

Computing GC content with for loops

BILD 62

Recap of last time:

computing GC content with
conditionals

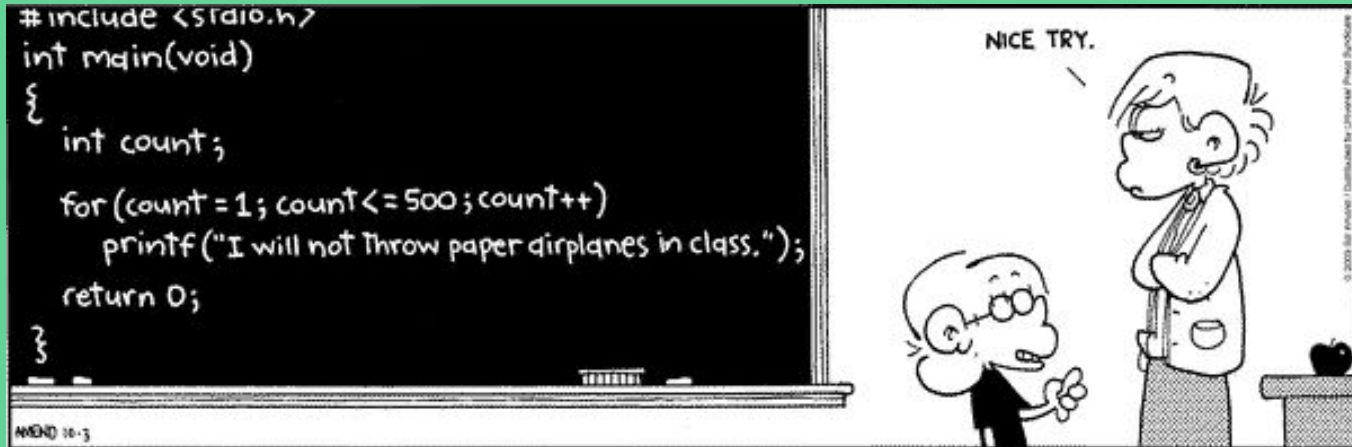
Can we do this with `elif` statements?

Can we alert the user if the function gets
incorrect input?

A **loop** is a procedure to repeat a piece of code.

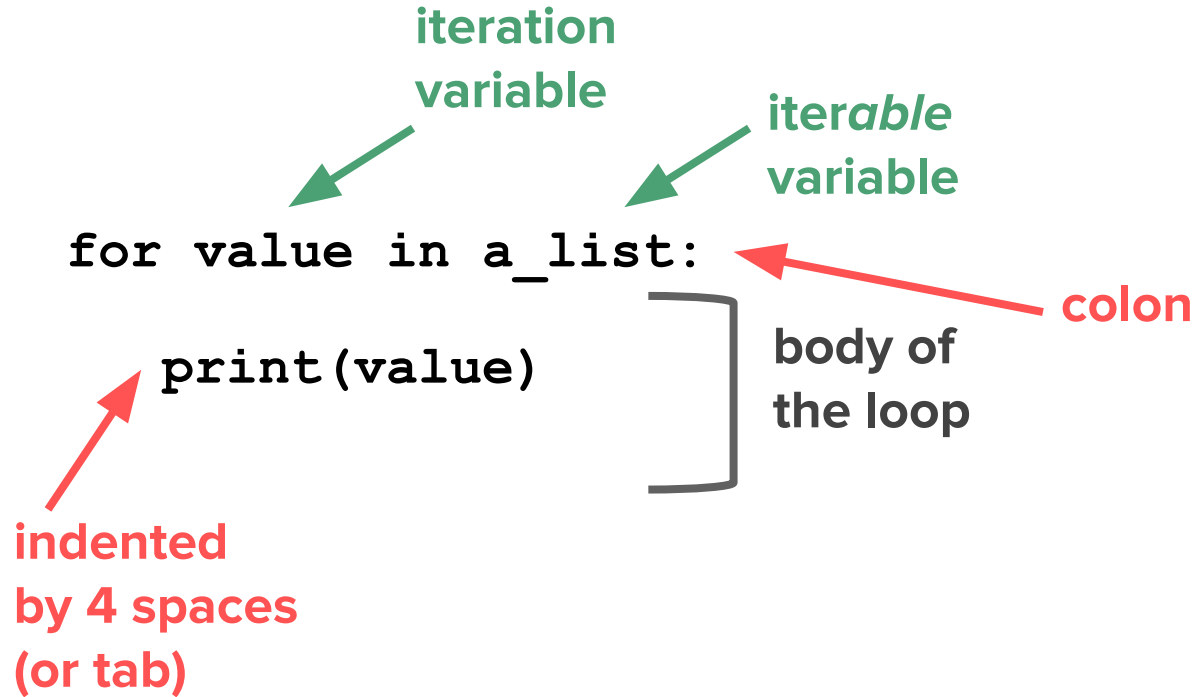
Loops enable you to re-run blocks of code for as many times as you need.

Python has two main ways to run loops: **for** & **while**



Bill Amend, FoxTrot, October 3, 2003

for loop syntax



The diagram illustrates the syntax of a Python for loop. It shows the code `for value in a_list:` on the first line and `print(value)` on the second line, which is indented. 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 on the right side of the indented line labeled "body of the loop".

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

iteration variable

iterable variable

colon

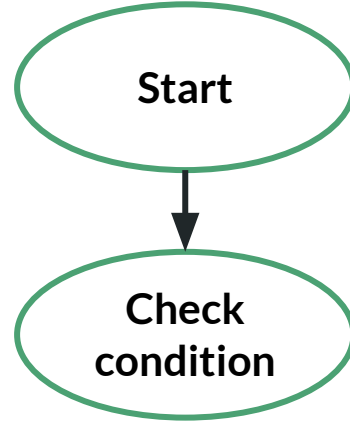
body of the loop

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

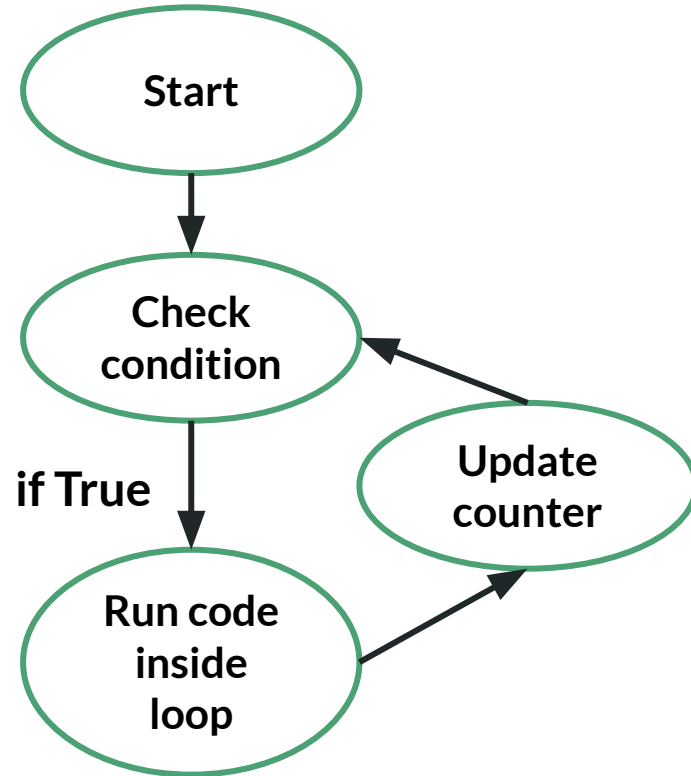
```
a_list = [1,2,3]
```

```
for value in a_list:
```

```
    print(value)
```

```
1
```

```
| output
```



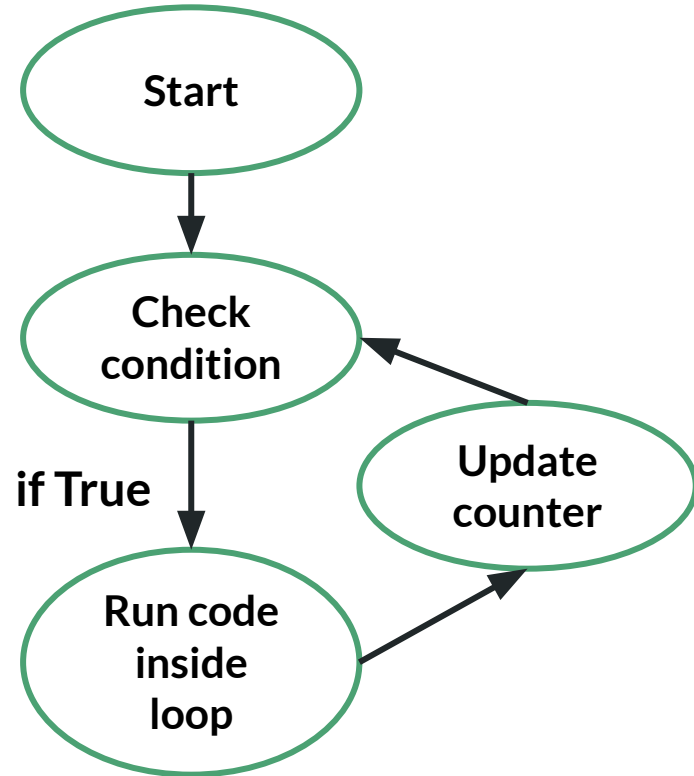
for loop syntax

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

```
for value in a_list:
```

```
    print(value)
```

```
1 |  
2 | output
```

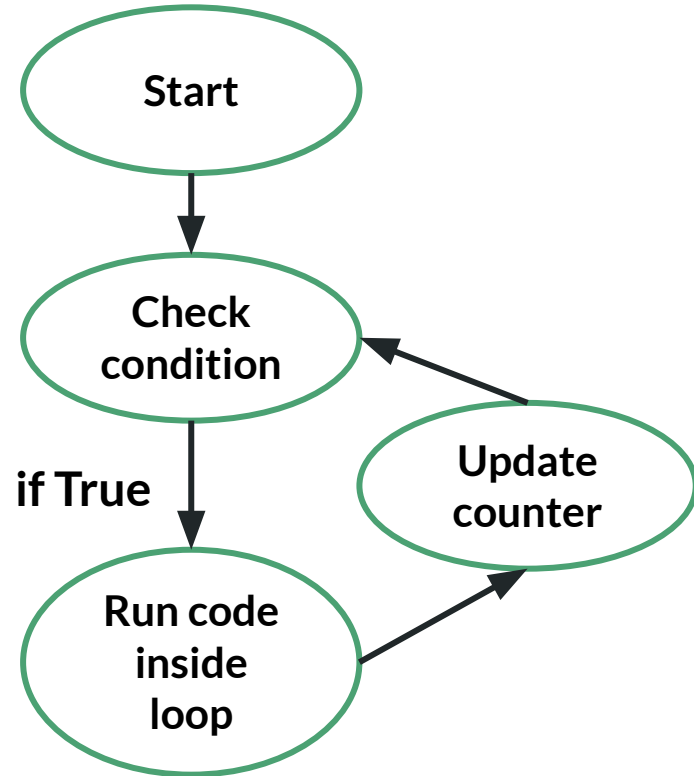


for loop syntax

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

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

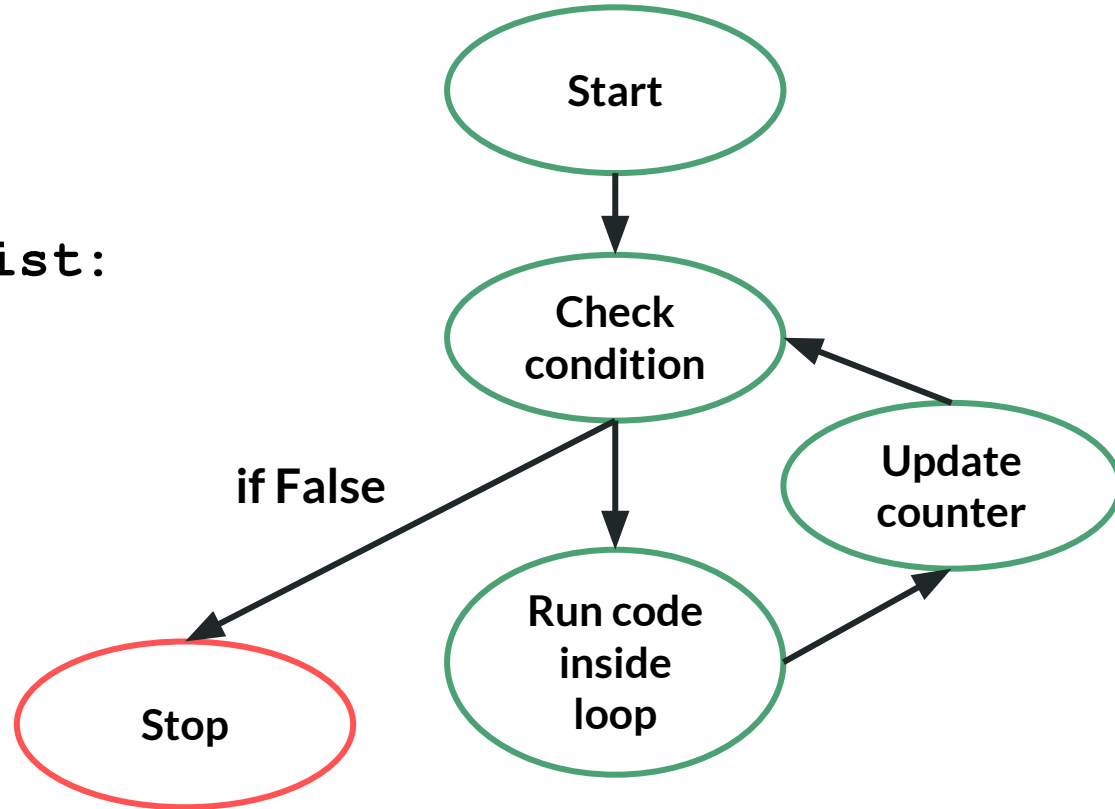
```
1 |
2 | output
3 |
```



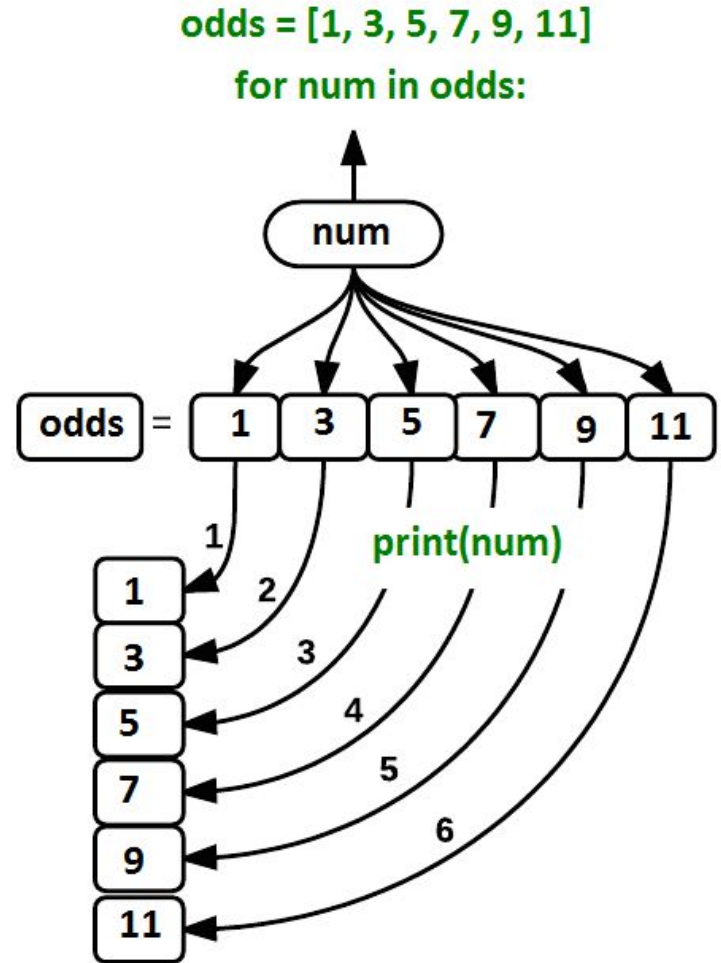
for loop syntax

```
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!

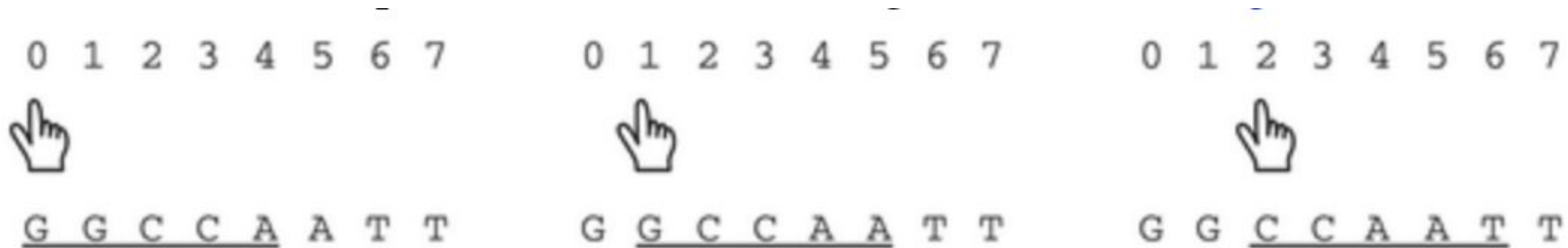


```
>>> countCCAAT ( 'GGCCAATTGCCAAT' )  
>>> 2
```

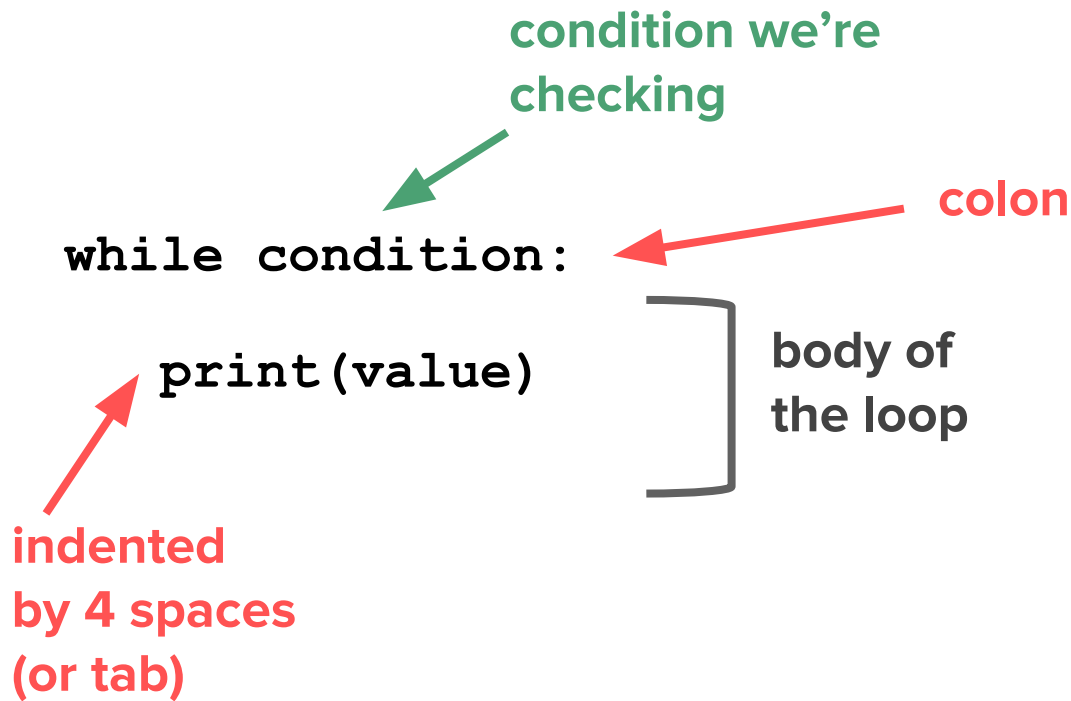
We can also loop over a list of indices!

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.



while loop syntax



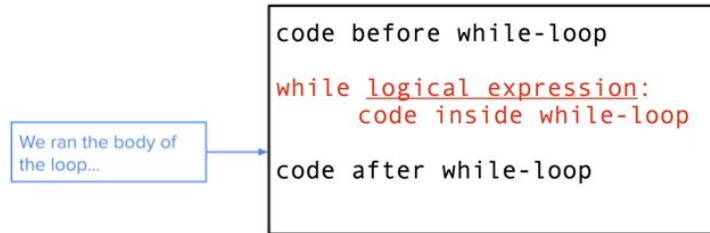
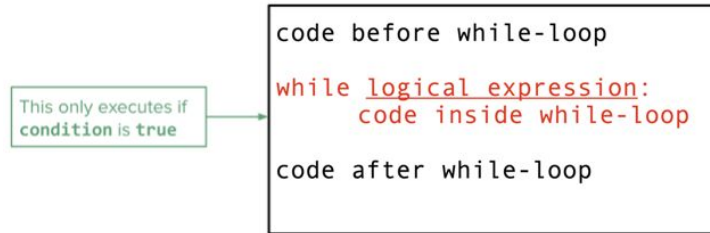
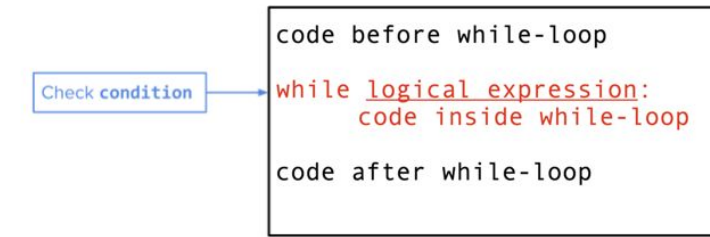
The diagram illustrates the syntax of a while loop with the following code and annotations:

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

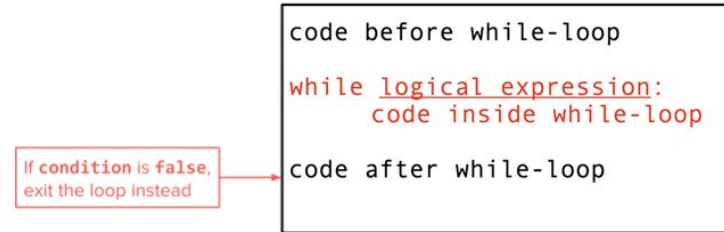
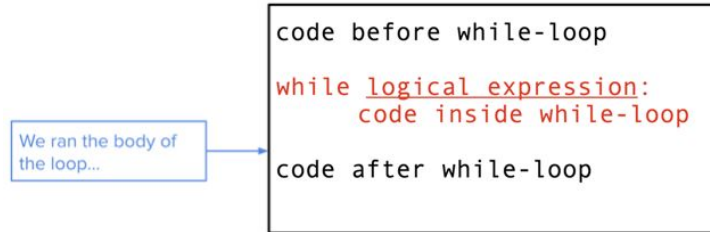
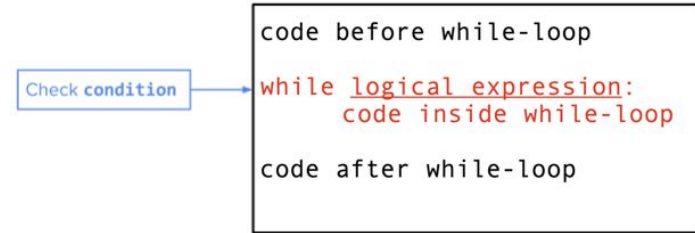
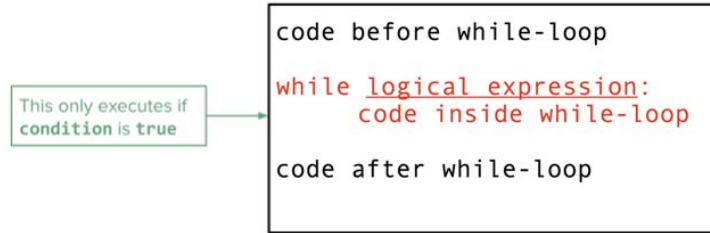
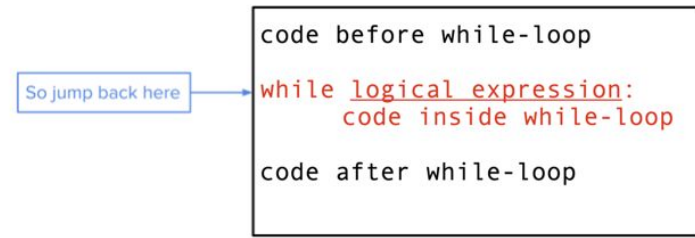
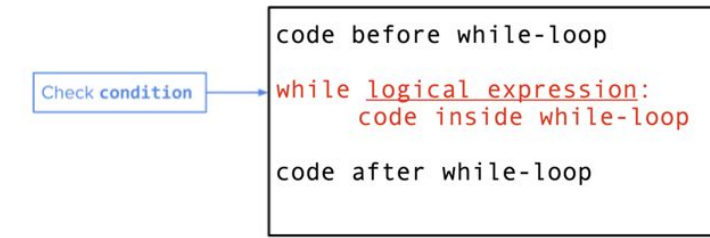
- condition we're checking**: A green arrow points to the `condition` in `while condition:`.
- colon**: A red arrow points to the colon `:` in `while condition:`.
- body of the loop**: A bracket on the right side of the indented line `print(value)` is labeled "body of the loop".
- indented by 4 spaces (or tab)**: A red arrow points to the indentation of `print(value)`.

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 (from [Stepik](#))



Order of execution in a while loop (from [Stepik](#))

Resources

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

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