CBS Department

Iterative Control: for loop, while loop

Soma Soleiman Zadeh

Object-Oriented Programming (CBS 215)

Fall 2025 - 2026

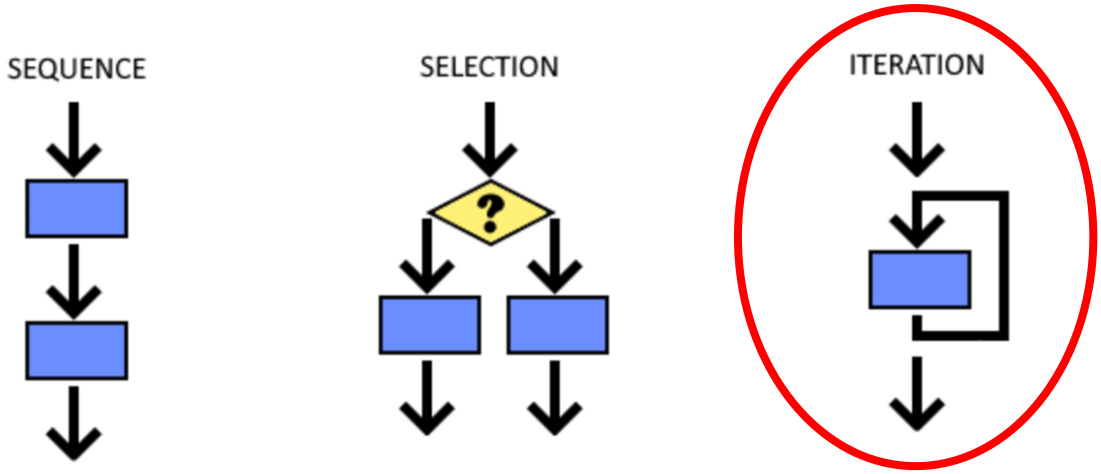
Week 3

October 22-23, 2025

Outline

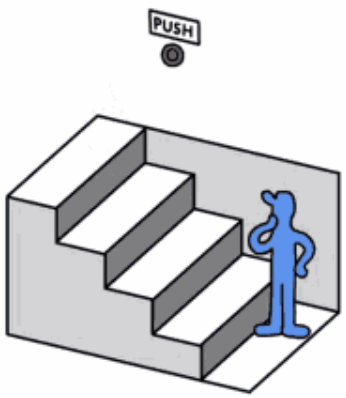
- **Loops**
  - **Definite** Loop (**Counting** Loop)
  - **Indefinite** Loop (**Conditional** Loop)
- **range( )** Function
- **for** Loop and **while** Loop
- **break** and **continue** Statements in Loop
- **Nested** Loop
- **Infinite** Loop
- **while** Loop **Control Variable**

ITERATION Control Structure



What is Loop in Programming?

- **Loops** are a way to **repeat a set of actions** a specific number of times under certain conditions.
- A **loop** takes a few lines of code, and runs them again and again.



Definite Loops vs. Indefinite Loops

- There are two types of loops in Python:
  1. **Definite** (**Counting**) Loops → **for** loop
     - **Exact number of iterations** to do.
     - Iterates through the members of a set (set of numbers, characters, strings).
  2. **Indefinite** (**Conditional**) loops → **while** loop
     - NO definite number of iterations.
     - Iterates while some condition is **True**.

for Loop

- A **for-loop** is a set of instructions that is repeated, or iterated, for every value in a sequence.
  - **Body** of loop → The code that is repeated in a loop
  - **Iteration** of the loop → Each repetition of the loop body
- General syntax of **for** loop:

```
for looping_variable in sequence :
    code block
```

What does happen in a for loop?

1. A for-loop assigns the looping variable to the first element of the sequence. It executes everything in the code block.
2. Then it assigns the looping variable to the next element of the sequence and executes the code block again.
3. It continues until there are no more elements in the sequence to assign.

```
for looping_variable in sequence :
    code block
```

A Simple Example of for Loop

Looping variable (iterator) Sequence

```
for i in [5,4,3,2,1]:
    print(i)
```

Output

```
5
4
3
2
1
```

For Statement Examples

```
for i in [0, 1, 2, 3]:
    print(i)
```

0
1
2
3

```
for item in [2, 4, 5, 10]:
    print(item)
```

2
4
5
10

range( ) Function

- The **range( )** function generates a sequence of numbers, often used in loops for iteration.
- The syntax of **range( )** function:

range (**start** , **stop** , **step**)

- **start** and **step** arguments are optional,
- while **stop** is mandatory.

range( ) Function

range (**start**, **stop**, **step**)

- **start** → The starting number of the sequence.

  The **default value of start is 0** if not specified.

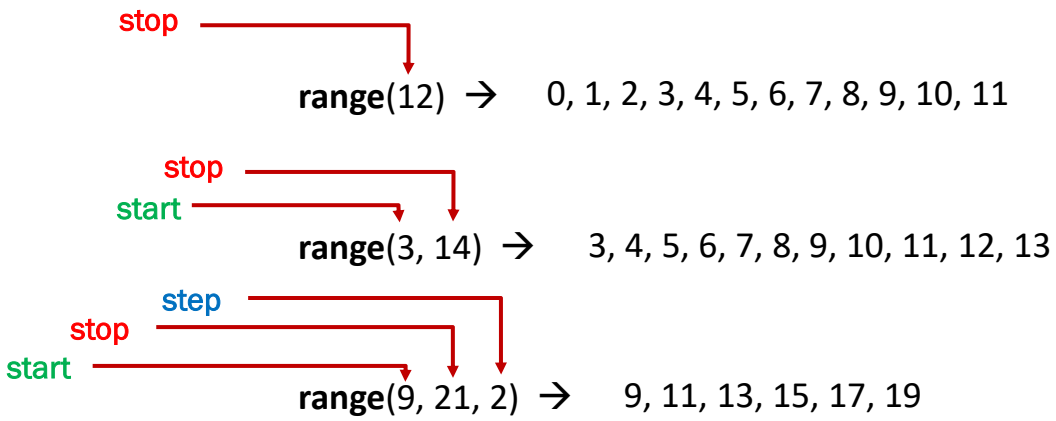
- **stop** → The sequence of numbers is generated up to this number.

  The **stop** number is **not included** in the result sequence.

- **step** → The increment value.

  The **default value of step is 1** if not specified.

range( ) Function Examples



for-range Loop vs for-each Loop

- **for-range Loop:**
  - for loop **with range( ) function**
  - implements repeated action
- **for-each Loop:**
  - for Loop **without range( ) Function**
  - iterates over values that are iterable (strings, lists, …)

for-range Loop Example

```
for i in range(4):
    print("Review of Lecture Notes")

print ("Done!")
```

Output

```
Review of Lecture Notes
Review of Lecture Notes
Review of Lecture Notes
Review of Lecture Notes
Done!
```

What is the code to get this output?

```
for i in range(1,5):
    print("Review", i , "of Lecture Notes")

print ("Done!")
```

Output

Review 1 of Lecture Notes

Review 2 of Lecture Notes

Review 3 of Lecture Notes

Review 4 of Lecture Notes

Done!

What is the output of this code? How many times the loop is executed?

```
for num in range(6):
    print(num*num)
```

Output

0

1

4

9

16

25

What is the Code?

```
for i in range(1,7):
    print("$"*i)

print("Lines of Dollars!")
```

Output

for-each Loop Example

- You can use **for** loop to iterate over every character of a string.
- Print all characters in “CBS Department”, each character in a separate line.

```
for char in 'CBS Department':
    print(char)
```

for-each Loop Example

- We have a list of four names of students. Write a code to get the following output.

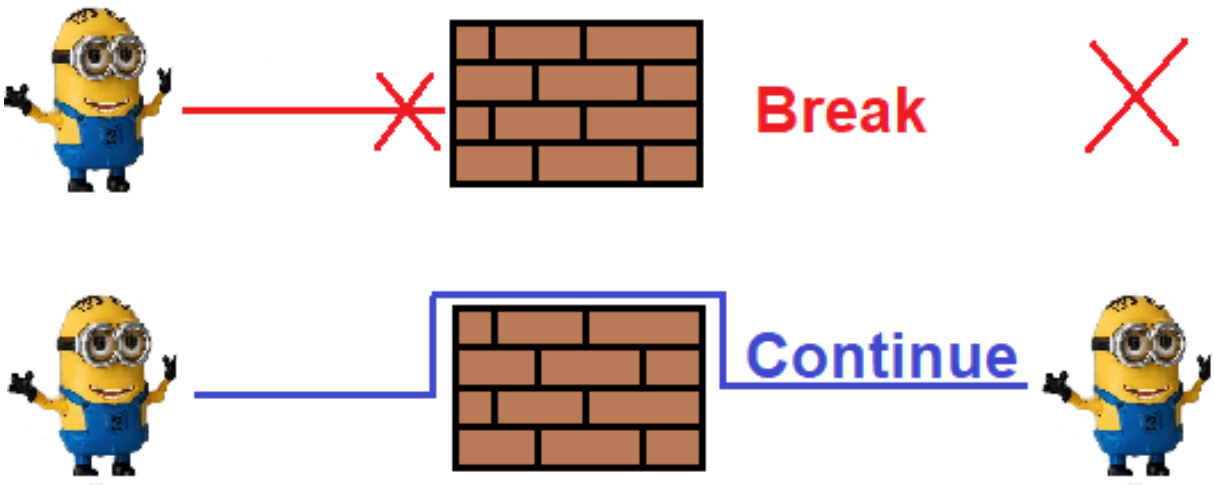
names = [“Ahmed” , “Milad” , “Sara” , “Ako”]

```
Welcome Ahmed
Welcome Milad
Welcome Sara
Welcome Ako
Once again, welcome all.
```

break and continue in Loops

- The **break** and **continue** keywords in a loop can change the flow of the loop.
- The **break** and **continue** statements are used when you want to
  - **terminate the whole loop**
  - or **terminate the current iteration of the loop**,
  - without checking the loop condition or waiting for all repetitions to be finished.

break and continue in Loops



Break Statement

- The **break** keyword in a loop exits the loop immediately.
- Usually, the **break** is put inside an **if** that checks for some condition.

```
numbers = [12 , 1 , 6 , 13 , 6 , 0]
for  num  in numbers :
        if (num == 6) :
            break
        print (num)
print ('All done')
```

Output

```
12
1
All done
```

Continue Statement

- The **continue** keyword skips the current iteration and directs to the start of the next iteration.

```
numbers = [12 , 1 , 6 , 13 , 6 , 0]
for num in numbers :
        if (num == 6) :
                continue
        print (num)
print ('All done')
```

Output

```
12
1
13
0
All done
```

What is the Output?

```
My_text = “I like Programming”
for  char  in  My_text:
        if  (char == “o”) or (char == “a”) :
                continue
        print (char)
print (“Edited Text”)
```

Output

```
I

l
i
k
e

P
r
g
r
m
m
i
n
g
Edited Text
```

Example

Write a Python code for the following list of numbers that:

numbers = [22 , 0 , 44 , 0 , 35 , - 4 , 33 , 48]

- **skips** any number equal to 0.
- **stops** the loop completely when a negative number is found.
- **Prints** the remaining numbers.

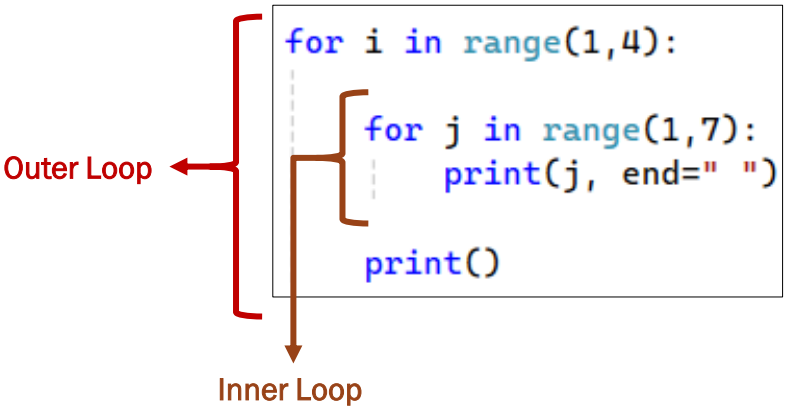
Nested Loop

- A **nested loop** is a loop inside another loop, where the outer loop determines the total number of times the inner loop will execute.

```
Outer_loop expression:
        inner_loop expression:
                Statements inside inner_loop
        statements inside outer_loop
```

Nested Loop – Example

- Consider following nested **for** loop:



Output

```
1 2 3 4 5 6
1 2 3 4 5 6
1 2 3 4 5 6
```

Nested Loop – Example

- What is the output of the following code?

```
for i in range(1,4):
    for j in range(1,7):
        if (i==2) and (j==4):
            break
        print(j, end=" ")
    print()
```

Output

```
1 2 3 4 5 6
1 2 3
1 2 3 4 5 6
```

Nested Loop – Example

- What is the output of the following code?

```
for i in range(1,4):
    for j in range(1,7):
        if (i==2) and (j==4):
            continue
        print(j, end=" ")
    print()
```

Output

```
1 2 3 4 5 6
1 2 3 5 6
1 2 3 4 5 6
```

Nested Loop – Example

- Consider following nested **for** loop. What is the output?

```
students = ["Akam" , "Hassan"]
courses = ["Programming" , "Network" , "Database"]

for i in students:
    for j in courses:
        print(i,"\t",j)
```

Output

```
Akam     Programming
Akam     Network
Akam     Database
Hassan   Programming
Hassan   Network
Hassan   Database
```

Classwork – October 23, 2025

- Write a Python code that **pairs digits between 1 and 9**, that their sum is 10.

```
1+9=10
2+8=10
3+7=10
4+6=10
5+5=10
6+4=10
7+3=10
8+2=10
9+1=10
```

while Loop

Why do We Need while Loop?

- Suppose we want to write a program that can **compute the sum of a series of numbers entered by the user**.
- It should work with any size set of numbers.
- Then we only need to know the **sum** and how many numbers have been added.

Why do We Need while Loop?

- A series of numbers could be handled by some sort of loop. If there are **n** numbers, the loop should execute **n** times.
- We need to calculate and update the **sum** at each iteration.

```
How many numbers do you have? 5
Enter a number: 6
Enter a number: 9
Enter a number: 2
Enter a number: 7
Enter a number: 5

Sum of numbers is:  29
```

Why do We Need while Loop?

- That program got the job done, but **before executing the for loop**, you need to know how many numbers are entered by user.
- **What to do if we don't know how many numbers are entered by user?**
  - and the computer takes care of counting how many numbers there are.

Why do We Need while Loop?

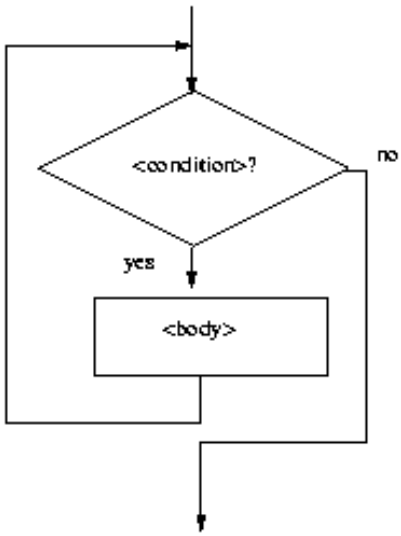
- We can't use a definite loop (**for** loop) unless we know the number of iterations ahead of time.
- We need another tool!
- The **indefinite** or **conditional** loop (**while** loop) keeps iterating until certain conditions are met.

while Loop

- A **while-loop** is used to execute a block of statements **as long as a given condition is True**.
  - **condition** → a Boolean expression
  - **Body** of loop → The code that is repeated in a loop
- General syntax of **while** loop:

```
while (condition) :
        code block
```

While Loop Flow Chart



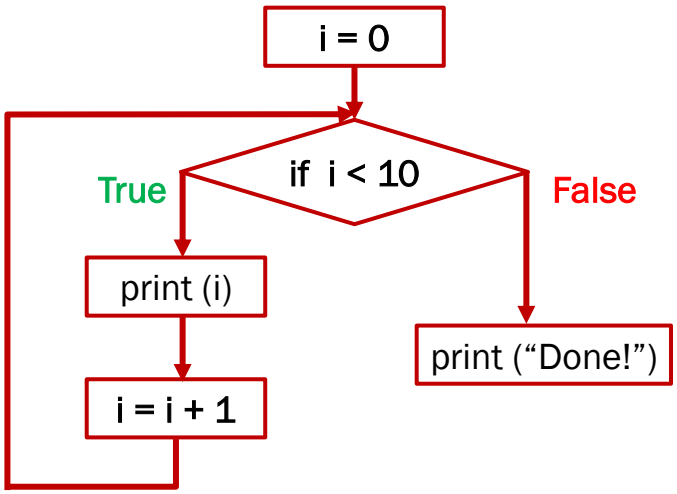
- The **condition** is tested at the top of the loop. As long as the condition is True, the loop body will be executed.
- As the condition becomes False, the loop ends and the code after the loop will be executed.

While Loop Flow Chart – Example

```
i = 0

while i < 10:
    print(i)
    i = i + 1

print("Done!")
```



Infinite while Loop

- An **infinite** loop is a loop that *keeps looping forever*.
- We get stuck in an **infinite loop** if the **continuing condition** in the loop is always **True** and never becomes **False**.

```
i = 0

while i < 10:
    print(i)

print("Done!")
```

Infinite while Loop Example

```
i = 20

while i > 10:
    print("Walk!")
    i = i + 1

print("Done!")
```

While Loop

- Using **for** loop to print numbers from 0 to 10:

```
for i in range(11):
    print(i)
```

- Using **while** loop to print numbers from 0 to 10:

```
i = 0
while (i <= 10):
    print(i)
    i = i + 1
```

Loop Variable in While Loop

- In the **`for`** loop, updating the loop variable is handled automatically.
- The **`while`** loop requires us to manage the loop variable **`i`** by
  - initializing it before the loop and
  - updating it at the bottom of the loop body.

```
i = 0
while (i <= 10):
    print(i)
    i = i + 1
```

1. **Initializing** the loop variable,
2. Decide on a **continuing condition**,
3. **Updating** the loop variable's value

What happens with this code?

```
i = 0
while i >= 10:
    print(i)
```

Output

This code doesn't have any output, because from the first iteration, the condition is False, so the loop body doesn't execute even once.

While Loop without Loop Variable

```
summation = 0

while (True):
    num = int(input("Enter a number: "))

    summation += num

    msg = input("Do you continue entering another number? (yes/no)")
    if msg.lower() == 'no':
        break

print("Sum of entered numbers is: ", summation)
```

This code asks the user to enter numbers and calculates the sum of all numbers entered by the user. We consider a condition to stop the loop when the user doesn't want to enter more numbers anymore.

While Loop with flag Variable

```
summation = 0
flag = True

while (flag):
    num = int(input("Enter a number: "))

    summation += num

    msg = input("Do you continue entering another number? (yes/no)")
    if msg.lower() == 'no':
        flag = False

print("Sum of entered numbers is: ", summation)
```

This code is similar to the code in the previous slide. The only difference is the use of a flag variable to control the loop. As the flag variable becomes False, the loop is terminated.

Change the Previous code to count the numbers entered by the user, too.

```
summation = 0
count = 0

while (True):
    num = int(input("Enter a number: "))
    count += 1
    summation += num

    msg = input("Do you continue entering another number? (yes/no)")
    if msg.lower() == 'no':
        break

print("Total numbers entered by the user: ", count)
print("Sum of entered numbers is: ", summation)
```