# Iteration

Repeatedly executing a code block of statements

Alwin Tareen

## What are Loops(Iteration)?

- Loops are also known as repetition or iteration.
- Loops allow the computer to do the same thing(or similar things) over and over.
- In other words, loops are a way for a program to execute the same code multiple times.
- Loops are an effective design tool, because if you need to change the code that gets repeated, you only need to change it once.

# The `while` Loop

## The indefinite loop

- A `while` loop repeats a section of code, over and over again, as long as some boolean condition is `True`.
- `while` loops are particularly useful when you don't know in advance how many times a loop should run.

## The structure of a `while` loop

- It consists of the keyword `while`, followed by a boolean condition, then a colon.
- The body of the `while` loop is an indented code block of statements.

## The `while` Loop

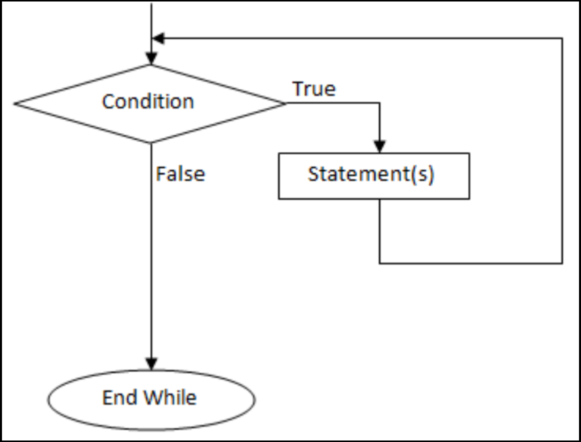
### A `while` loop's code structure

```
while condition:
    code block of statements
```

### The flow of execution of a `while` loop

- A `while` loop first checks the condition it is given, yielding `True` or `False`.
- If the condition evaluates as `True`, then it executes the code block of statements, and repeats execution from the condition check.
- If the condition evaluates as `False`, then the `while` loop is immediately exited.

## Flowchart representation of a `while` loop



## The Counter-controlled Loop

### Looping a given number of times

- A counter-controlled loop is one that repeats a predetermined number of times.
- The condition in this loop is controlled by a counter variable.
- The counter variable keeps track of the number of times that the loop is executed.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

## The Infinite Loop

### Beware the endless loop

- If a `while` loop is given a condition that is always `True`, then the loop will never stop running.
- A common mistake is when a programmer forgets to increment the counter variable within the body of the `while` loop.
- Since the boolean condition will never be `False`, the loop will continue running indefinitely.

```
count = 0
while count < 10:
    print(count)
```

## Summing a Sequence of Integers with `while`

The following `Python` program uses a `while` statement to sum the following sequence of integers:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$$

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

## Incrementing by a Different Amount

- A counter variable can be incremented by a value other than one.
- For example, the following counter is incremented by 10, each time through the loop.

```
count = 0
while count < 100:
    print(count)
    count += 10
```

## The `for` Loop

### The definite loop

- A `for` loop repeats a section of code, for as many times as there are items in a corresponding set of elements.
- In other words, since a `for` loop passes through a known set of items, there is a definite limit as to how many iterations it can run through.
- Usually, we use a data structure known as a **list** to represent the set of items.

```
nums = [19, 384, 485, 714, 55, 61, 856, 329, 28]
```

## The Structure of a `for` Loop

- The first line consists of the keyword `for`, followed by a variable name(usually `item`), then the keyword `in`, then a series of elements(usually a list), followed by a colon.
- The next line is where the body of the `for` loop begins. It consists of an indented code block of statements which we want to be repeated, over and over.

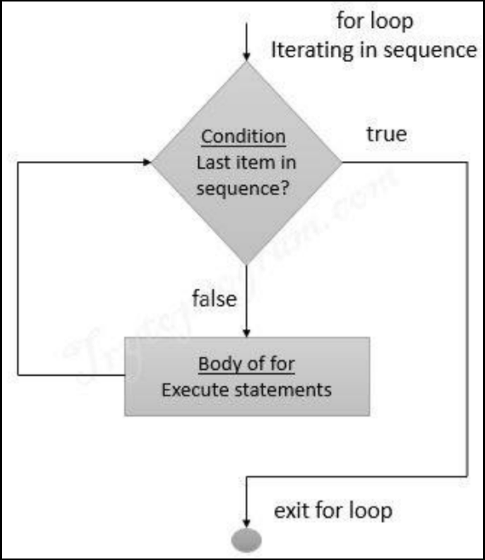
### A `for` loop's code structure

```
for item in elements:
    code block of statements
```

## The Flow of Execution of a `for` Loop

- Initially, the variable name(usually `item`) is set to the first element in the group.
- Then the statements in the code block are run.
- Afterwards, the `for` loop checks to see if there are any more elements in the group. If not, then the `for` loop exits.
- Otherwise, the variable `item` is set to the next element in the group, and the execution repeats.

## Flowchart representation of a `for` loop



## Typical Uses of a for Loop

### Using a for loop as a counter-controlled loop

```
for count in [0, 1, 2, 3, 4]:
    print(count)
```

### Summing a sequence of integers using a for loop

- The following integers are added together:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$$

```
total = 0
for item in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    total += item
print(total)
```

## The break Statement

### Immediately exiting a loop

- Sometimes, you don't know that it's time to end a loop, until you get halfway through the body.
- The `break` statement is like an emergency escape command for a `while` loop or a `for` loop.
- `break` causes an immediate jump to the statements after the end of the loop body.
- For example, suppose you want to exit if 8 appears:

```
import random
while True:
    num = random.randint(1, 10)
    if num == 8:
        break
```

## The continue Statement

- Sometimes, you are in the middle of a code block of statements in a loop, and you want to pass over the rest of the statements, and resume execution from the next iteration.
- In such a case, you can use the `continue` statement to skip to the next iteration, without finishing the rest of the statements in the code block.
- For example, the following program won't display the number 4:

```
for num in [0, 1, 2, 3, 4, 5, 6, 7, 8]:
    if num == 4:
        continue
    print(num)
```

## Iteration: End of Notes