LOOPS
WHILE LOOPS

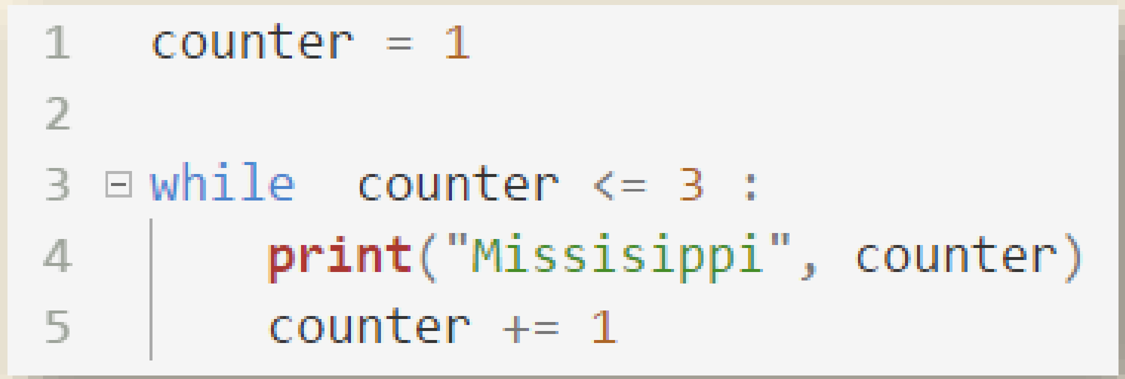
Slides By Björgvin B. Björgvinsson

WHILE LOOPS

- Computers are pretty fast and can do the same thing over and over again without getting tired of it.
- Us humans however get distracted pretty easily
  - For example: let's say that you‘re a teacher and you need to review 50 exams. You start out pretty focused but after a while you get distracted, you might start thinking about how much laundry is waiting for at home or just start thinking about the weather like icelandic people tend to do. So your work becomes sloppy and you get more irritated as you realize how much time this is going to take.
  - However, computers will happily do this kind of repetetive work! They will even do it much faster than we could ever do and they would do it without complaining.
  - The only thing we need to do is give the computer instructions on how to do the job and to do that we need to learn about **loops**!

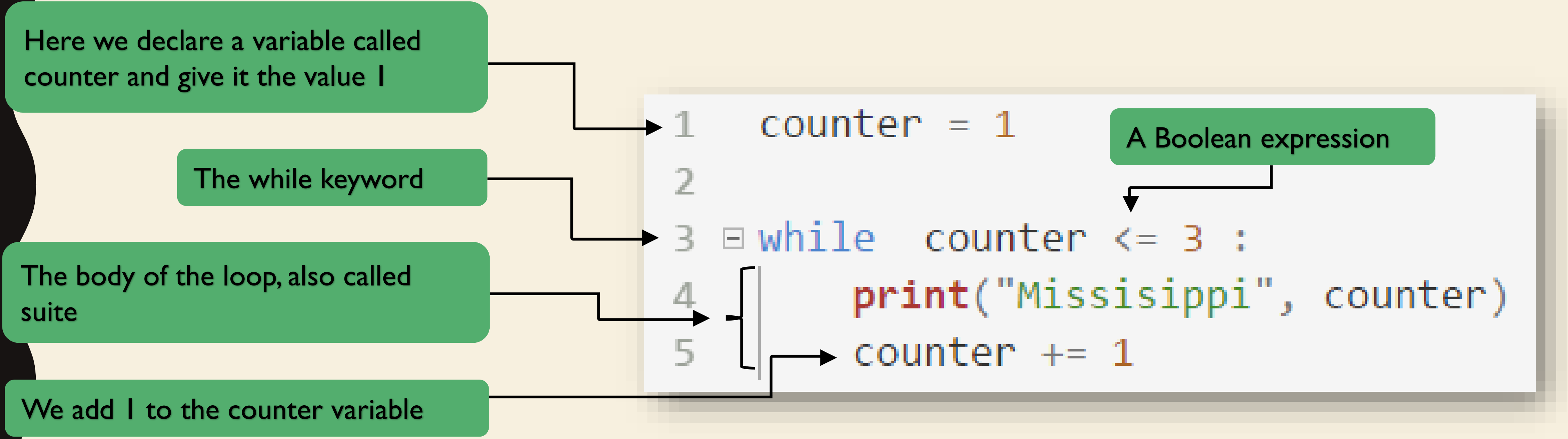
WHILE LOOPS

- A loop is a mechanism that let's us be able to repeat a set of operations multiple times.
- We already know **boolean expressions**, which comes in handy because loops make great use of them.
- A loop will repeat itself **while** a certain **boolean expression** evaluates as **True**
  - Once the **boolean expression** evaluates as **False** the loop will stop
- Here is an example of a **while loop**
- let's take a look at this loop in more detail on the next slide



```python
counter = 1

while  counter <= 3 :
    print("Missisippi", counter)
    counter += 1
```

WHILE LOOPS



Here we declare a variable called counter and give it the value 1
The while keyword
The body of the loop, also called suite
We add 1 to the counter variable
A Boolean expression

```
counter = 1

while  counter <= 3 :
    print("Missisippi", counter)
    counter += 1
```

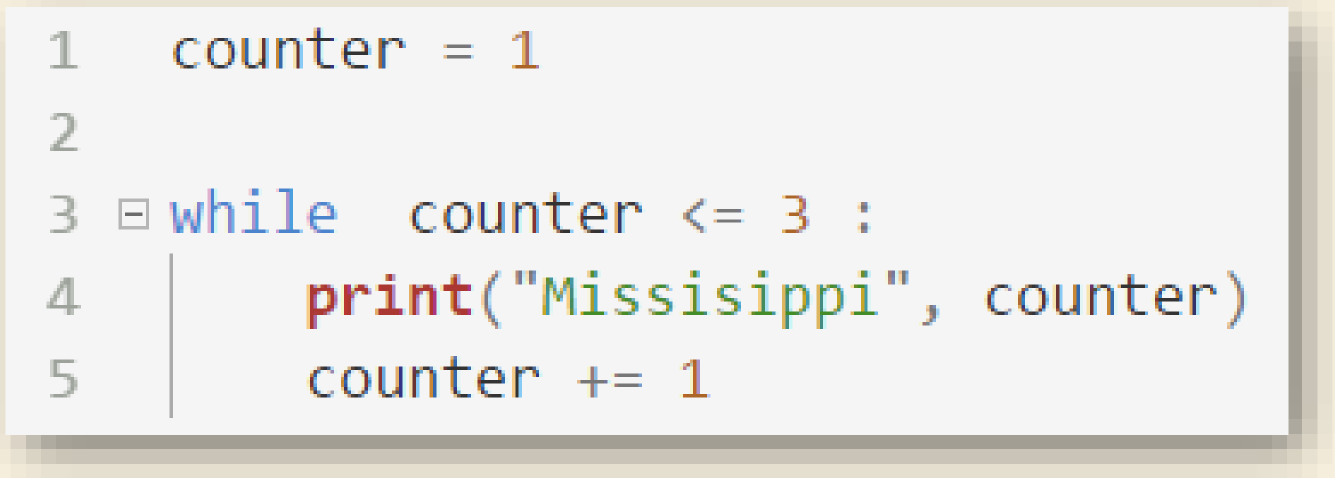
WHILE LOOPS

- The body of the loop will run 3 times, let's go to through the iterations

1. On line 3 the Boolean expression evaluates to True, Mississippi 1 is printed and 1 is added to the counter variable.

2. The execution jumps back to line 3 and evaluates the expression again (now counter has the value 2) as True, Mississippi 2 is printed and 1 is added to the counter variable.

3. The execution jumps back to line 3 and evaluates the expression again (now counter has the value 3) as True, Mississippi 3 is printed and 1 is added to the counter variable.

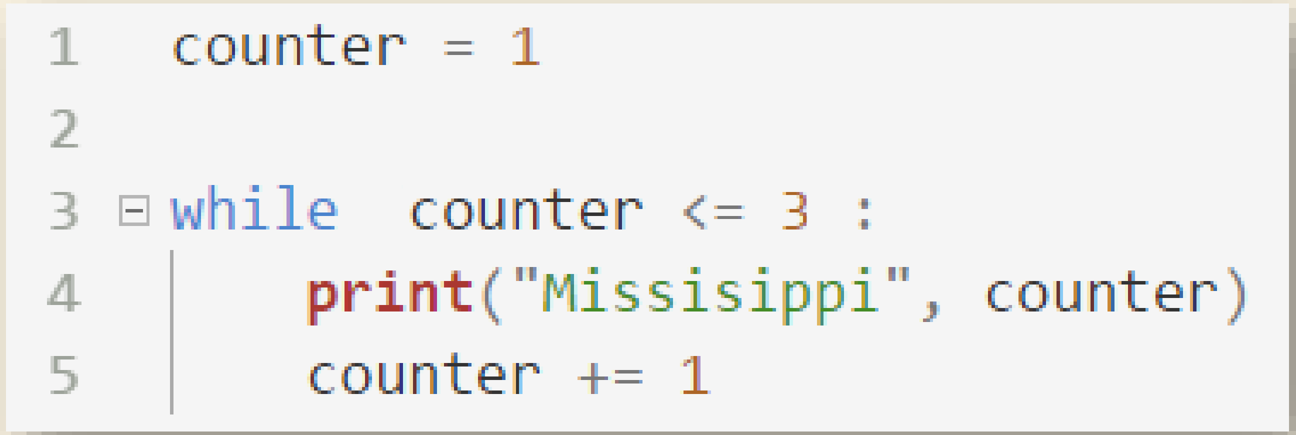


```python
1  counter = 1
2
3  while  counter <= 3 :
4      print("Missisippi", counter)
5      counter += 1
```

4. The execution jumps back to line 3 and evaluates the expression again (now counter has the value 4) as False! Now the Boolean expression is false and the execution jumps/breaks out of the loop.

WHILE LOOPS

- Pay special attention to the **indentation** within the while loop
- Also notice that the while statement is followed by a colon just like the if and else statements



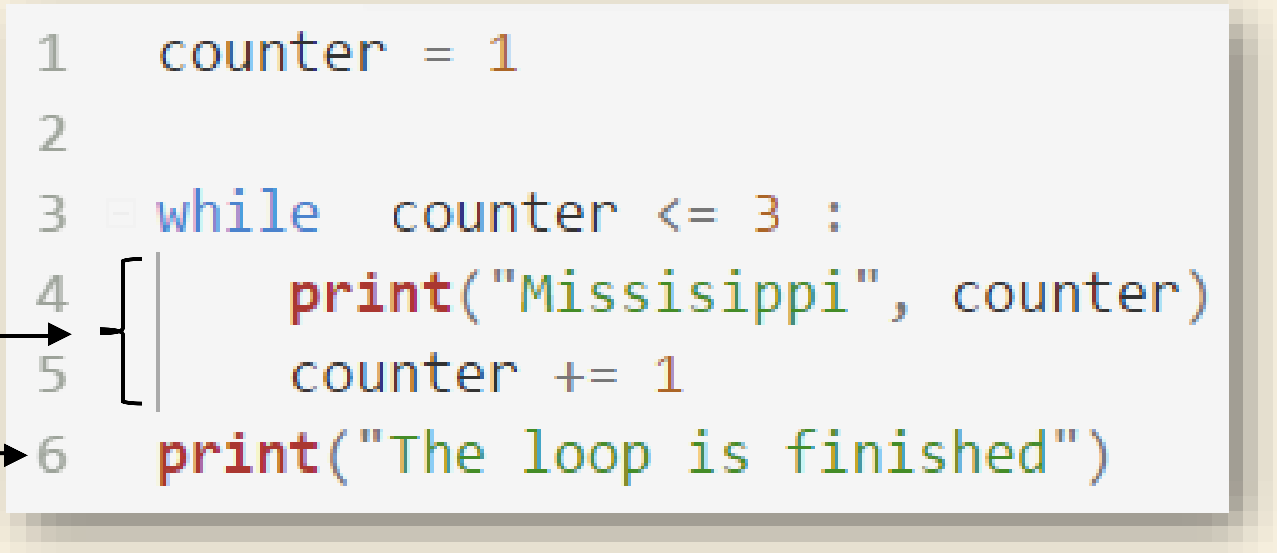
```
counter = 1

while  counter <= 3 :
    print("Missisippi", counter)
    counter += 1
```

WHILE LOOPS

- Here is another example to emphasize the importance of indentation

The indentation of these two lines tells us that they are a part of the loop’s body

This line is not part of the loops body. It will only execute once the loop has finished



```python
counter = 1

while  counter <= 3 :
    print("Missisippi", counter)
    counter += 1
print("The loop is finished")
```

WHILE LOOPS ANALYZED

- The first thing that happens in a while loop is the evaluation of the boolean expression
- If the boolean expression evaluates as **True** the body of the loop will be executed
- If the boolean expression evaluates as **False** the body of the loop won‘t be executed and the execution jumps over the loops body and continues

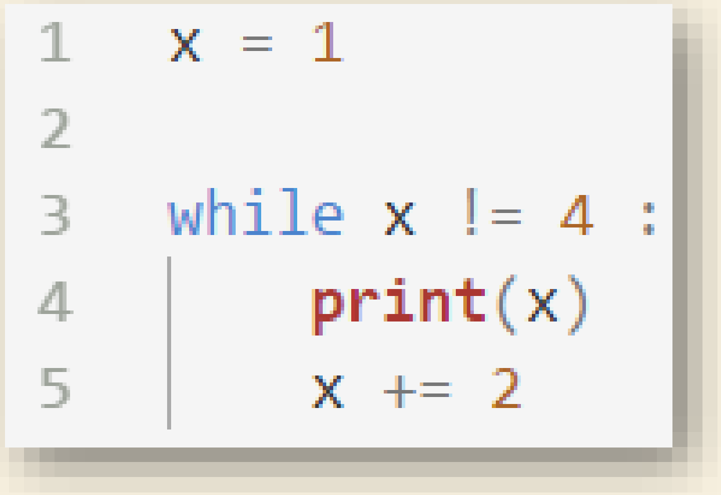
WHILE LOOPS

A very common part of loops is having a variable that acts like a **counter**. That counter is most often **incremented** or **decremented** by 1. But it is worth noting that it can be incremented or decremented by any number.

INFINITE LOOPS

A COMMON MISTAKE...

- A common mistake beginner programmers tend to do is to create infinite loops.
- When you create an infinite loop, the body of the loop will execute repeatedly until the program crashes.
- This code is an example of an infinite loop.



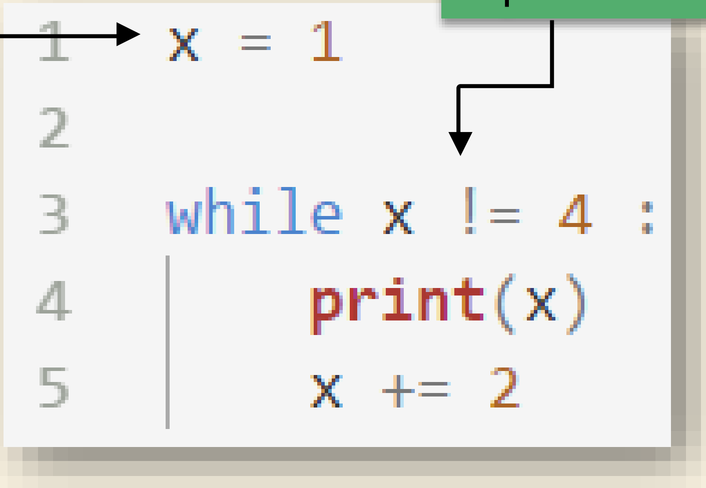
```
x = 1

while x != 4 :
    print(x)
    x += 2
```

INFINITE LOOPS

A COMMON MISTAKE...

x is initialized with the value 1

The Boolean expression



```
x = 1

while x != 4 :
    print(x)
    x += 2
```

1 The value of x is 1.The Boolean expression check x(1) != 4 which is true and the loop body is executed. 1 is printed and 2 is added to x.

2 The value of x is 3.The Boolean expression check x(3) != 4 which is true and the loop body is executed. 3 is printed and 2 is added to x.

3 The value of x is 5.The Boolean expression check x(5) != 4 which is true and the loop body is executed. 5 is printed and 2 is added to x.

4 The value of x is 7.We've already exceeded 4 but the loops is still running. Now it's impossible for the Boolean expression to be false. So the loop will continue iterating until the program crashes.