Computing GC content with for loops

BILD 62

Warm up challenge: guessing game

Ask the user: “What’s my favorite food?” If it’s the same as yours, respond, “Yep, delicious!” If not, say “Nope, you’re wrong.” Regardless, tell the user “Thanks for playing.”

Hint: You can use the `input` function to ask the user for a string input. For example,

```
response = input("What's my name?”)
```

Working with a team, pseudocode this on the whiteboard (write out your general idea for the code), and then write this code in Python (either in a Jupyter Notebook or a .py script).

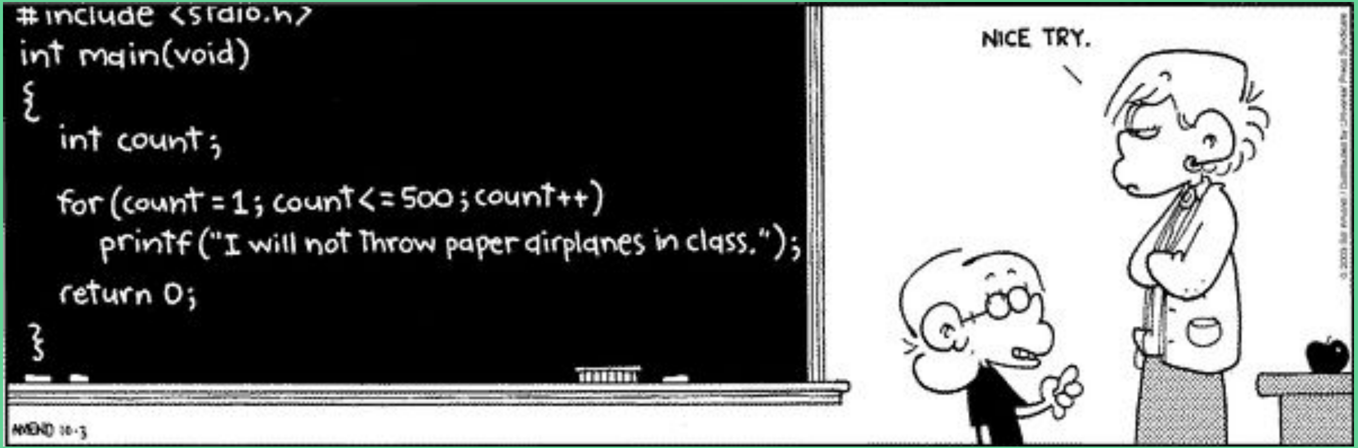
Recap of last time: computing GC content with conditionals

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`**

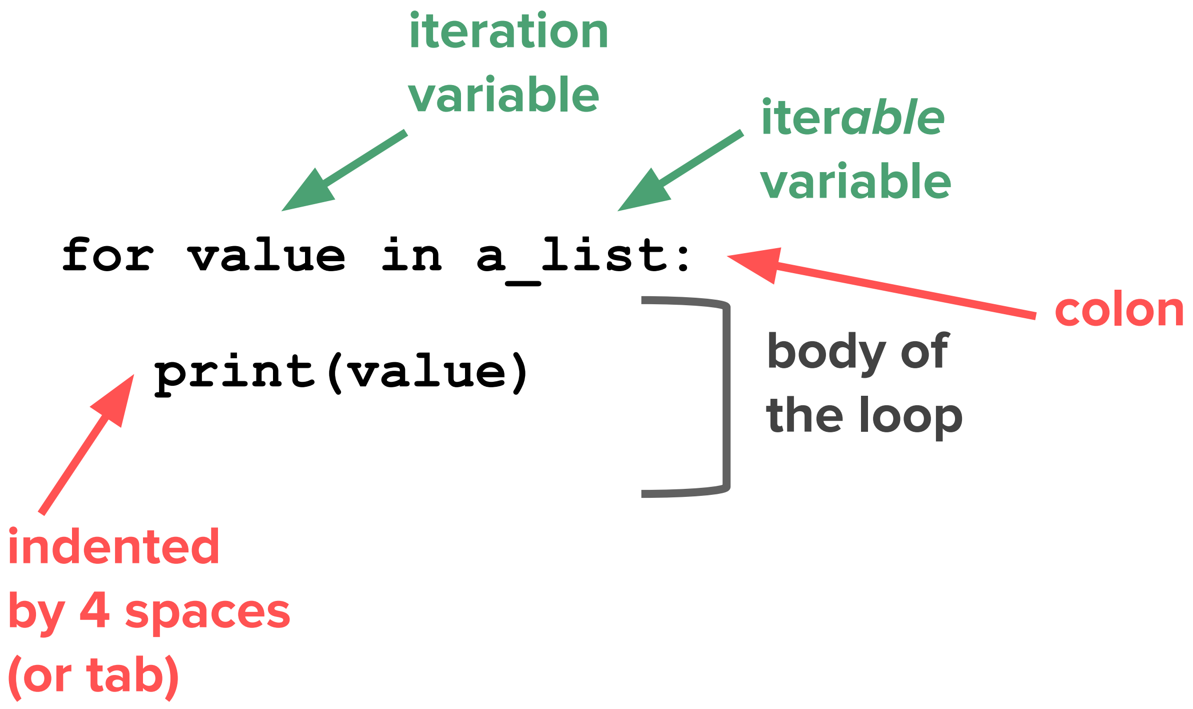


```
#include <stdio.h>
int main(void)
{
   int count;

   for (count = 1; count <= 500; count++)
      printf("I will not Throw paper airplanes in class.");
   return 0;
}
```

Bill Amend, FoxTrot, October 3, 2003

`for` loop syntax



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

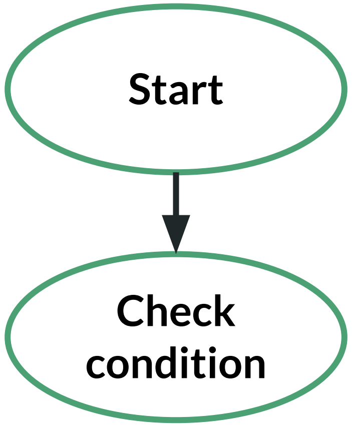
A **for loop** is a procedure to repeat code for every element in a sequence.

`for` loop syntax

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

for value in a_list:

    print(value)
```



`for` loop syntax

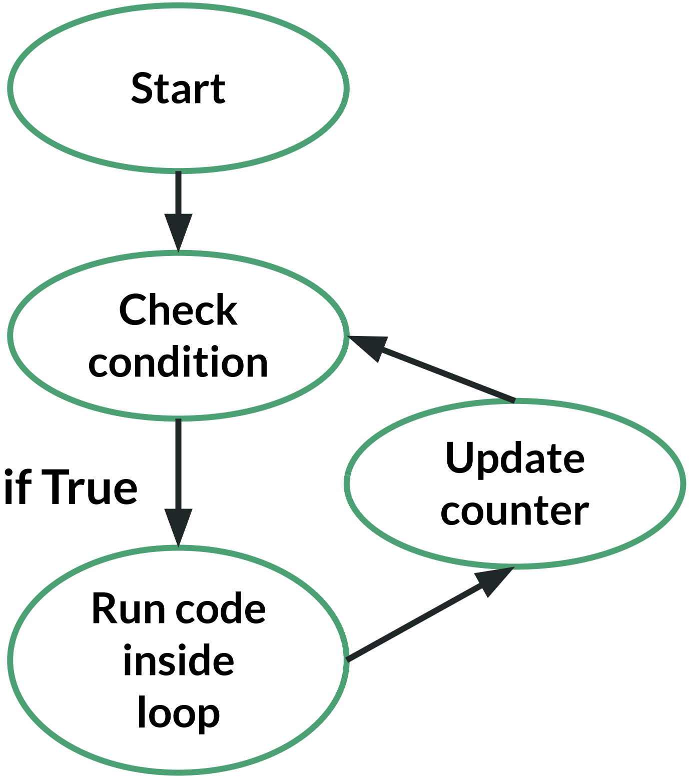
```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
```

output



for loop syntax

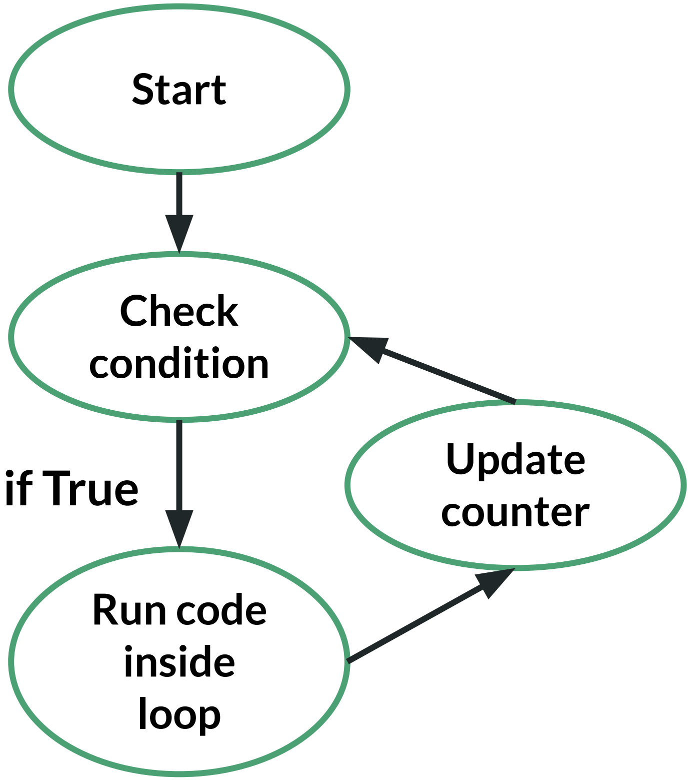
```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
2
```

output



`for` loop syntax

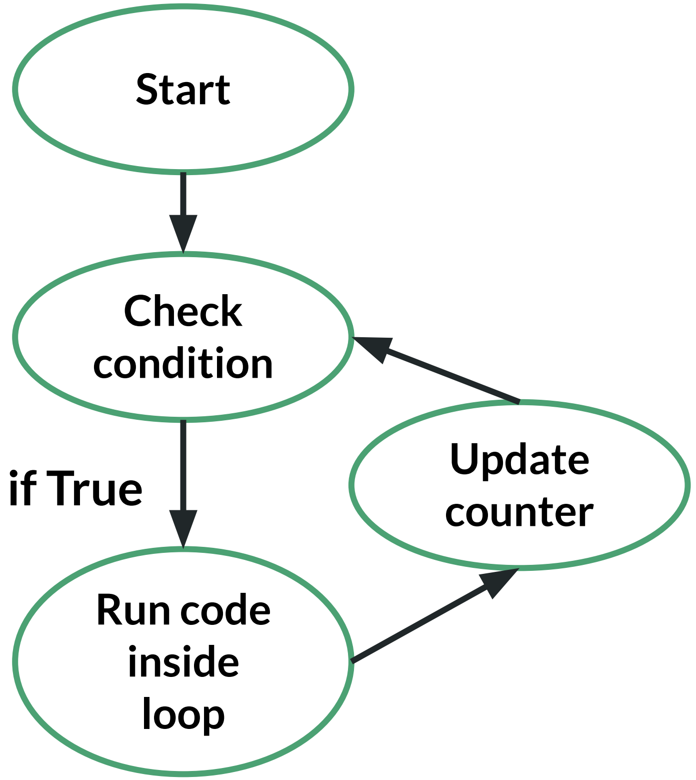
```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
2
3
```

output



`for` loop syntax

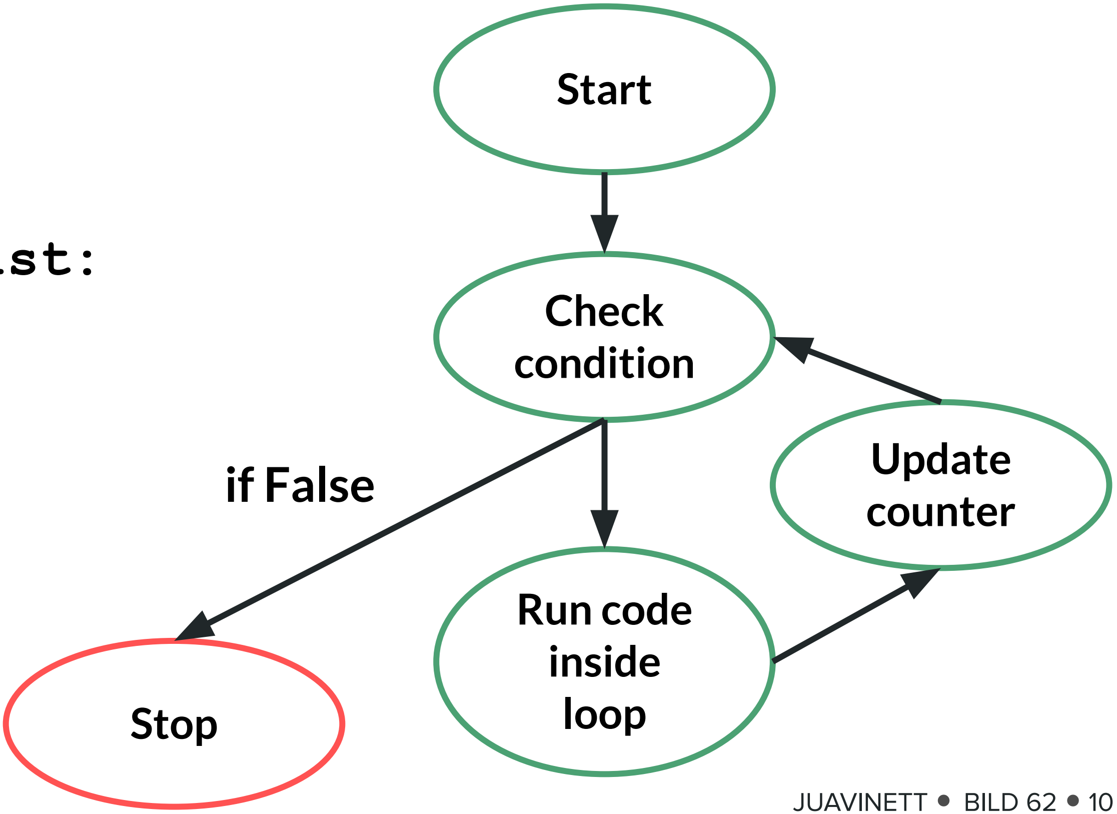
```
a_list = [1,2,3]

for value in a_list:

    print(value)
```

```
1
2
3
```

output



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.

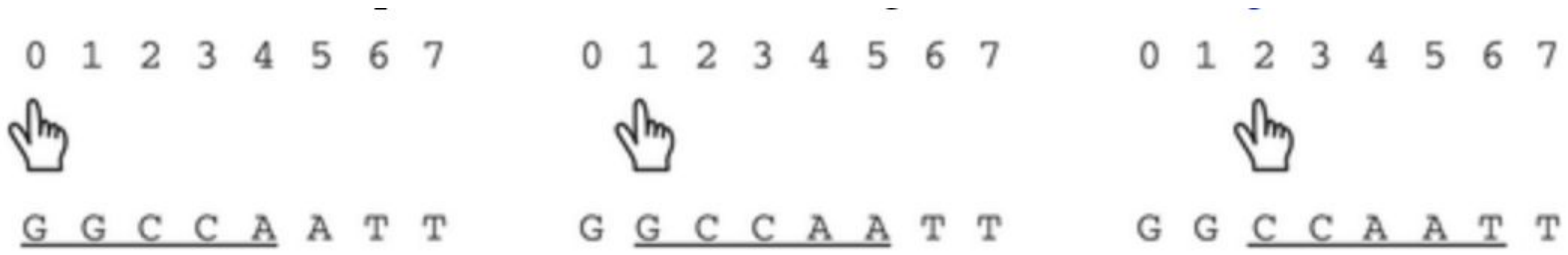
The "CAT" box generally appears near the spot where transcription begins!

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

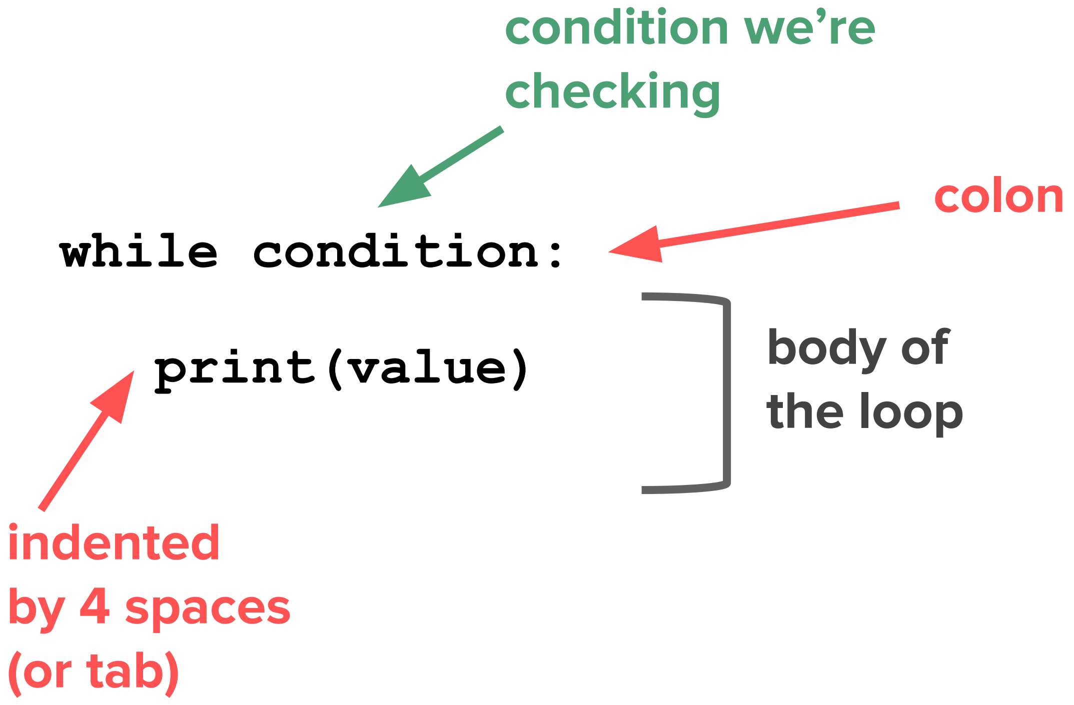
We can also loop over a list of indices!

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.



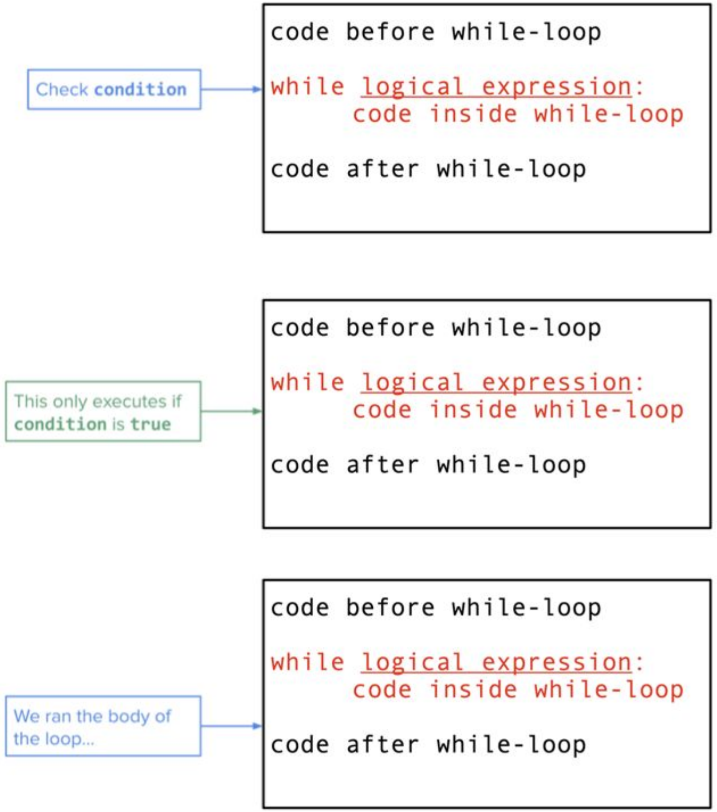
`while` loop syntax



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

While this condition is true, the loop will run!

It will repeat until the condition is no longer True.

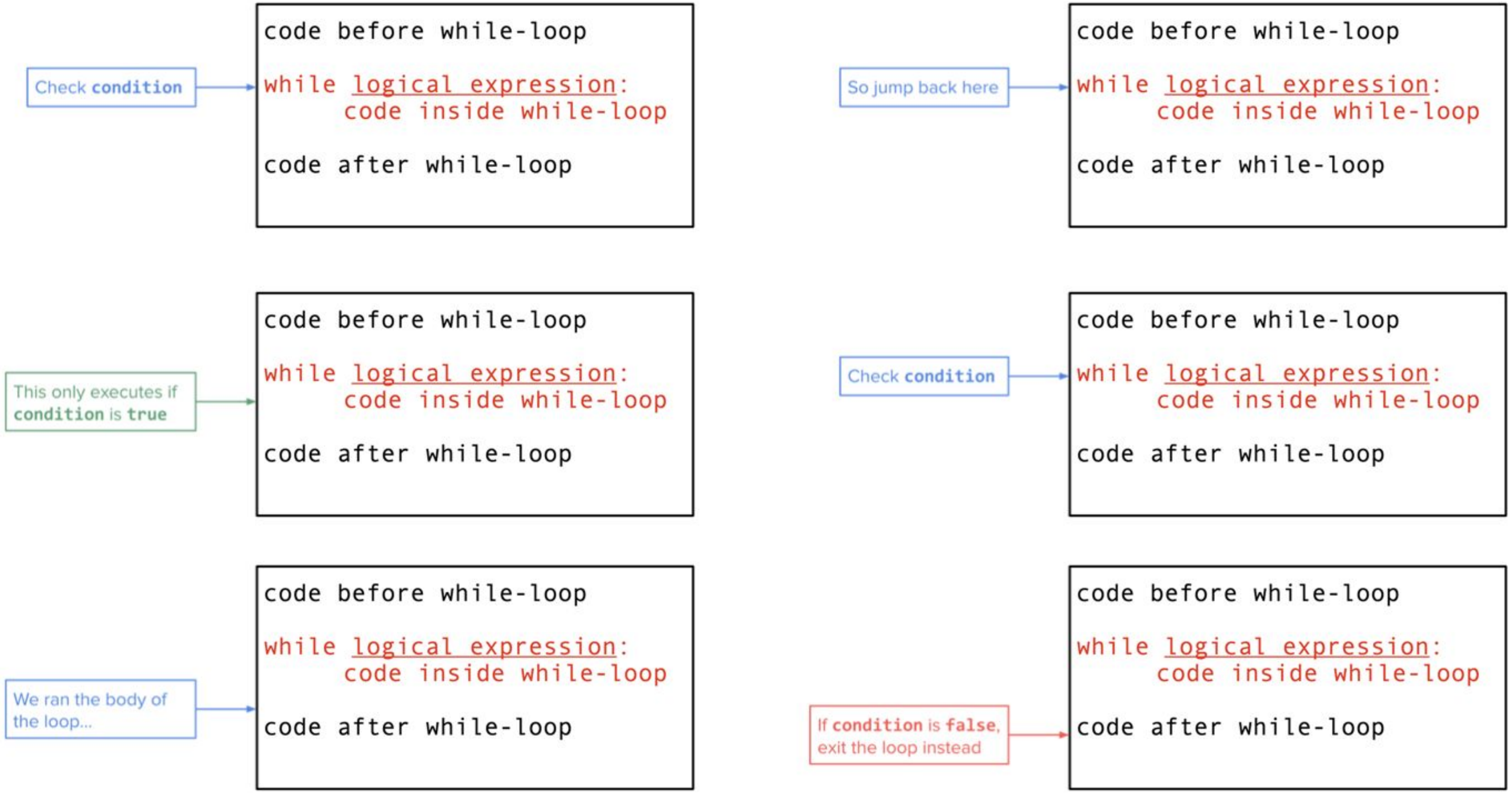

Check condition
This only executes if condition is true
We ran the body of the loop...

```
code before while-loop

while logical expression:
      code inside while-loop

code after while-loop
```

Order of execution in a while loop (from Stepik)


Check condition
So jump back here
This only executes if condition is true
Check condition
We ran the body of the loop...
If condition is false, exit the loop instead

```
code before while-loop

while logical expression:
        code inside while-loop

code after while-loop
```

Order of execution in a while loop (from Stepik)

Resources

Stepik Introduction to Python book, Chapter 3

Whirlwind Tour of Python: Control Flow