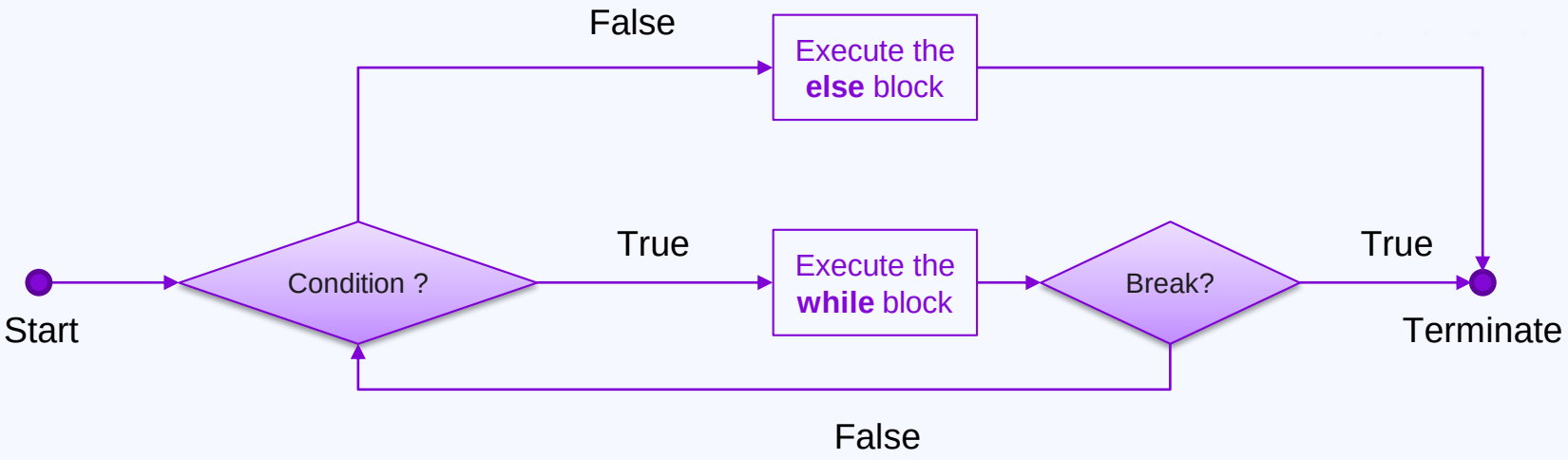
- Python allows an **else** block with a **while** loop.
- **else** block used with a **while** loop will only be executed if the **while** loop terminates normally.
- Normal termination means *completion* of **all iterations** without encountering a **break** statement.
- The **else** doesn't mean "*otherwise*" but rather "*if the loop didn't break*."
- This feature is specific to Python and is **not** commonly found in other programming languages.

while-else loop (Syntax)

Python

```
while condition:
    # loop body
else:
    # Executed only if the loop wasn't broken
```

while-else loop (flowchart)



while-else loop (Example)

Python	Output

counter = 1
while counter <= 3:
    print(counter)
    counter += 1
else:
    print("Loop completed.")
	1
	2
	3
	Loop completed.

03

Nested while loop

Overview, Syntax, Example, loop controls

Nested while loop

- A loop inside another loop is called **nested loop**.
- A **nested while loop** is simply a while loop **inside** another while loop.
- For each iteration of the **outer loop**, the **inner loop** runs entirely.

Nested while loop (Syntax)

Python

```
while outer_condition:
    # Outer loop code
    while inner_condition:
        # Inner loop code
        # Runs for each iteration of the outer loop
```

Nested while loop (Example)

Python	Output

i = 1
while i <= 2:
    j = 1
    while j <= 2:
        print(i, j)
        j += 1
     i += 1
	1 1
	1 2
	2 1
	2 2

Nested while loop (Loop Control)

Both **break** and **continue** statements can be used within nested loops. They will only affect the loop in which they are used.

- **break in an Inner Loop**: Only breaks out of the inner loop, not the outer loop.
- **continue in an Inner Loop**: Skips the rest of the current inner loop iteration but does not affect the outer loop.

Knowledge Reinforcement

1

Question

Which of the following statements about the **while** loop is correct?

Answer

A. The **while** loop always runs a fixed number of times.

B. The **while** loop will execute as long as the specified condition is **True**.

C. The **while** loop can only iterate over lists.

D. The **while** loop does not support **break** and **continue** statements.

1

Question

Which of the following statements about the **while** loop is correct?

Answer

A. The **while** loop always runs a fixed number of times.

B. The while loop will execute as long as the specified condition is True.

C. The **while** loop can only iterate over lists.

D. The **while** loop does not support **break** and **continue** statements.

2

Question

What is the purpose of the **continue** statement in a **while** loop?

Answer

A. To end the loop and exit completely.

B. To skip the current iteration and proceed to the next iteration.

C. To pause the loop temporarily.

D. To repeat the last executed line.

2

Question

What is the purpose of the **continue** statement in a **while** loop?

Answer

A. To end the loop and exit completely.

B. To skip the current iteration and proceed to the next iteration.

C. To pause the loop temporarily.

D. To repeat the last executed line.

3

Question

What does the following code print?

Python

```
x = 1
while x < 5:
    x += 1
print(x)
```

Answer

A. 1

B. 4

C. 5

D. None of the above

3

Question

What does the following code print?

Python

```
x = 1
while x < 5:
    x += 1
print(x)
```

Answer

A. 1

B. 4

C. 5

D. None of the above

4

Question

What type of loop is created if the **loop control variable** is not updated correctly?

Answer

A. Sentinel Loop

B. Finite Loop

C. Infinite Loop

D. Conditional Loop

4

Question

What type of loop is created if the **loop control variable** is not updated correctly?

Answer

A. Sentinel Loop

B. Finite Loop

C. Infinite Loop

D. Conditional Loop

5

Question

The **while** loop always runs at least once, even if the condition is **False** initially.

Answer

A) True

B) False

5

Question

The **while** loop always runs at least once, even if the condition is **False** initially.

Answer

A) True

B) False

6

Question

What does the following code print?

Python

```
count = 1
while count < 4:
    print(count)
    count += 1
```

Answer

A. 1 2

B. 1 2 3

C. 1 2 3 4

D. None of the above

6

Question

What does the following code print?

Python

```
count = 1
while count < 4:
    print(count)
    count += 1
```

Answer

A. 1 2

B. 1 2 3

C. 1 2 3 4

D. None of the above

7

Question

What does the following code print?

Python

```
x = 5
while x > 0:
    x -= 2
    if x == 1:
        break
    print(x)
```

Answer

A. 3 1

B. 3

C. 5 3

D. None

7

Question

What does the following code print?

Python

```
x = 5
while x > 0:
    x -= 2
    if x == 1:
        break
    print(x)
```

Answer

A. 3 1

B. 3

C. 5 3

D. None

8

Question

What does the following code print?

Python

```
x = 10
while x > 7:
    x -= 1
    if x == 8:
        continue
    print(x)
```

Answer

A. 9 8 7

B. 9 7

C. 10 9 8

D. 7 8 9

8

Question

What does the following code print?

Python

```
x = 10
while x > 7:
    x -= 1
    if x == 8:
        continue
    print(x)
```

Answer

A. 9 8 7

B. 9 7

C. 10 9 8

D. 7 8 9

Practice Set

Number Manipulation

Download Link in **Description**

WATCH

Level up your coding with each episode in this focused Python series.

Next Video!

Practice Set – 3 (Solution)