Loops & Iterations

- What are loop
- While loops
- For loops

Loops

- Loops help us whenever we wish to run a block of code multiple times.
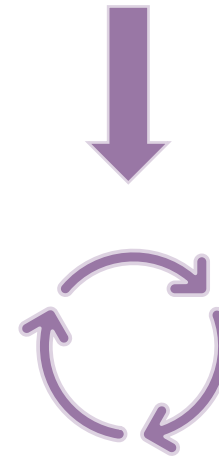
Bart must write down a hundred times "I'm Sorry" → print ("I'm Sorry") → **x100**

While Loop

- A, *while* loop in Python, helps us repeat a group of statements as long as a condition is **true**!

- We can look at the '*while*' expression as – "*As long as*.."

- It <u>requires</u> **conditions** and **statements**.

- A *while* loop receives a *Boolean condition.*

- *As long as a* condition is evaluated as *True*, the code block inside a '*while'* loop executes repeatedly.

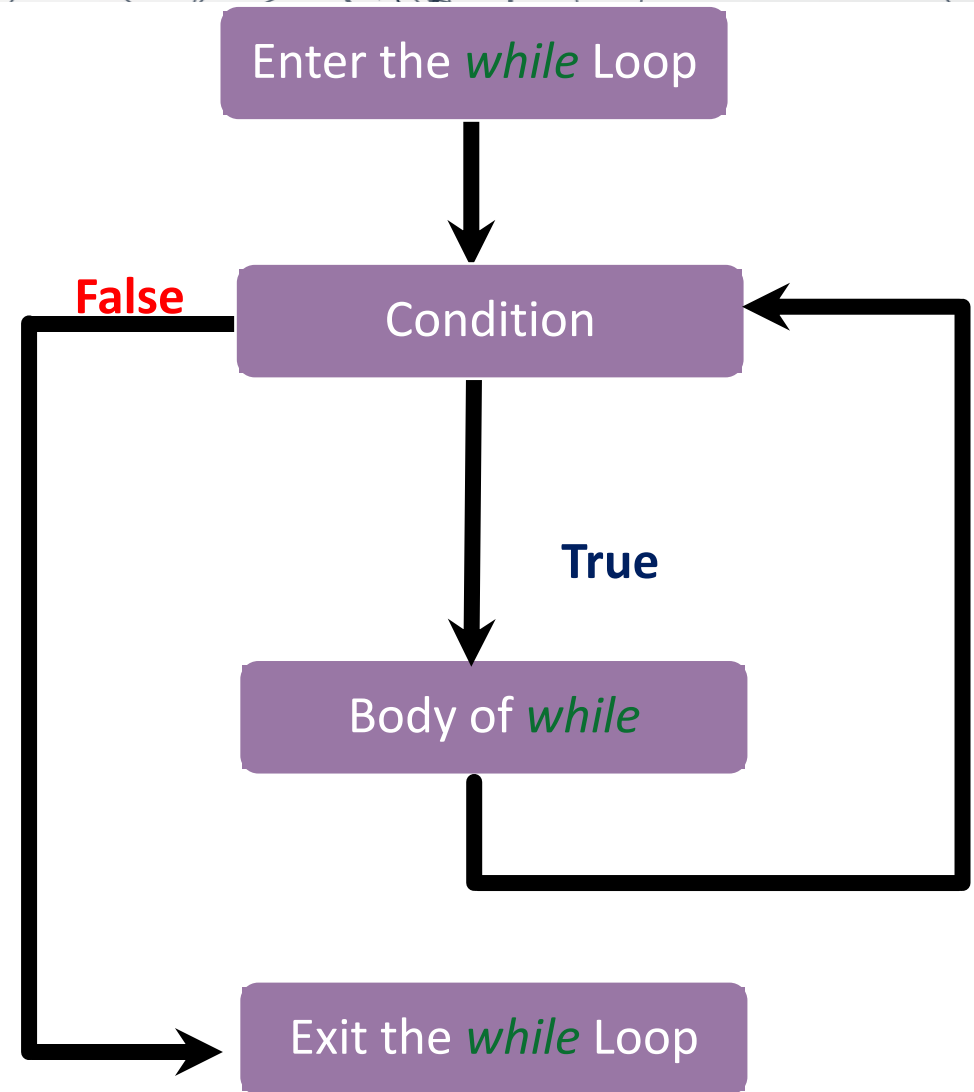- We create a *while* loop in Python using the '*while*' key-word!

While Loop – Syntax

while (*Boolean Condition*):
Statements

- Starts with '*while*' keyword.

- A *Boolean Condition*.

- A colon. (**:**)

- The line after the colon **must** be indented. (4 Spaces)

- Indented Statements.

While Loop – Flow Chart

- **As long as** the Boolean Condition evaluates as *True*, the *while* loop keeps on looping!

- **Once** the *Boolean Condition* evaluates as *False*, we exit the *while* loop!

Enter the *while* Loop

False

Condition

True

Body of *while*

Exit the *while* Loop

While Loop – Example

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

- Create an a variable and name it as "**num**" *before-hand*!

- Declare a *while* loop using the '*while*' keyword!

- ***As long as*** the *Boolean Condition* evaluates as true:
  - *Print* out the value of **num**.
  - Update the value of **num** by assigning the value of **num** + 1 to **num**

- ***Once*** the *Boolean Condition* evaluates as ***false***; we *exit* the loop!

TECHNION
Azrieli Continuing Education and
External Studies Division

Infinite Loops

- What is wrong with this loop?

```python
num = 5
while (num > 0):
        print(num)
```
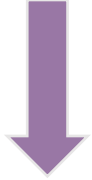
For Loop!

- A *for* loop in Python, helps us repeat a group of statements a *specified* number of times!

- It requires *three* features in order to work.

  → *(Iterating Variable, Statements, Sequence)*

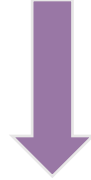- We declare a *for* loop in Python using the '*for*' key-word!

The Three Features

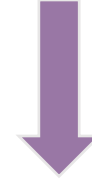- In order to use a *for* Loop in Python, it must consist of **three** features:

Iterating Variable   **Statement(s)**   **Sequence**

An indicator
For Starting State

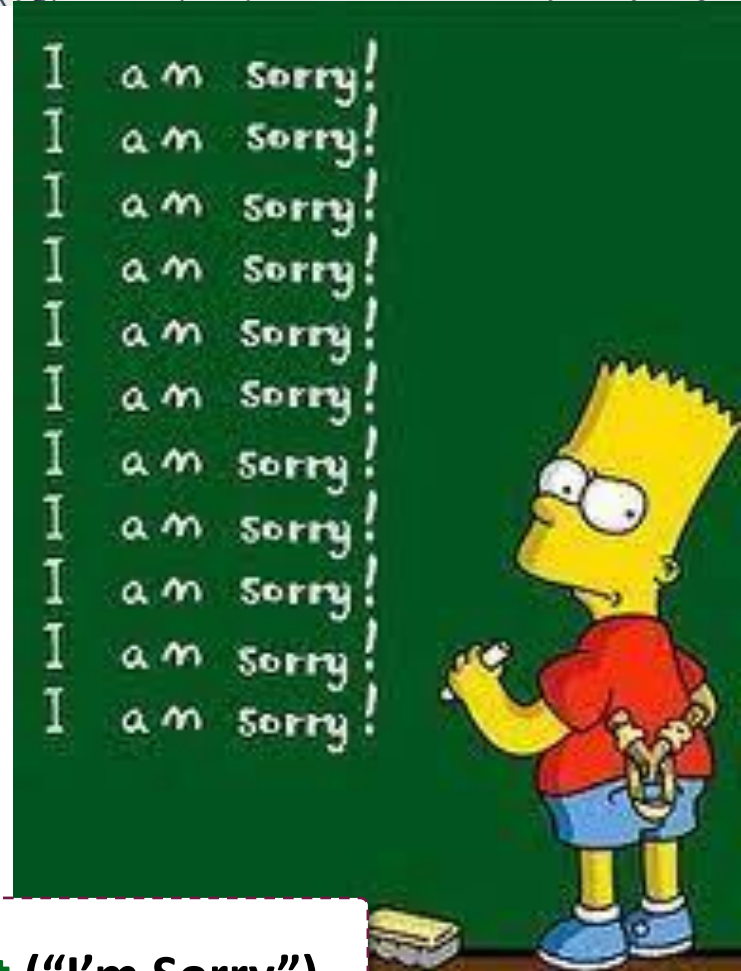What to do next

Something to Loop over

Iterating Variable

- An *iteration* is one step in a loop.

- Bart had to write down a hundred times that he is sorry.
→ Once Bart writes down for the first time "I am Sorry", he makes an iteration of the punishment.

  - An *Iterating Variable* is a variable used as an indicator for the
  - program to keep track of our loop iterations!

print ("I'm Sorry")

Statement

- *Statement(s)* is a block of code indented inside a *for* a loop.

- For each iteration of a loop, the desired block of code is being executed!

- Without mentioning any statement, we will face a *Syntax Error*!

Sequence

- A *sequence* is a collection of some objects.

- A *sequence* can be anything we can iterate over!

- For instance →

  - A string is a collection of characters that we can loop over it, character by character!
  - A simple integer/float *does not* have an iterator.

For Loop – Syntax

> *for iterating_var in sequence:*
> *Statement(s)*

- Starts with '*for*' keyword.

- A *Boolean Condition*.

- A colon. (**:**)

- The line after the colon **must** be indented. (4 Spaces)

- Indented Statement.

Implementation of the three-features

```
for letter in 'AI Accelerator':
    print(letter)
```

- *Iterating Variable* → letter

- *Sequence* → '*AI Accelerator*'

- Statement → *print*(letter)

For Loop – Flow Chart

- *Initialization* is the part where we create the loop.

- *Initialization* is also the part where we assign the **first** value into the *iterating variable*.

- **Once** the *Boolean Condition* evaluates as *True*, we enter the body of the *for* loop and execute the *statements*!

- **Once** the first iteration is done, we update the *iterating variable* and check the *Boolean Condition* once again!

- **Once** the *Boolean Condition* evaluates as *False*;we exit the *for* loop!

Start

Initialization

False

Condition

True

Body of for

Update

Exit the *for* Loop
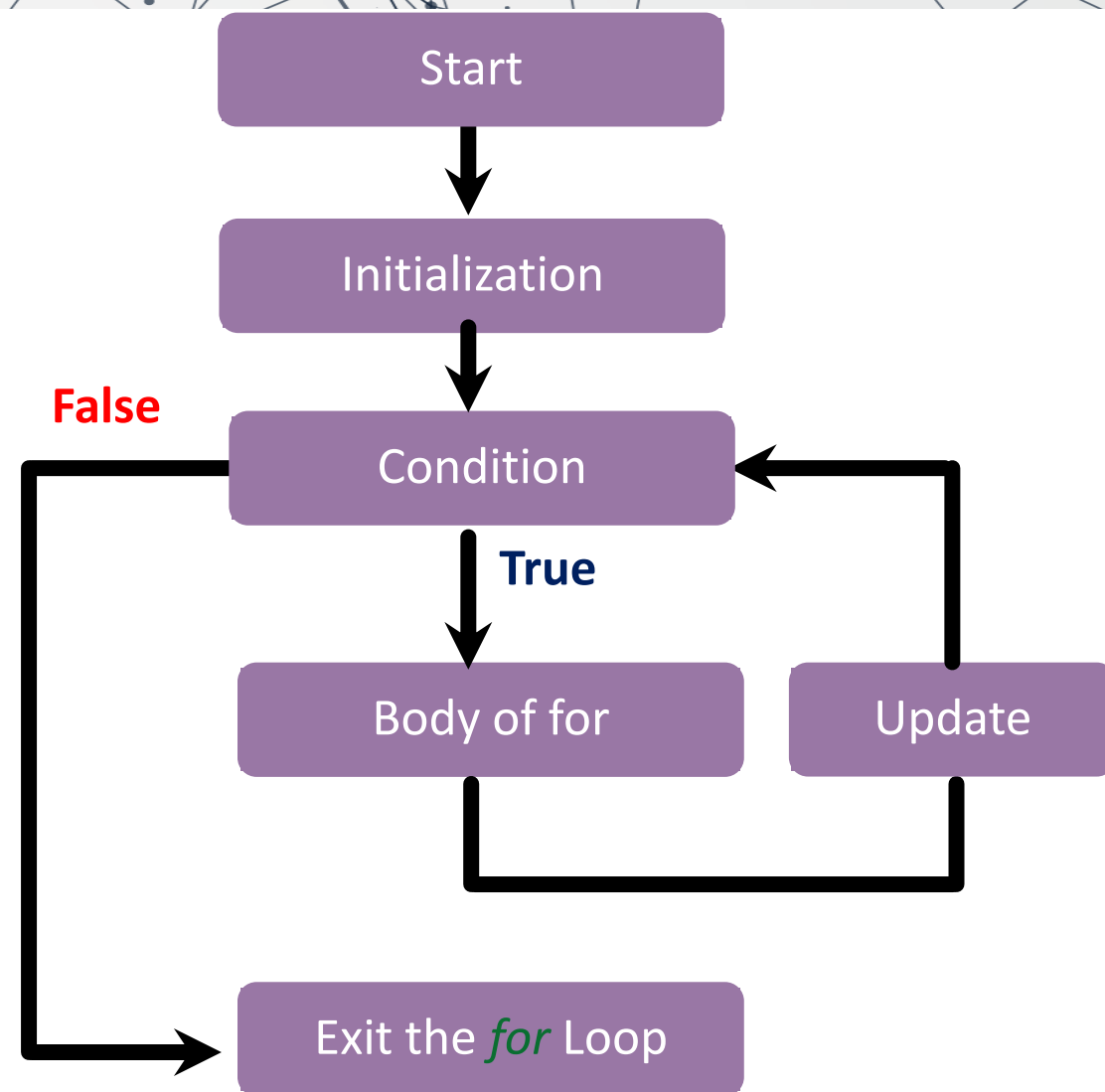
Looping Over Numbers

- Many times, we would like to loop over numbers in order.

- In the previous example, we created a *sequence*, using characters of a string.

- What if we want to create a *sequence* of numbers?

- Let us get to know the *range*() function!

range() Function!

Using the *range()* function, we can create a *sequence* of numbers.

For instance →

　　　range(10) creates a *sequence* of numbers between *0* and *10.*

Counting in Python always *starts with 0* and *ends on **n**-1*, with ***n*** being the length of the object being counted.

range(10) →

　　　　　(0,10) →

　　　　　　　[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] →

　　　　　　　　　starts with 0, ends with 10-1

range() Function – List View

- We can look at a *range*() as a *list*!

- Calling out *range*(10) returns as:

-      *range*(0, 10)

- We can use the *list*() function to cast a range *sequence* into a *list*!

```
list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

For Loop – Example

> *for* number *in range*(10) :
>   *print*(number)

- Declare a for loop using the '*for*' keyword!

- Create an *Iterating Variable* "**number**"!

- Create a *sequence* of digits from 0 to 9, using *range(10)*.

- In the *first iteration*, the *first* value in the *sequence* is assigned to the *number*. (0)

- *For each iteration* of the *sequence*, we execute the statement: print out the value of stored inside *number*.

- The value stored inside the *number* is set to the next value stored inside the *sequence*!

TECHNION
Azrieli Continuing Education and
External Studies Division

- Looping Over Numbers - Demonstration

```python
for number in range(10):
    print(number)
```

```python
for number in [0,1,2,3,4,5,6,7,8,9]:
    print(number)
```

→ Same Output!

Looping Over Numbers – Bart!

Bart must write down a hundred times "I'm Sorry"

↓

```
for number in range(100):
    print(number, "I'm Sorry!")
```

range() Function - Arguments

Instead of only specifying the *upper limit* of the *range,* we can also specify the *bottom limit*:

      range(a, b)

The *sequence* is going to start from **a** to **b-1**, For Instance:

      range(3,10) →   Creates a *sequence* of digits starting from 3 and ends on 9.

We can also add a *third argument*, to specify the exact number to *increment/decrement* by:

      range(a, b, c)

For instance:

      range(2, 11, 2) → *Creates a sequence of digits starting from 2 and ends on 10. ([2, 4, 6, 8, 10])*

For Loop Variable Names

- *Iterating Variables* i, j, and k are the conventional variable names used for loops.

- That being said, if the iterations have meaning, a more indicative name should be used, to improve readability:

```
for celebrity name in ['The Rock', 'Bruno Mars', 'Cristiano Ronaldo']:
        print(celebrity name)
```

When do we use FOR, and when do we use WHILE?

- Rule of thumb:
  - If we know ahead of the loop execution how many loops we will want, use *for*.
  - If we will only know when to stop the loop during the loop execution, use *while*.

- Nested Loops

Remember Nested *if* Condition?
    (an *if* statement inside an *if* statement)

```
if (flavor == "Chocolate"):
    print("Yes!")
    if (price < 15):
        print("Buy")
```


wait whut?

A loop inside another loop is called a **Nested Loop**!

Nested Loops

- Python allows us to use one loop inside another loop!
- A Nested loop can be either a Nested '*for*' loop or a Nested '*while*' loop!
- A Nested loop can also be a mix of the two.

Nested for Loop – Syntax

```
for iterating_var in sequence:
        for iterating_var2 in sequence2:
            Statements
    Statements
```

- A colon for both loops. (**:**)

- The line after the colon **must** be indented. (4 Spaces)

- Indented Statements for each loop!

Nested for Loop – Logic

> *for* iterating_var *in sequence:*
>   *for* iterating_var2 *in sequence2:*
>   *Statements*
>   *Statements*

- Declare a '*for*' loop. (We are going to refer to this as "**outer loop**")

- Declare another '*for*' loop inside the *outer* '*for*' loop. (We are going to refer to this as "***inner loop***")

- For every iteration of the **outer** loop, the **inner** loop executes.

- The **inner** loop continues until <u>all iterations</u> over the given *sequence* are complete and then heads back to the **outer** loop.

TECHNION
Azrieli Continuing Education and
External Studies Division

Nested while Loop – Syntax

```
while (Condition):
        while (Condition):
            Statements
    Statements
```

- A colon. (:)
- The line after the colon **must** be indented. (4 Spaces)
- Indented Statements.

Nested while Loop – Logic

while (Condition):
                while (Condition):
                        Statements
              Statements

- Declare a '*while*' loop. (*outer loop*)

- Declare another '*while*' loop inside the *outer* '*while*' loop. (*inner loop*)

- *Once* the *outer* loop condition evaluates as *True*, the *inner* '*while*' loop is being called.

- *Once* the *inner* loop condition evaluates as *True*, the *inner* statements are executed.

- *As long as the inner* loop condition evaluates as *True*, the *inner* loop keeps iterating.

- *As long as the outer* loop condition evaluates as *True*, the *outer* loop keeps iterating.

pass

- The *pass* command is a place-holder.

- It continues execution with the next line.

- This is very useful when a *loop* is written, in which an indented code block is *needed* but we want to deal with it *later*.

```
for i in range(100):
        pass
```
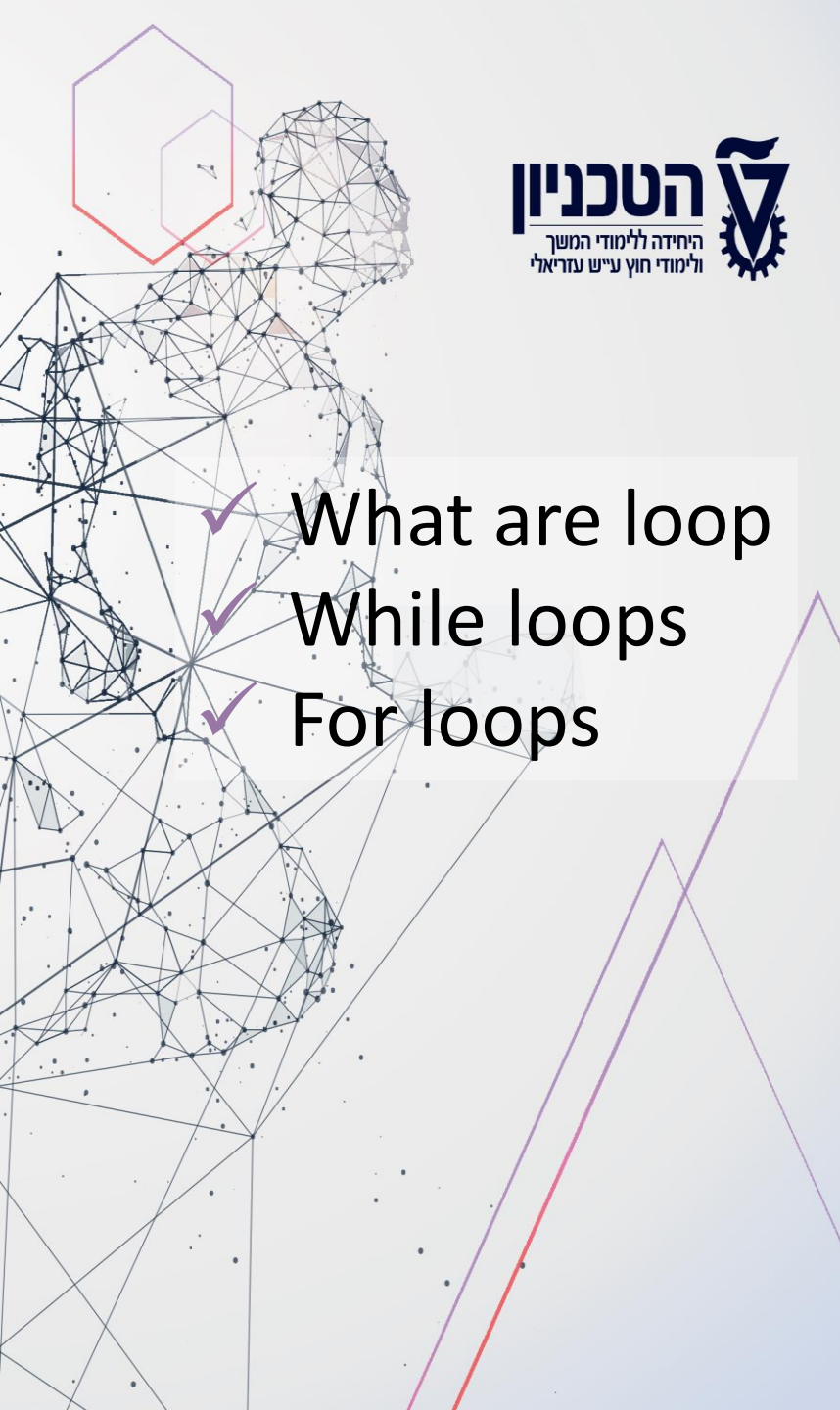
break

- The *break* command *stops* the current loop and jumps *out of it*.

- Example:

```
for i in range(100):
    if (i == 3):
        break
    print (i)
```

continue

- The *continue* command *stops* the current iteration in the loop, *continuing* with the *next* iteration.

- Example:

```
for i in range(10):
    if (i % 3):
        continue
    print (i)
```

What are loop
While loops
For loops

Thank you