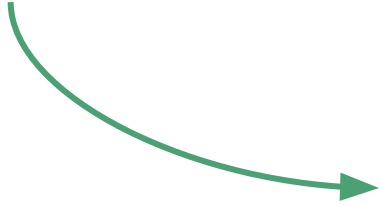


# Computing GC content with for loops

---

BILD 62

Quick check in...



<https://www.menti.com/blhpa9c4jpki>

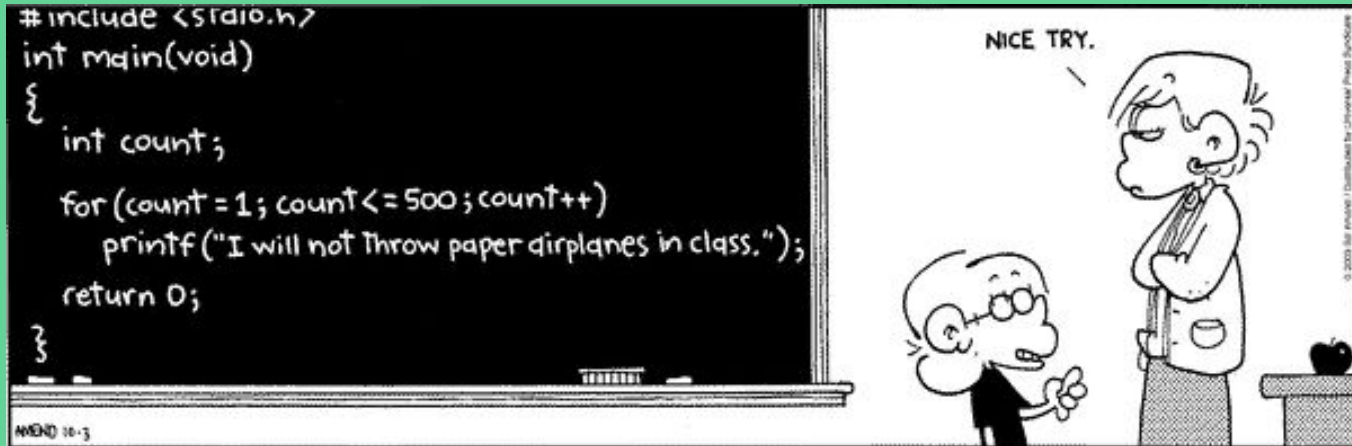
# Recap of last time:

## computing GC content with conditionals

- Can we do this with `elif` statements?
- Can we alert the user if the function gets incorrect input (a string of incorrect length)?

A **loop** is a procedure to repeat a piece of code.  
(another way of saying this is that it **iterates** through code)

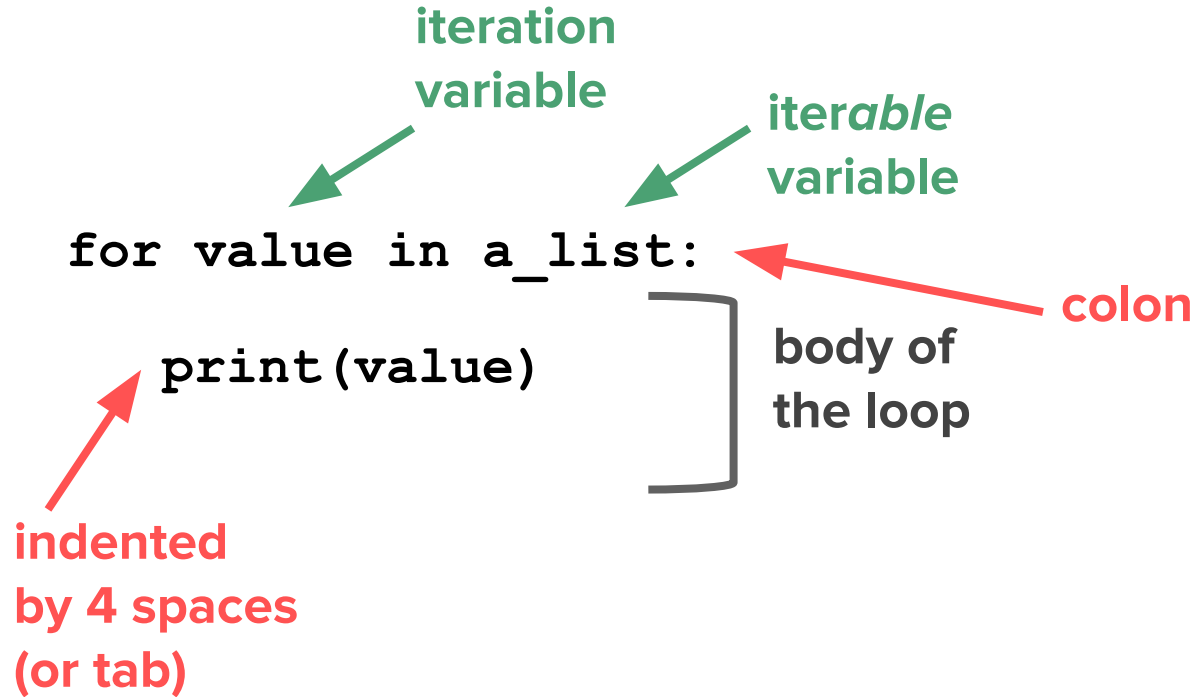
- Loops enable you to re-run blocks of code for as many times as you need.
- Python has two main ways to run loops: **for** & **while**



# Objectives for this week

- Write a **for** loop to iterate over elements in an object
  - Use a **counter** in a loop
  - Use a **for** loop to count GC content and CCAAT boxes in a DNA string
  - Write **while** loops and implement **continue** and **break**
  - Develop a **growth mindset** towards your programming growth
-

# for loop syntax



The diagram illustrates the syntax of a Python for loop. The code snippet is `for value in a_list:` followed by an indented line `print(value)`. Annotations include: a green arrow pointing to `value` labeled "iteration variable"; a green arrow pointing to `a_list` labeled "iterable variable"; a red arrow pointing to the colon `:` labeled "colon"; a red arrow pointing to the indentation of `print(value)` labeled "indented by 4 spaces (or tab)"; and a bracket under the indented line labeled "body of the loop".

```
for value in a_list:
    print(value)
```

iteration variable

iterable variable

colon

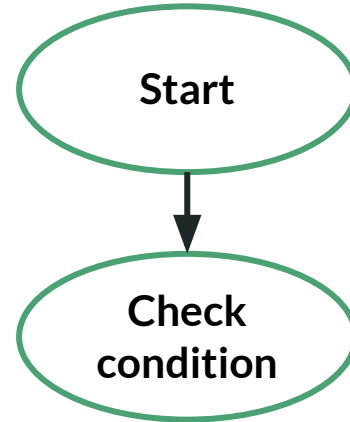
body of the loop

indented by 4 spaces (or tab)

A **for loop** is a procedure to repeat code for every element in a sequence.

# for loop syntax

```
a_list = [1,2,3]  
  
for value in a_list:  
    print(value)
```



# for loop syntax

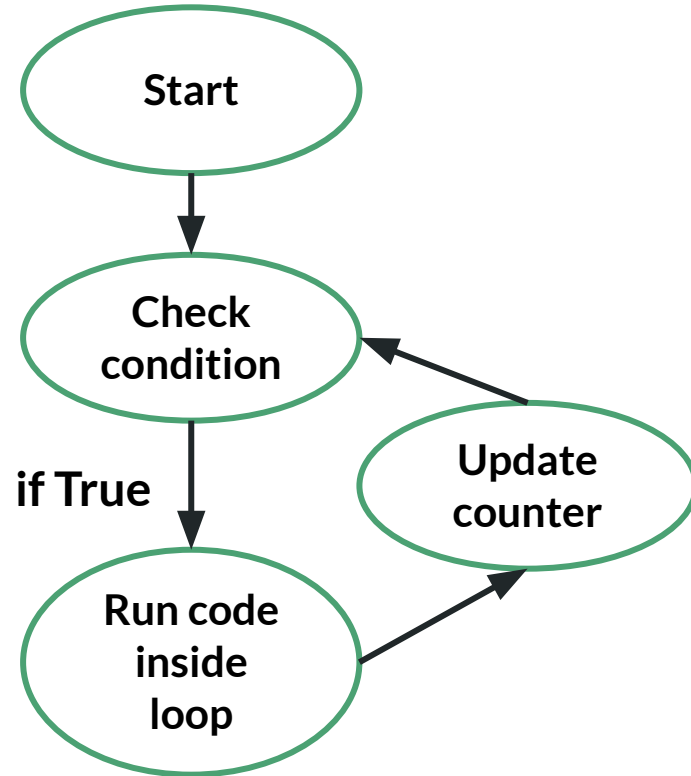
```
a_list = [1,2,3]
```

```
for value in a_list:
```

```
    print(value)
```

1

output





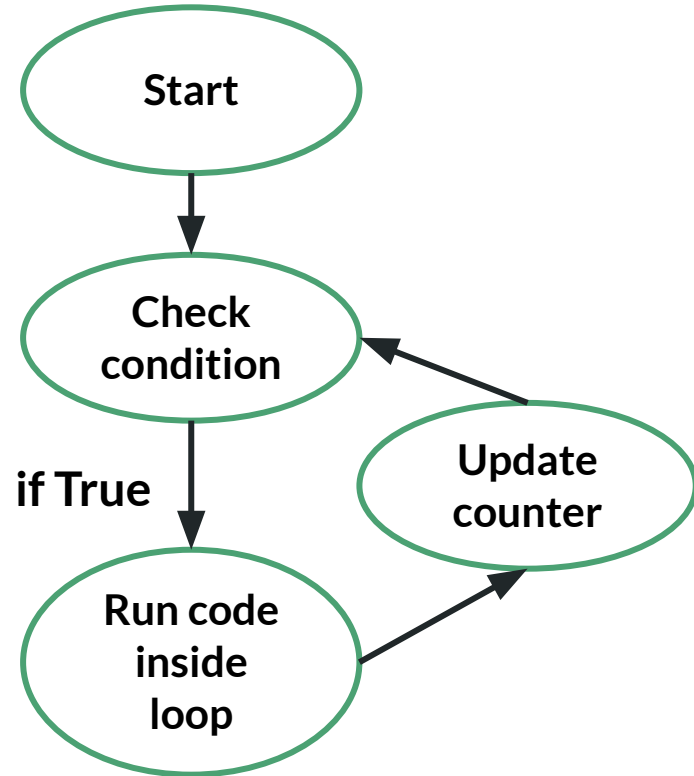
# for loop syntax

```
a_list = [1,2,3]
```

```
for value in a_list:
```

```
    print(value)
```

```
1 |  
2 | output
```

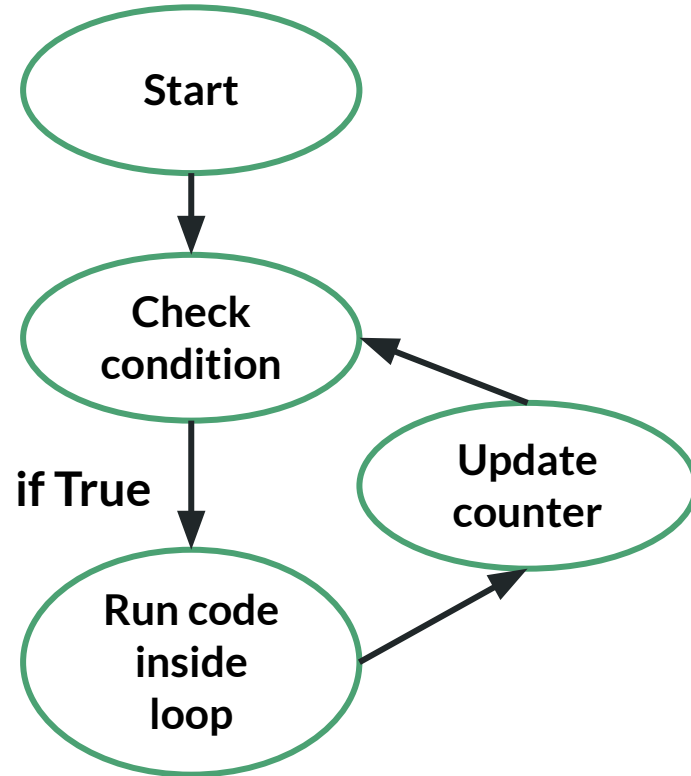


# for loop syntax

```
a_list = [1,2,3]

for value in a_list:
    print(value)
```

```
1 |
2 | output
3 |
```



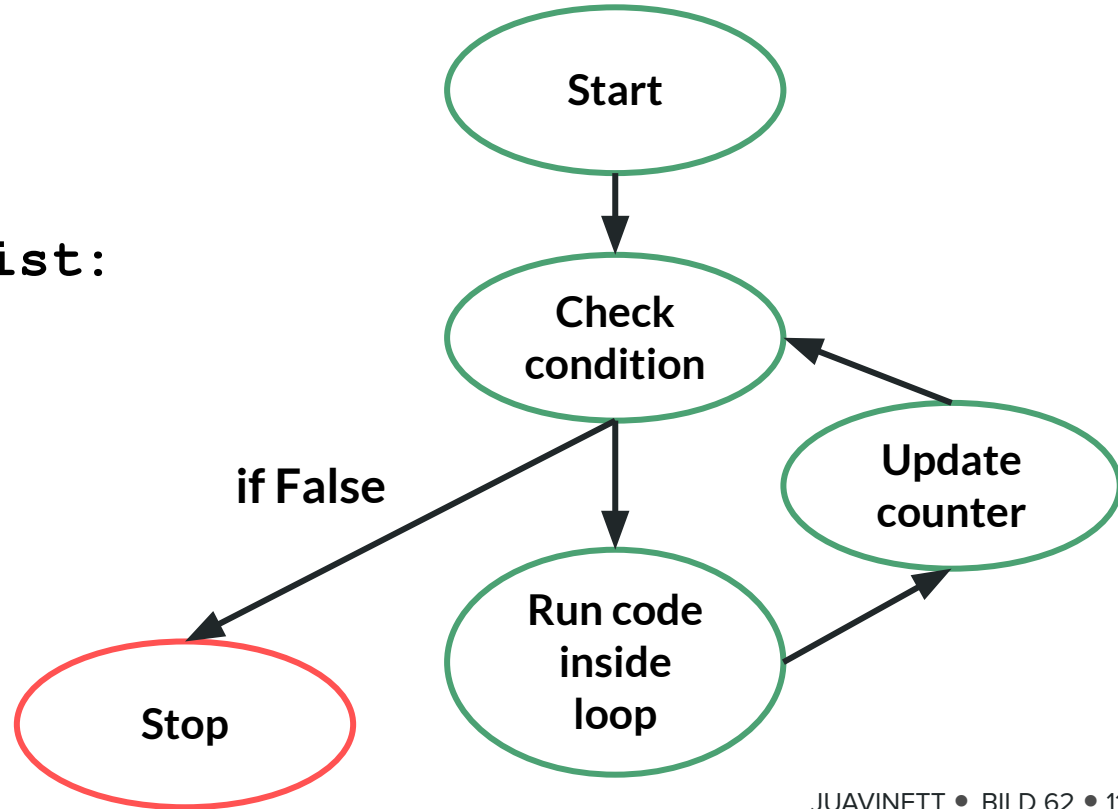
# for loop syntax

```
a_list = [1,2,3]

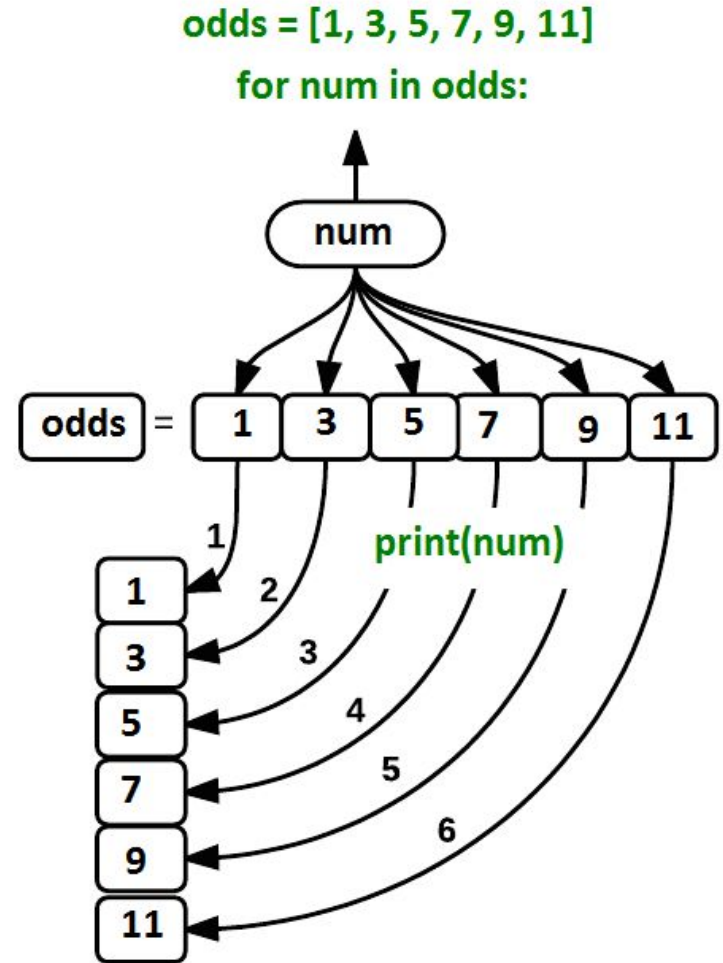
for value in a_list:

    print(value)
```

```
1 |
2 | output
3 |
```



Another way to visualize working  
through a loop  
([source](#))



# efficiency benefit of `for` loops

Each of these would accomplish the same thing:

**Option #1: 2+ lines of code**

```
for value in a_list:  
    print(value)
```

**Option #2: as many lines of code  
as there are list entries**

```
print(a_list[0])  
print(a_list[1])  
print(a_list[2])  
...
```

**Second task:** count the # of “CAT” boxes (CCAAT) in a string of DNA.

The “CAT” box generally appears near the spot where transcription begins!



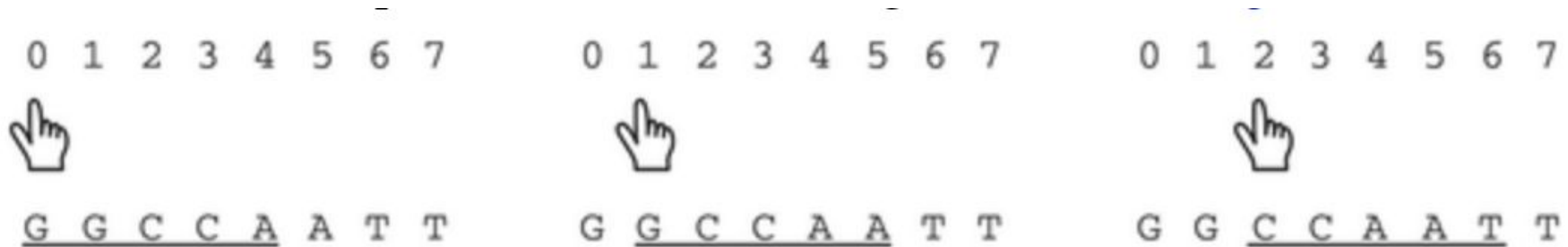
```
>>> countCCAAT( 'GGCCAATTGCCAAT' )
```

```
>>> 2
```

# We can also loop over a list of indices!

Let's say we want to look for a “CAT” box, a common motif in DNA, with the sequence CCAATT

Since we want to look at a **slice** of DNA, rather than looping through individual items in the string, we need the indices.

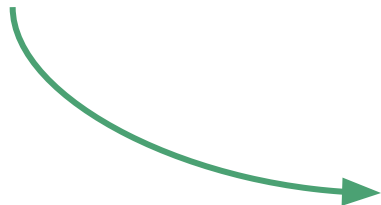


Into the notebook...



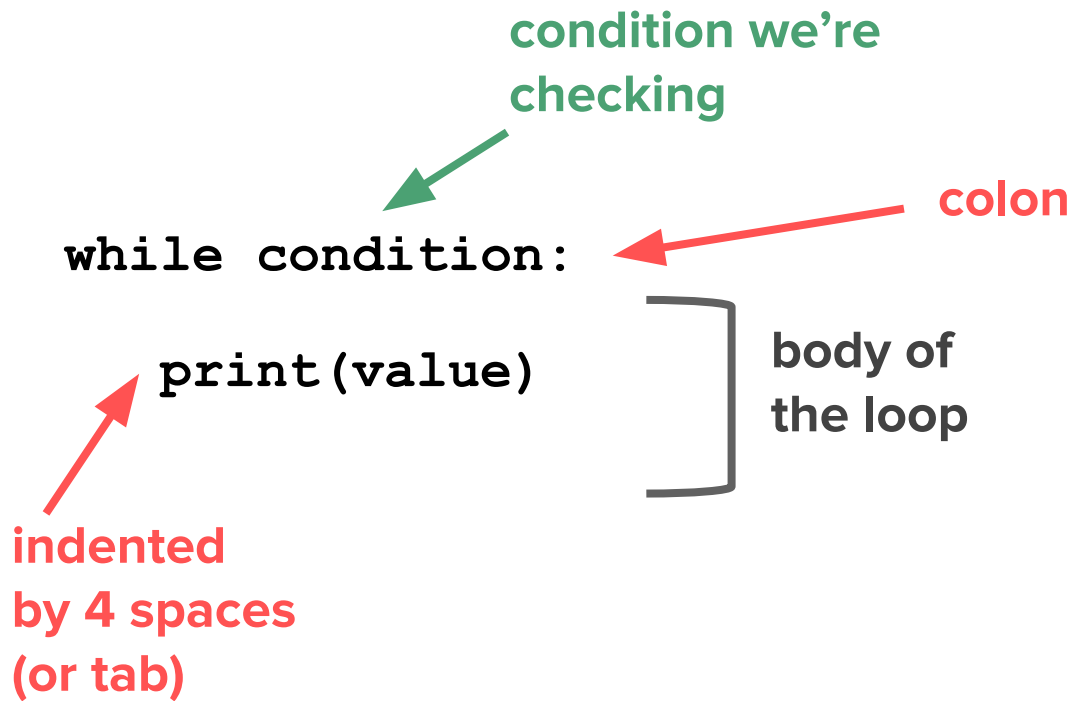


Quick check in...



<https://www.menti.com/blhpa9c4jpki>

# while loop syntax



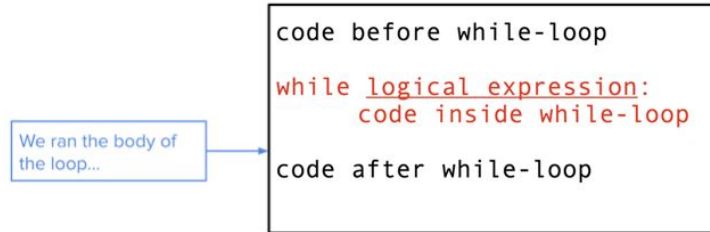
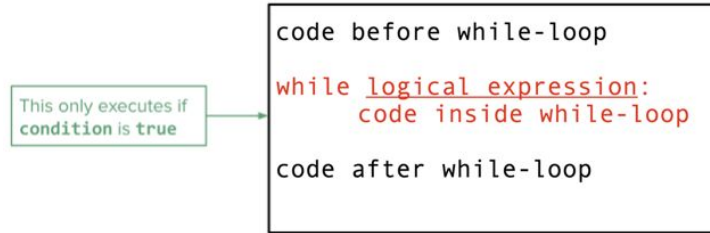
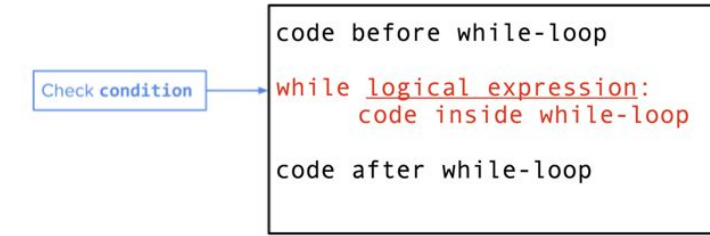
The diagram illustrates the syntax of a while loop with the following code and annotations:

```
while condition:  
    print(value)
```

- condition we're checking**: A green arrow points to the `condition` part of the `while` statement.
- colon**: A red arrow points to the colon (`:`) at the end of the `while` statement.
- body of the loop**: A bracket on the right side of the `print(value)` line indicates the code block that is repeated.
- indented by 4 spaces (or tab)**: A red arrow points to the indentation of the `print(value)` line.

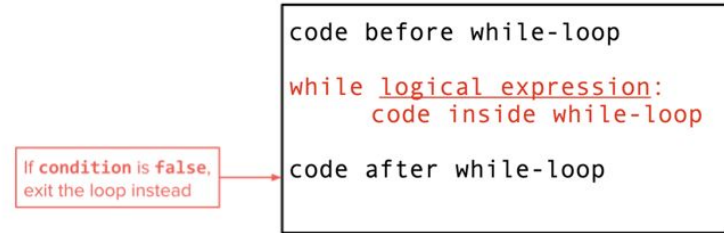
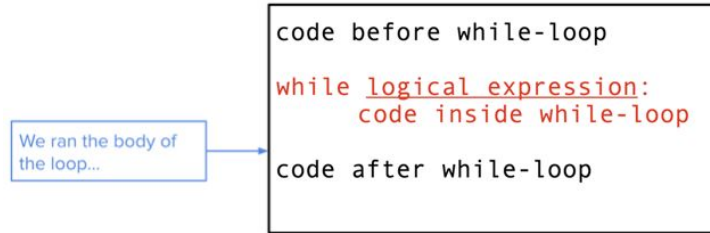
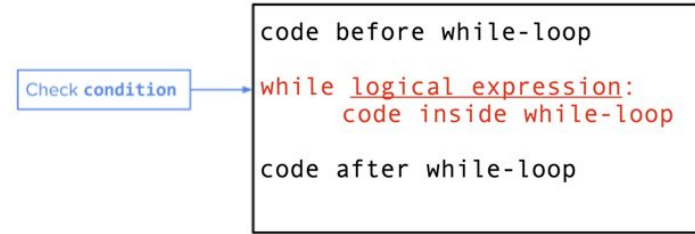
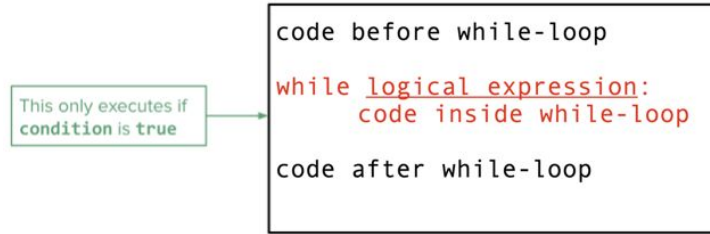
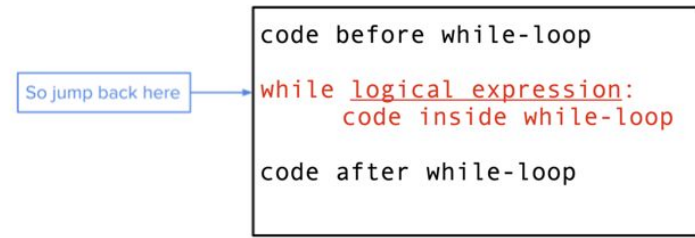
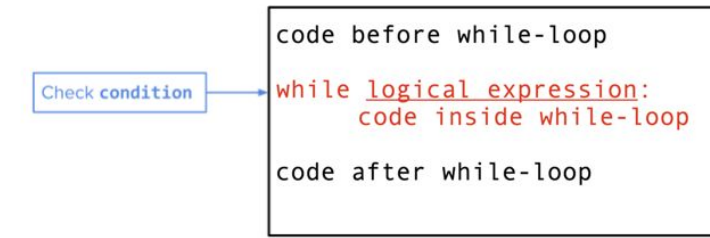
While this condition is true, the loop will run!

It will repeat until the condition is no longer True.



## Order of execution in a while loop

(Image credit: Niema Moshiri & Sabeel Mansuri)



## Order of execution in a while loop

(Image credit: Niema Moshiri & Sabeel Mansuri)

# Mindsets about intelligence (and programming)

---

## Fixed Mindset

Human traits (including programming skills) are ***fixed/innate***.

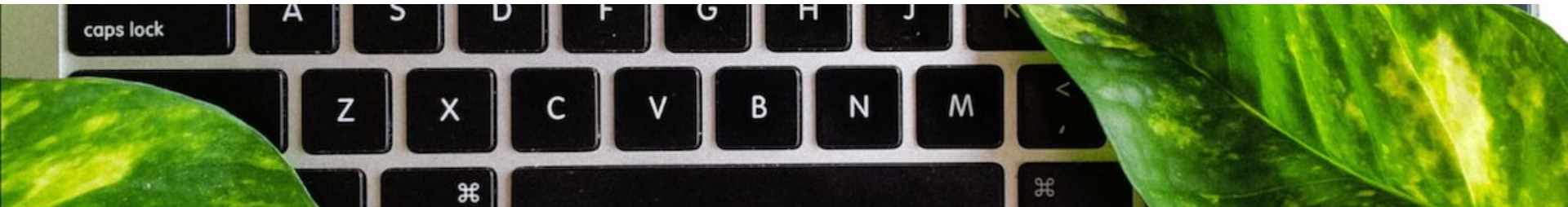
Fixed mindset about programming:

You have a certain amount of programming ability and *can't do anything to change it*.

## Growth mindset

Human traits (such as programming skills) are ***malleable and can be shaped/developed***.

Programming skill can be developed through personal effort, good learning strategies, and feedback.



# Which of these are indicative of a growth mindset?

<b>Views on effort</b>	Effort is seen as an important component of learning	Effort is seen as sign of weakness
<b>Goal orientation</b>	<b>Performance goal orientation</b> (picks challenges they know they can meet, uses them to prove yourself to others)	<b>Mastery goal orientation</b> (picks increasingly more difficult challenges)
<b>Attribution of failure</b>	Attributes failure to lacking ability or blames others or the circumstances	Attributes failure to not having put in enough effort or preparation, or having used ineffective strategies
<b>Strategies</b>	Increases effort, tries new things, asks for help from others	“Learned helplessness” or tries to persevere with the same (ineffective) study strategy
<b>Feedback</b>	Avoids feedback, acts defensively	Seeks out feedback
<b>Results</b>	Persistence, overcomes initial challenges, finds ways around it	Loses interest and withdraws in response to challenges, self-sabotage



How do these individuals demonstrate growth mindsets?





Learn more about their beginnings in programming!



# Topics from this lecture & corresponding notebook

- Syntax of **for** and **while** loops
- How to iterate through strings, lists, and dictionaries
- Using a counter to count loop iterations
- Looping over lists of indices
- Calling functions within functions
- Using **break** to interrupt a loop, and **continue** to skip a loop
- Functions we learned: **range()** , **enumerate()**

# Resources

[Stepik Introduction to Python book, Chapter 3](#)

[Whirlwind Tour of Python: Control Flow](#)