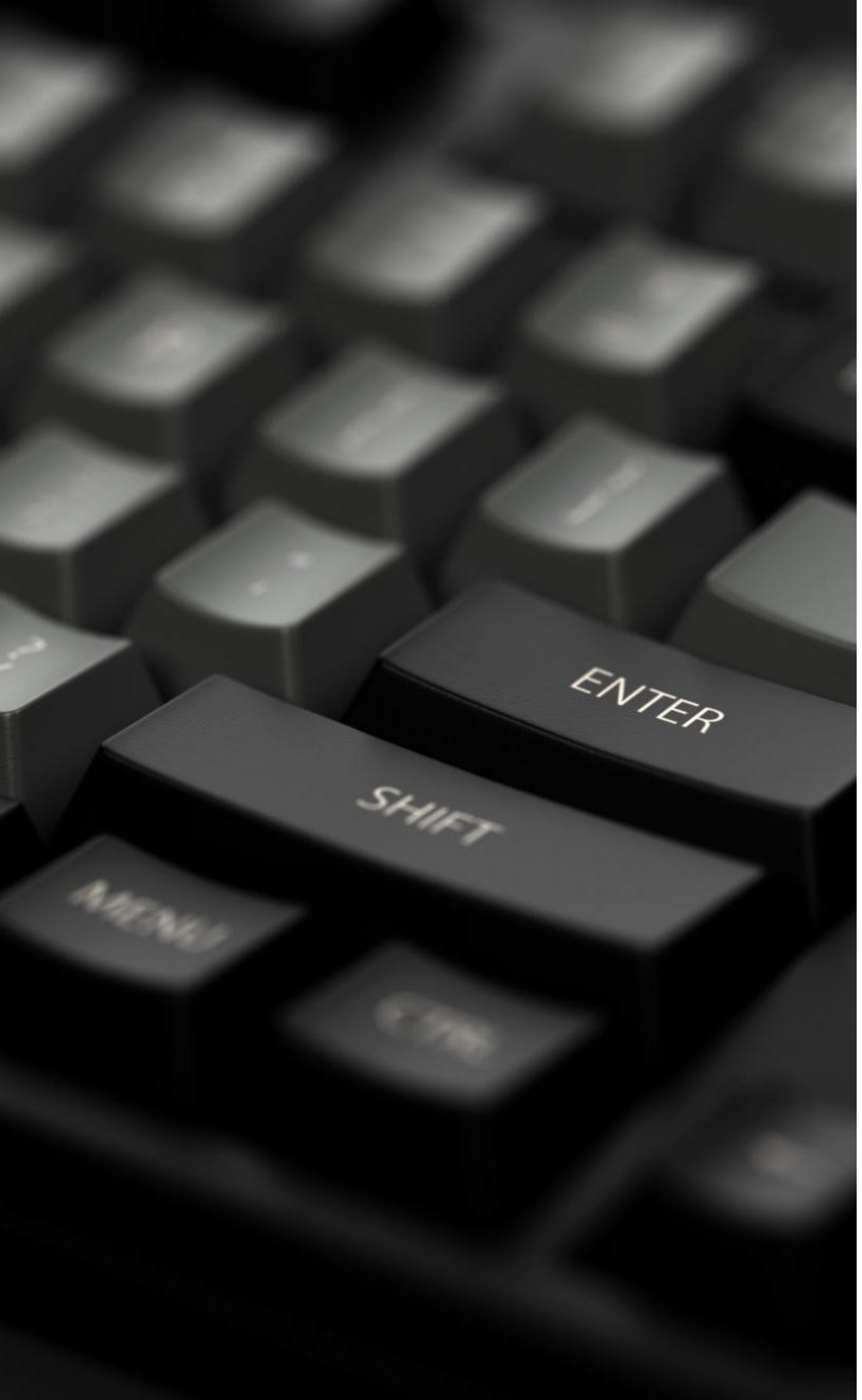
A gamified introduction to Python Programming

Lecture 5
Looping with while statements

Matthieu DE MARI – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



Outline (Chapter 5)

- What is the **while** statement?
- What are **infinite loops**, why are they problematic and **how to stop them**?
- What is the **break** statement and how to use it in while statements?
- What are **good practices** when it comes to using while loops?
- (If time allows, let us discuss recursion!)

The while statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

The while statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**

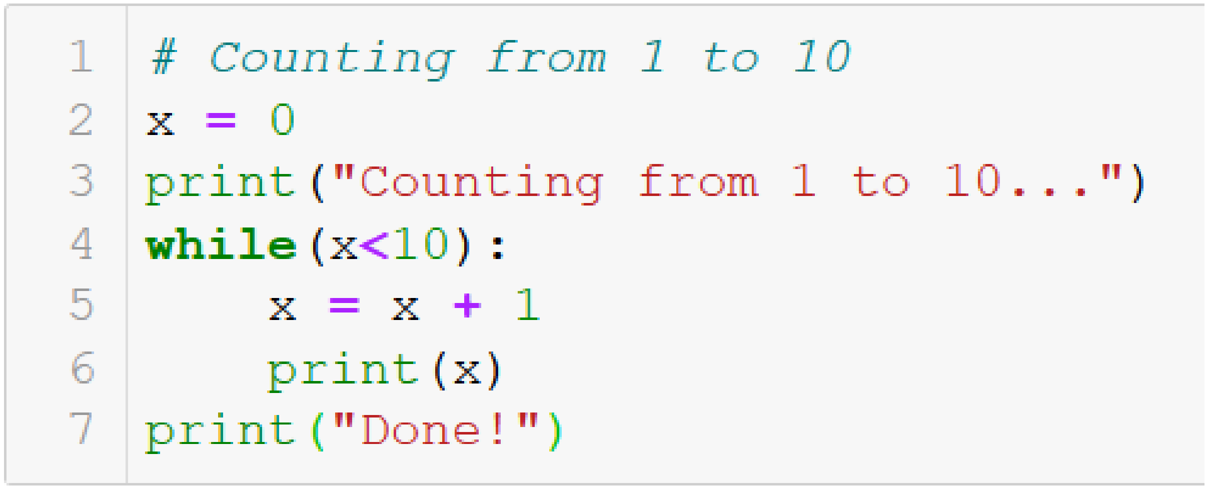
The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), **move on to the next (non-indented) line.**

The while statement

The **while** statement is another type of **conditional structure.**

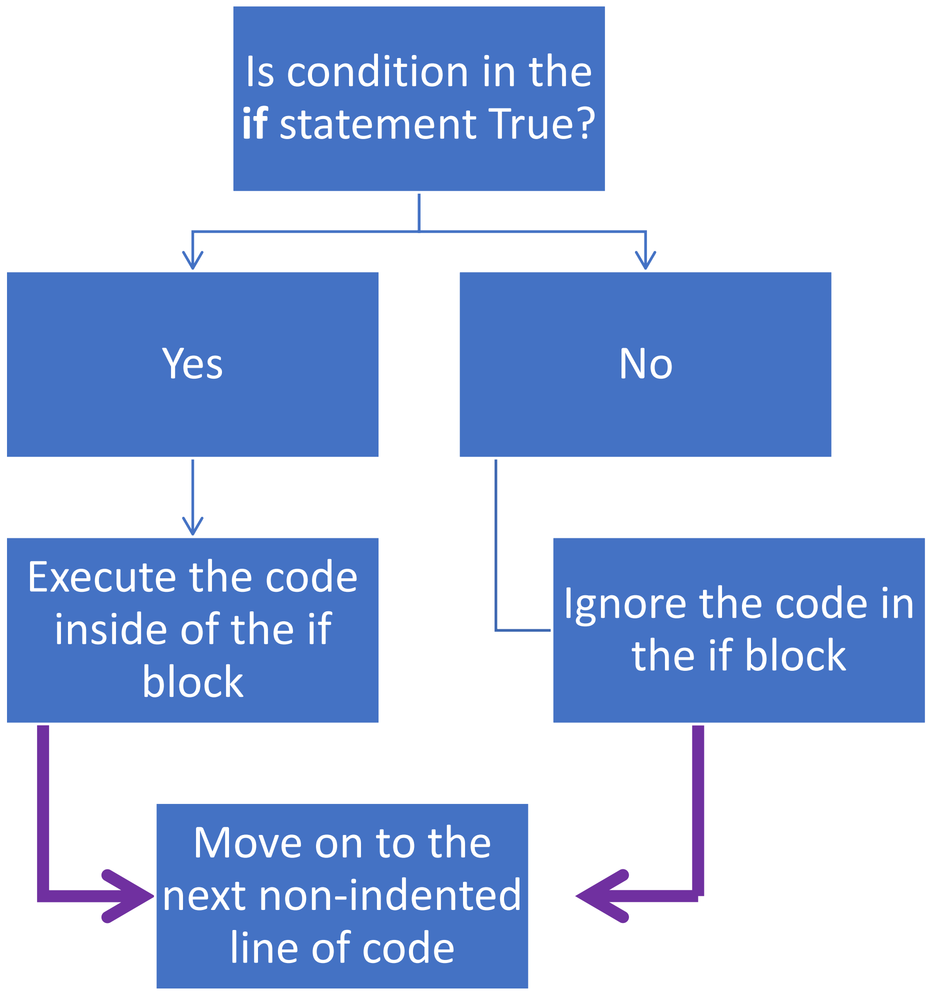
- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**

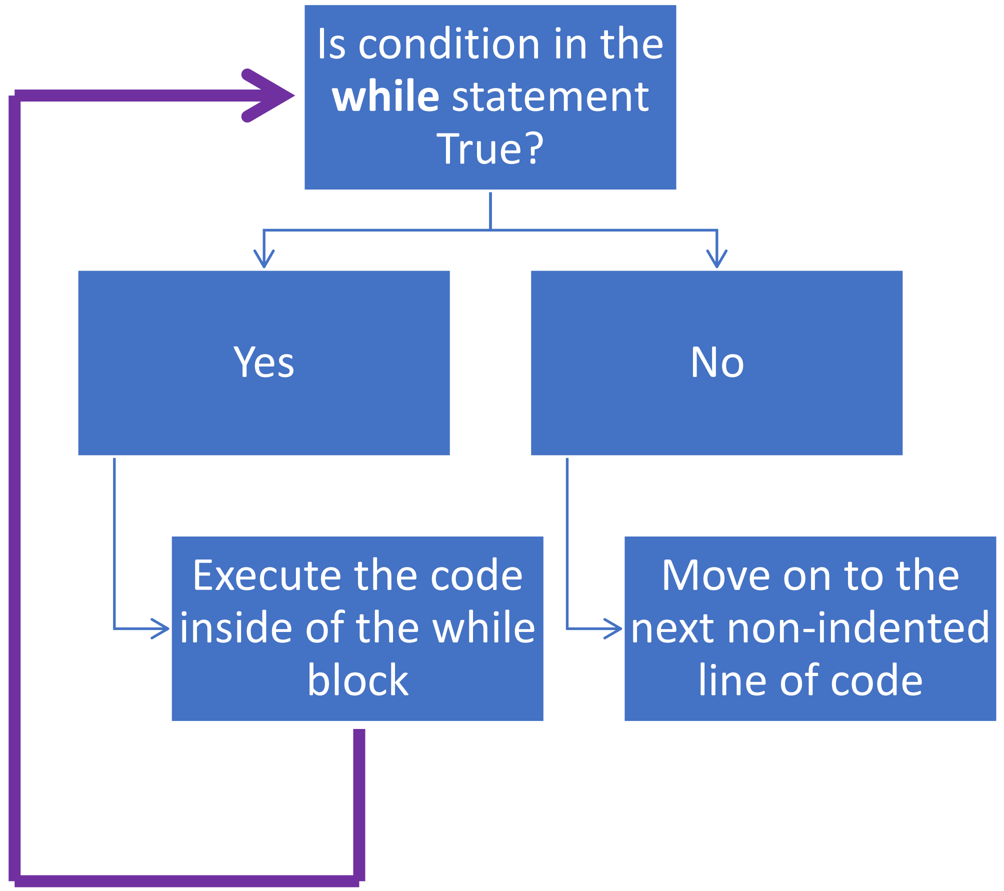


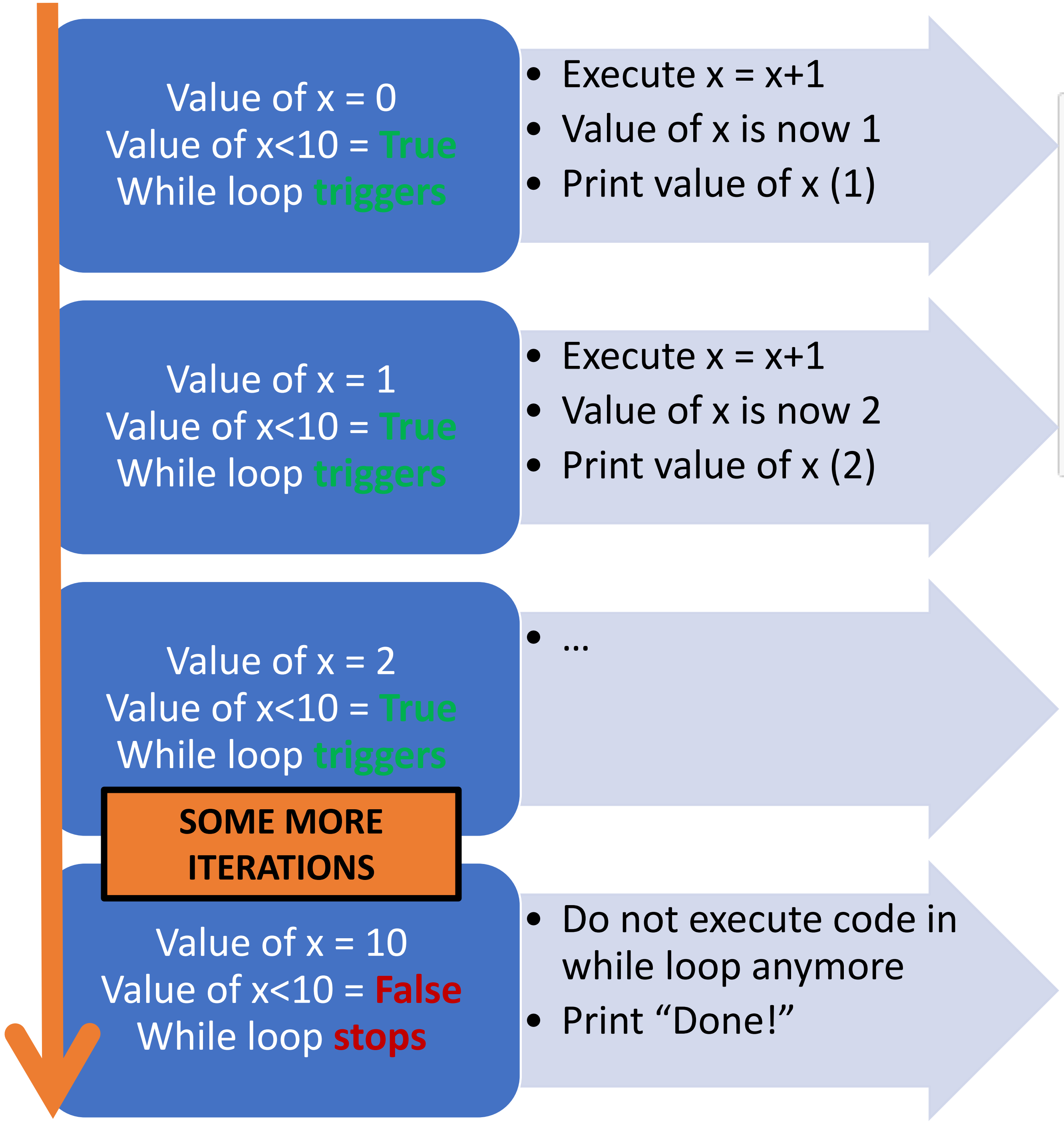
```python
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

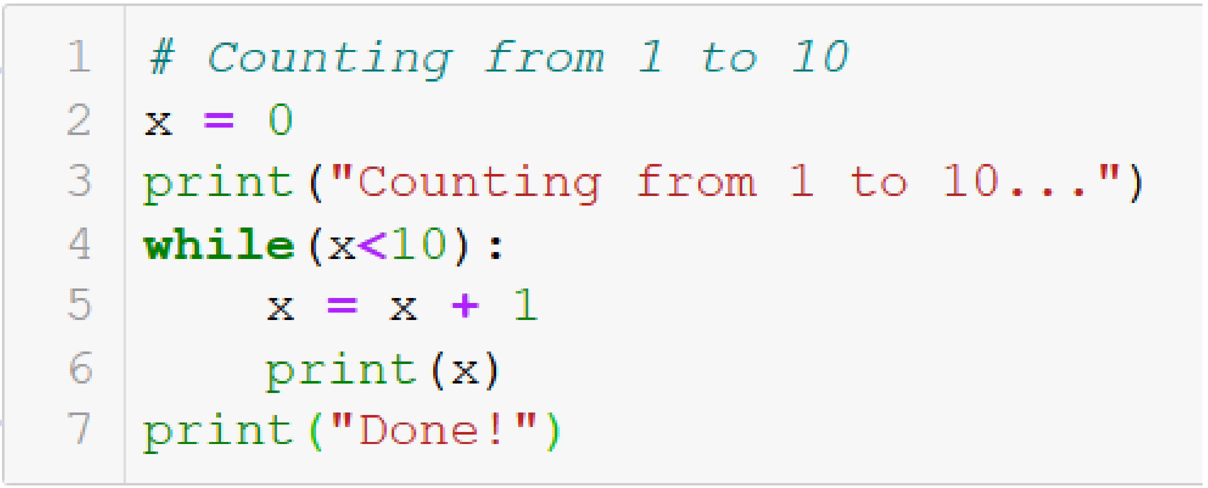
```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

Architectures: if vs. while


Is condition in the **if** statement True?
Yes
No
Execute the code inside of the if block
Ignore the code in the if block
Move on to the next non-indented line of code


Is condition in the **while** statement True?
Yes
No
Execute the code inside of the while block
Move on to the next non-indented line of code





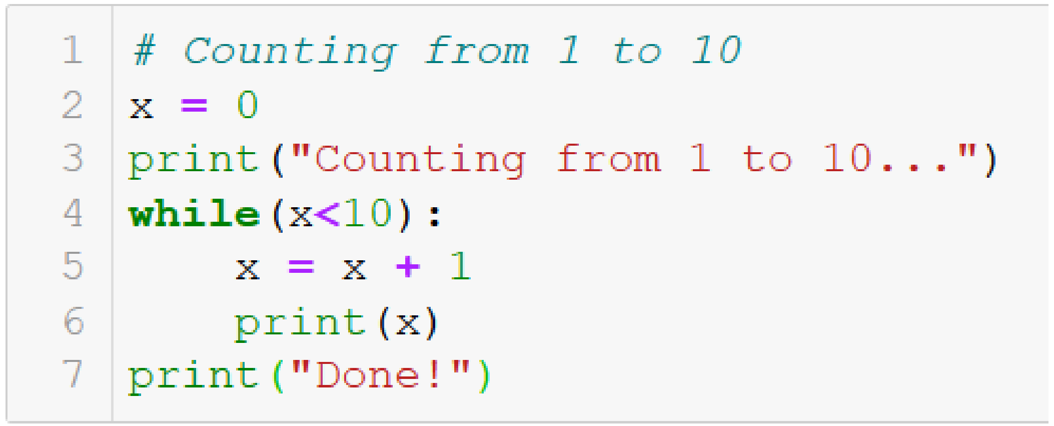
```python
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

Infinite loops

The **while** statement repeats a condition until it is no longer **True**.

This means that there should be a clear process that **makes your condition no longer True**, at some point.



```
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

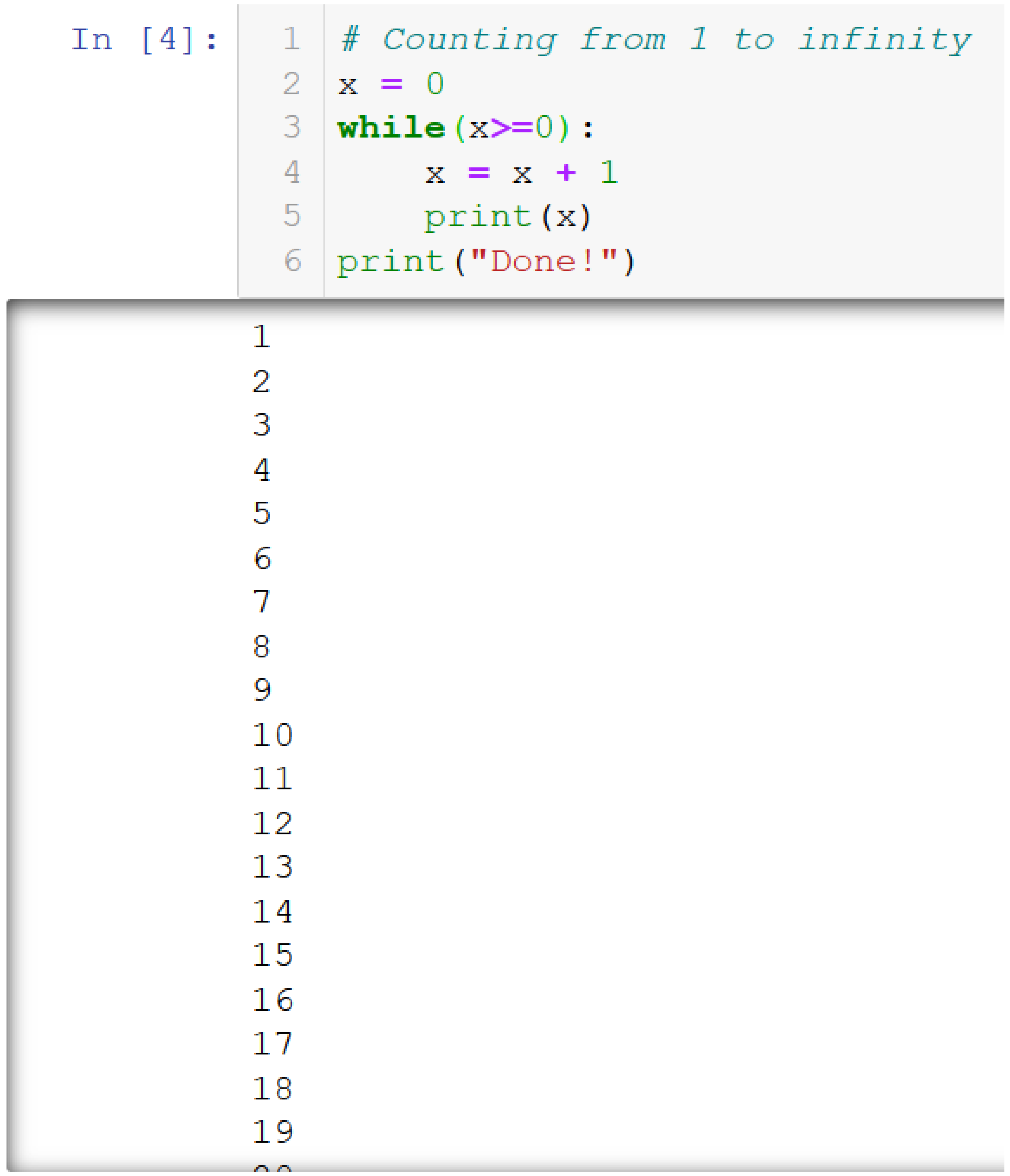
Infinite loops

The **while** statement repeats a condition until it is no longer **True**.

This means that there should be a clear process that **makes your condition no longer True**, at some point.

Otherwise, big problem!

- The **while** block will keep on repeating indefinitely…
  This is called an **infinite loop**.



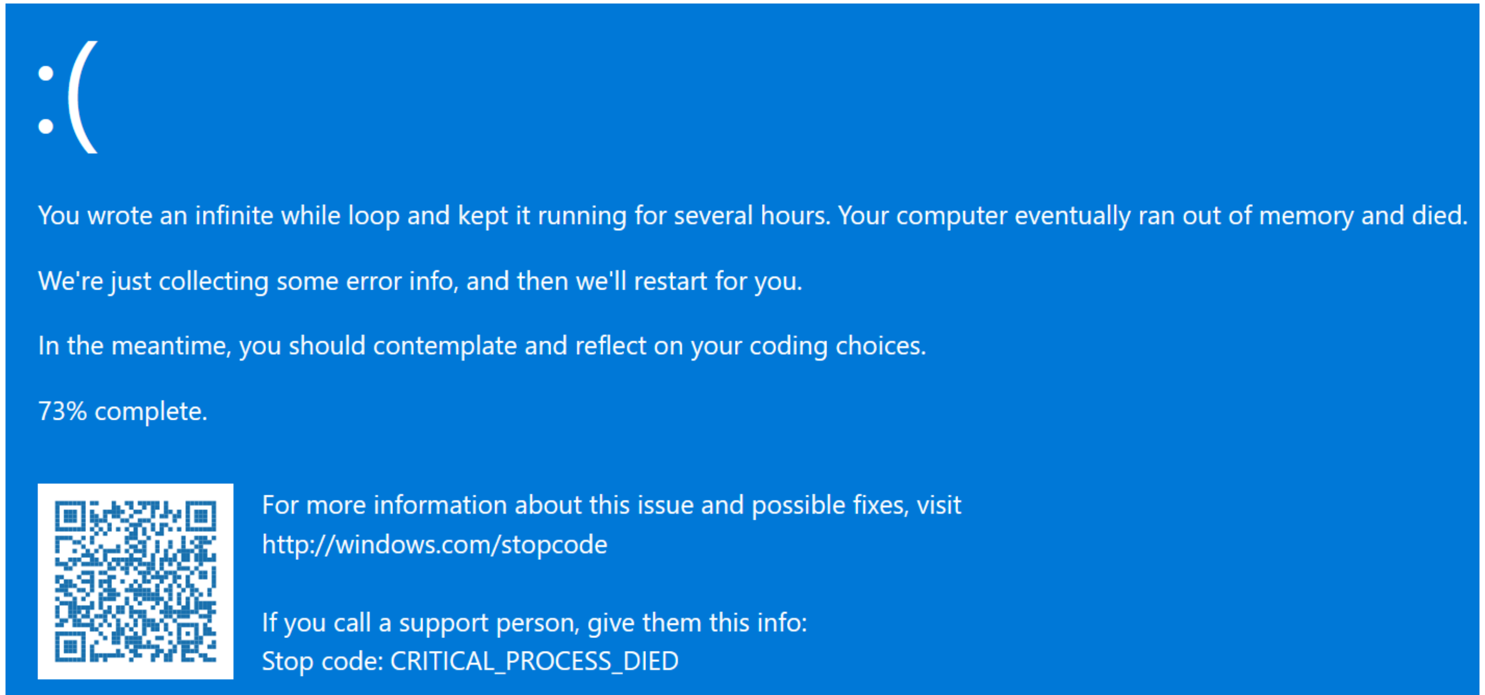
```
In [4]:
# Counting from 1 to infinity
x = 0
while(x>=0):
    x = x + 1
    print(x)
print("Done!")
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. ~~Your computer runs out of resources/memory and performs an emergency shutdown before exploding (**bad thing to do**),~~



:(

You wrote an infinite while loop and kept it running for several hours. Your computer eventually ran out of memory and died.

We're just collecting some error info, and then we'll restart for you.

In the meantime, you should contemplate and reflect on your coding choices.

73% complete.

For more information about this issue and possible fixes, visit http://windows.com/stopcode

If you call a support person, give them this info:
Stop code: CRITICAL_PROCESS_DIED

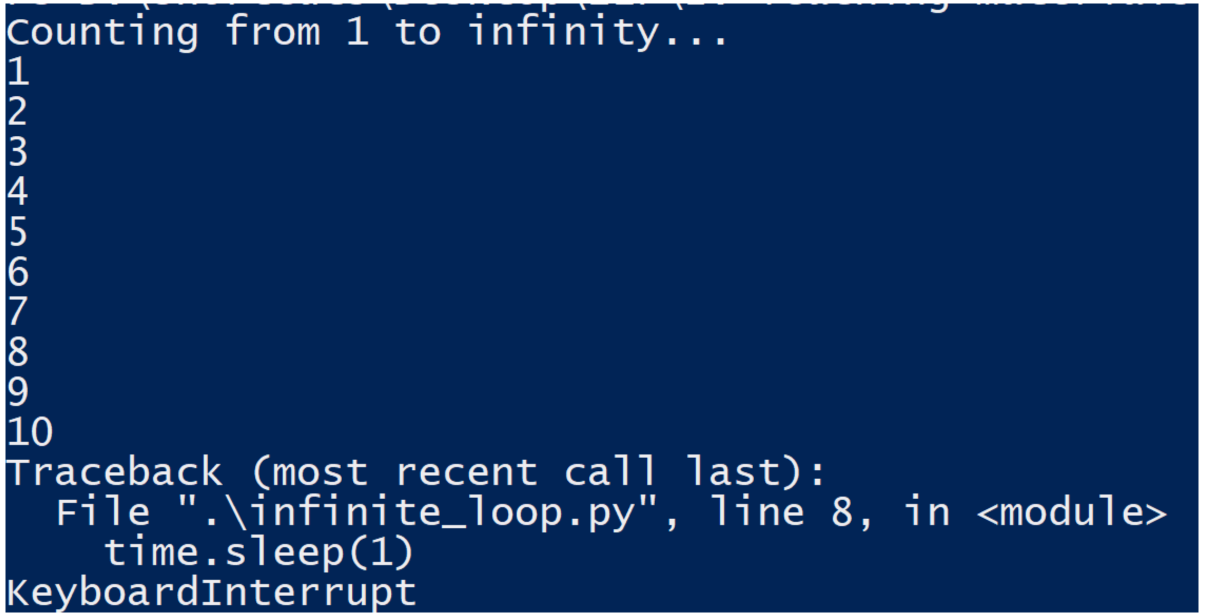
Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

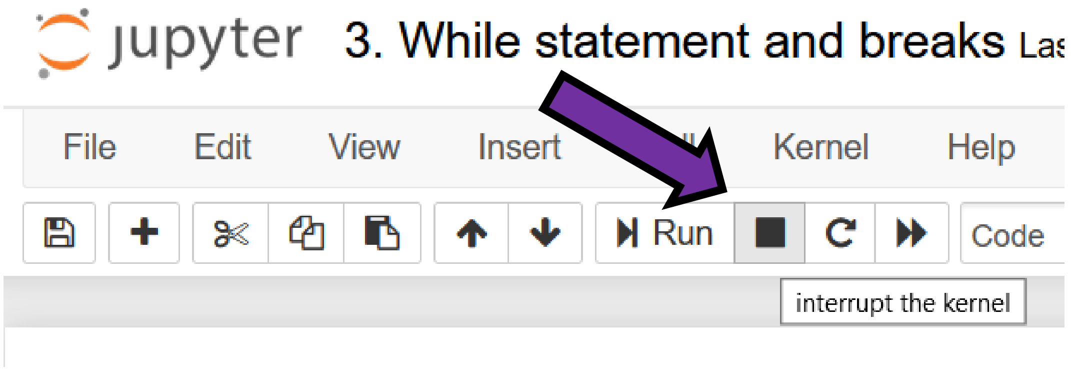
1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done by pressing (repeatedly) **CTRL+C** (or **CMD+C** on mac), in console mode and most IDEs.

Or, by using the **stop button** on Jupyter.



```
Counting from 1 to infinity...
1
2
3
4
5
6
7
8
9
10
Traceback (most recent call last):
  File ".\infinite_loop.py", line 8, in <module>
    time.sleep(1)
KeyboardInterrupt
```



Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done by pressing (repeatedly) **CTRL+C** (or **CMD+C** on mac), in console mode and most IDEs.

Or, by using the **stop button** on Jupyter.

How it feels when you perform a keyword interrupt on your poor Python

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

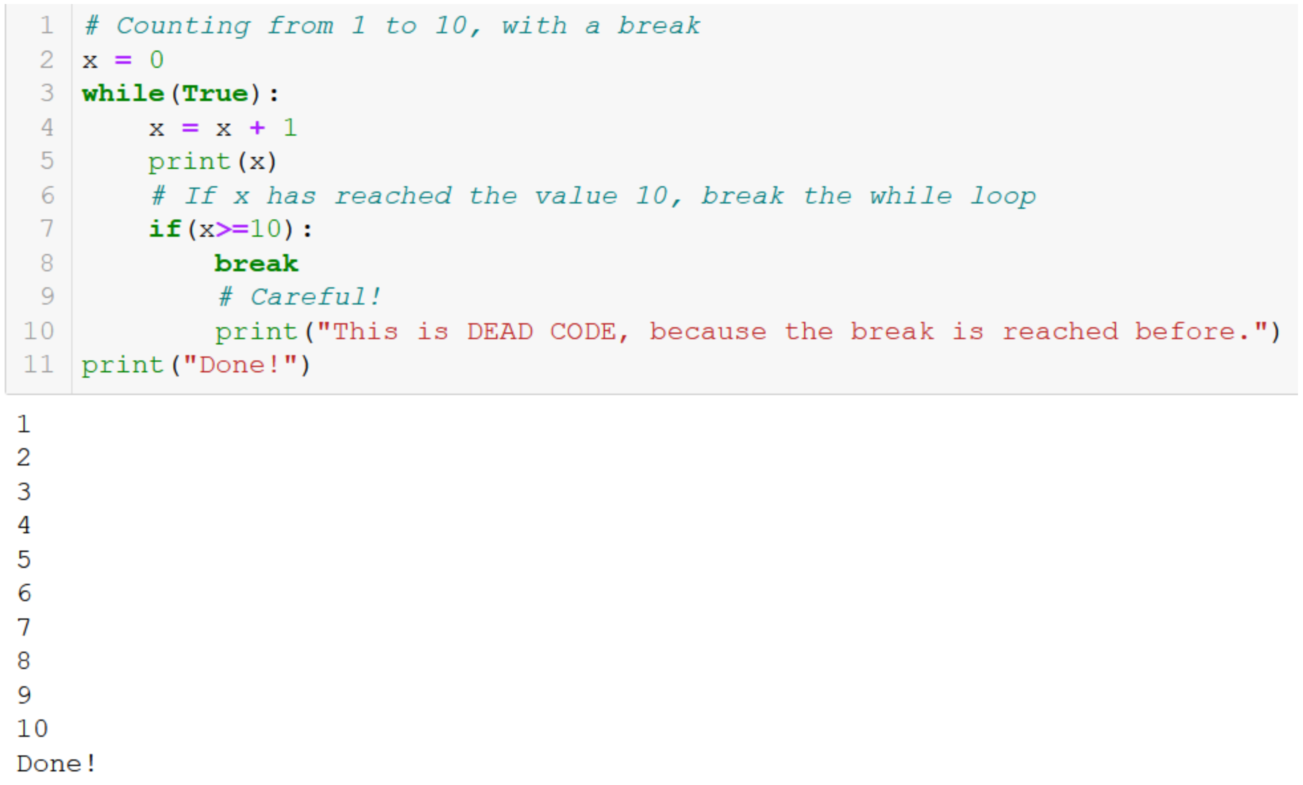
Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

The code then resumes its execution with the next line outside of the **while** block.

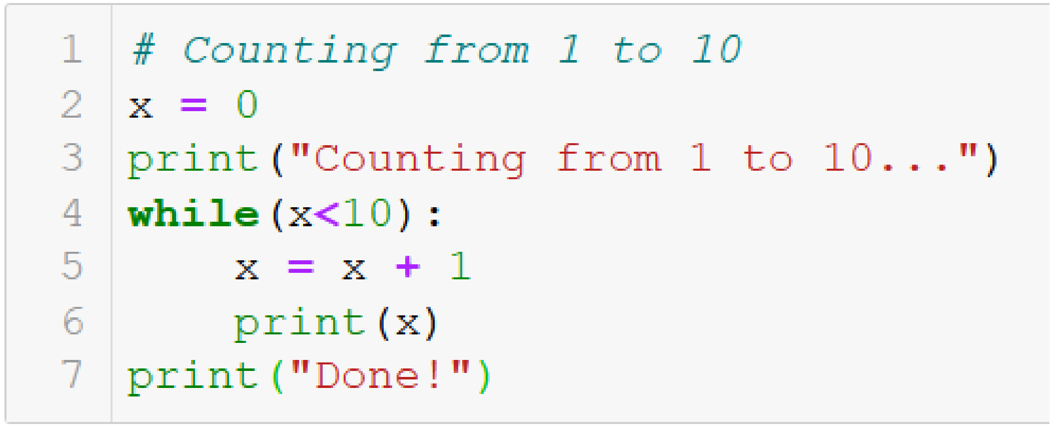


```python
# Counting from 1 to 10, with a break
x = 0
while(True):
    x = x + 1
    print(x)
    # If x has reached the value 10, break the while loop
    if(x>=10):
        break
        # Careful!
        print("This is DEAD CODE, because the break is reached before.")
print("Done!")
```

```
1
2
3
4
5
6
7
8
9
10
Done!
```

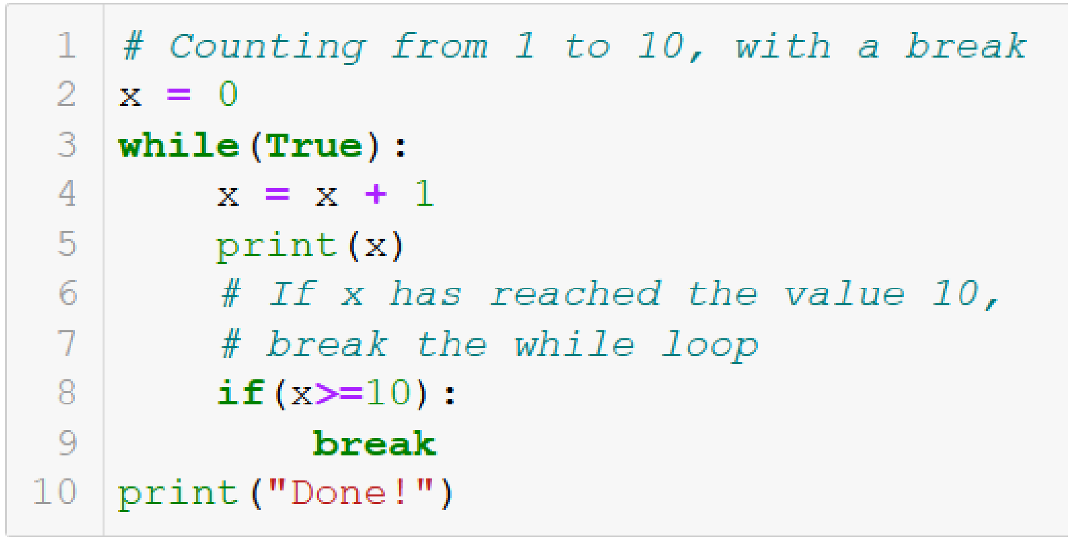
Standard while vs. infinite while + break

1. Standard **while** loop with condition in the while statement.



```python
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

2. Infinite **while** loop with condition in an **if** statement, and **break** in the **if** block.



```python
# Counting from 1 to 10, with a break
x = 0
while(True):
    x = x + 1
    print(x)
    # If x has reached the value 10,
    # break the while loop
    if(x>=10):
        break
print("Done!")
```

→ Both loops work and do the job, which one is better though?

Matt’s Great advice

Matt’s Great Advice: Avoid the infinite loops, if possible (part 2).

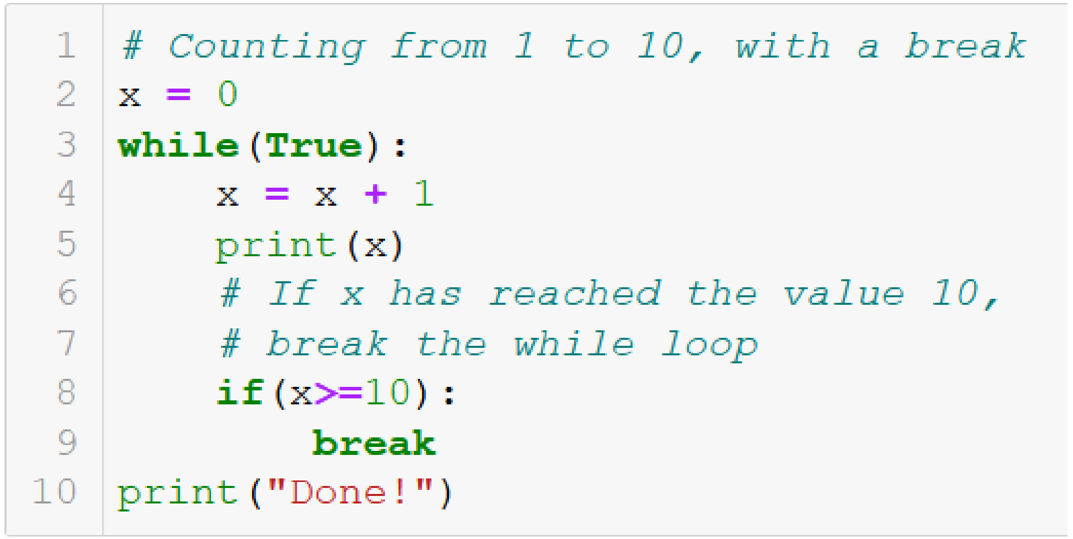
Relying on an **infinite** **while** loop with a **break** is <u>**risky**</u> and should be avoided when possible.

Matt’s Great advice

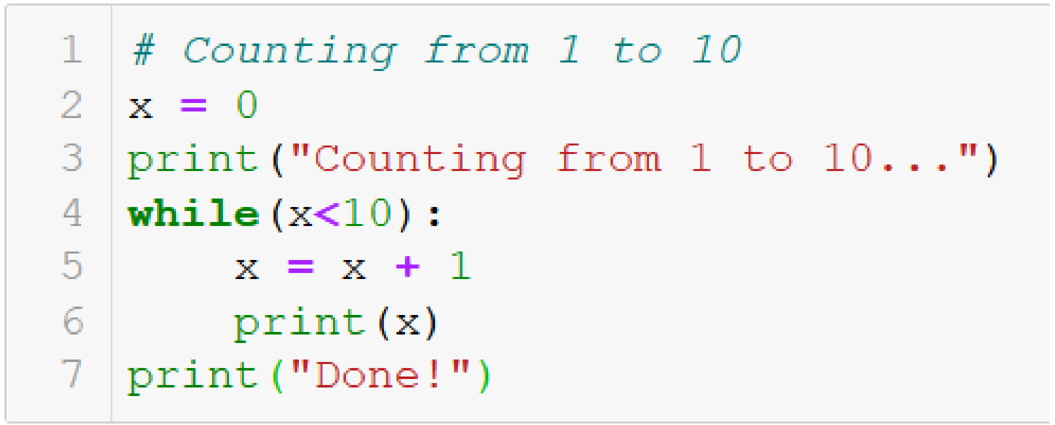
Matt’s Great Advice: Avoid the infinite loops, if possible (part 2).

Relying on an **infinite** **while** loop with a **break** is **<u>risky</u>** and should be avoided when possible.

It is often easily avoided, by using the Boolean expression of the **if** statement used for **break**, as the condition in the **while** statement.



```python
# Counting from 1 to 10, with a break
x = 0
while(True):
    x = x + 1
    print(x)
    # If x has reached the value 10,
    # break the while loop
    if(x>=10):
        break
print("Done!")
```



```python
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

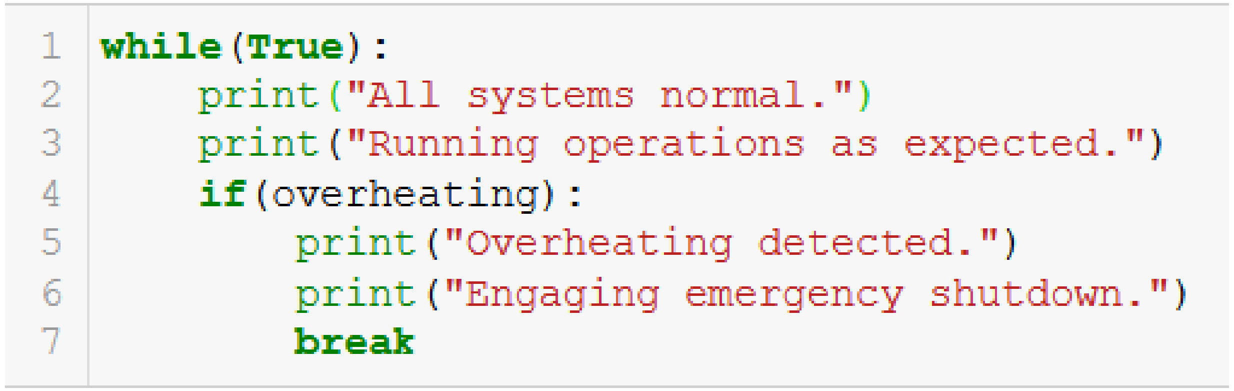
Matt’s Great advice

Matt’s Great Advice: Avoid the infinite loops, if possible (part 2).

Relying on an **infinite while** loop with a **break** is **risky** and should be avoided when possible.

It is often easily avoided, by using the Boolean expression of the if statement used for **break**, as the condition in the **while** statement.

Note: A few cases, however, require the use of a **break** statement.
For instance, **emergency shutdowns**.



```
while(True):
    print("All systems normal.")
    print("Running operations as expected.")
    if(overheating):
        print("Overheating detected.")
        print("Engaging emergency shutdown.")
        break
```

Some final advice before we move to practice

- **Understand the loop's purpose:** Before writing a loop, understand what you're trying to achieve. Is it iterating over a sequence until a condition is met, or repeating an action a certain number of times?
- **Write a simple pseudocode on paper:** Outline the logic of your loop in plain language or pseudocode. You <u>NEED</u> to clarify your thoughts before jumping into actual coding. Remember, this is step 1 for the life of a programmer! How are you supposed to translate into code a logic that you have not yet clarified begin with?
- **Set a clear stopping mechanism**: Make sure the loop has a well-defined condition that will eventually turn false, preventing infinite loops. Ask yourself, "Under what condition will this loop stop?".

Some final advice before we move to practice

- **Start simple and refactor:** Do not try to figure the entire code at once and hope you will land the correct solution in one shot. It is often better to start simple, and then refactor/improve the loop, progressively adding more operations, in a controlled manner.
- **Use and abuse the print:** Display the values of key variables at each iteration, to understand how your loop progresses and identify issues. You can always comment them/remove them later.
- **Review and simplify:** After getting your loop to work, revisit the code to see if it can be improved or be made more efficient. As you become more comfortable with loops, you will start noticing common loop patterns that can be reused in other problems.

Matt’s Great advice

Matt’s Great Advice: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose (!), usually follow from a **poor design** in your code.

Drawing a **structural diagram**, **<u>before coding</u>**, greatly helps figuring out the right structure for your code.

Use prints in your loops to see them in action and confirm their behavior!

Use pen and paper as well if it helps!

Matt’s Great advice

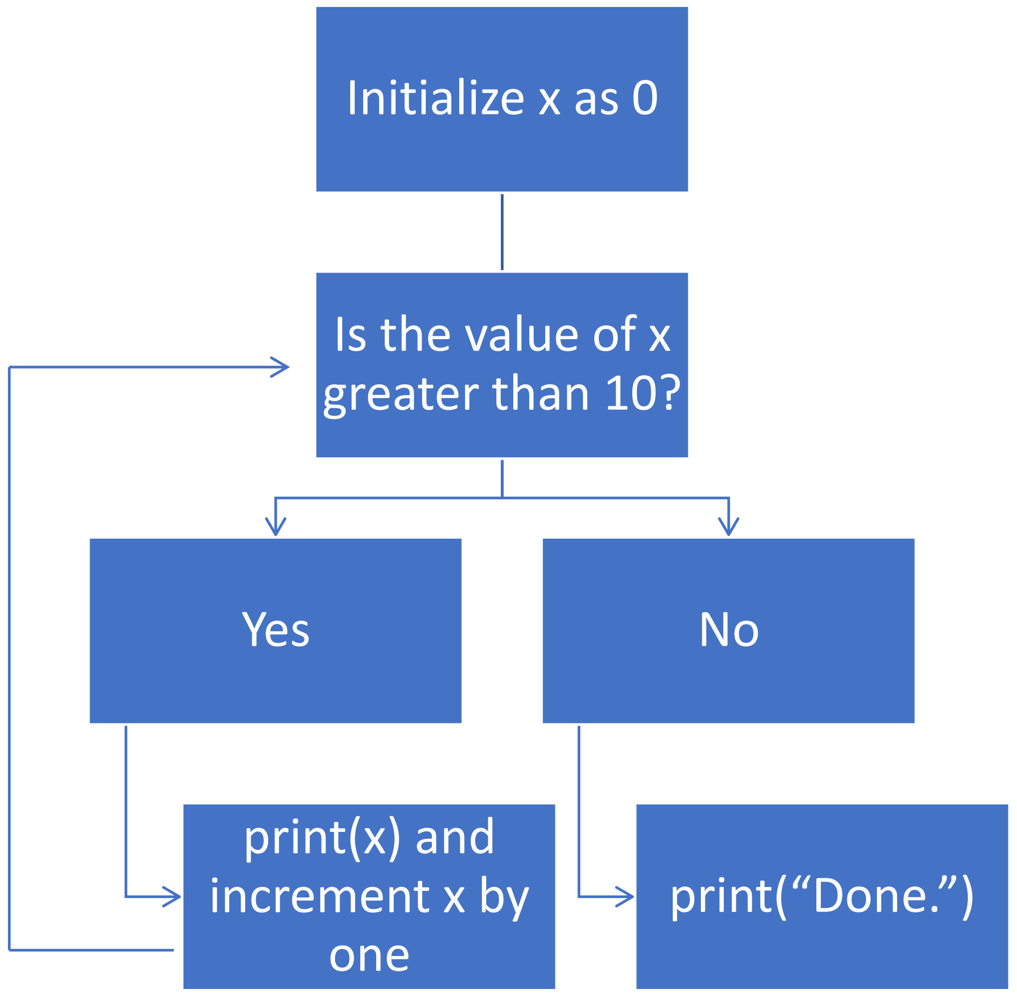
Matt’s Great Advice: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose (!), usually follow from a **poor design** in your code.

Drawing a **structural diagram**, **<u>before coding</u>**, greatly helps figuring out the right structure for your code.

Use prints in your loops to see them in action and confirm their behavior!

Use pen and paper as well if it helps!


Initialize x as 0
Is the value of x greater than 10?
Yes
No
print(x) and increment x by one
print(“Done.”)

Example: diagram for our while loop, counting from 1 to 10.

Matt’s Great advice

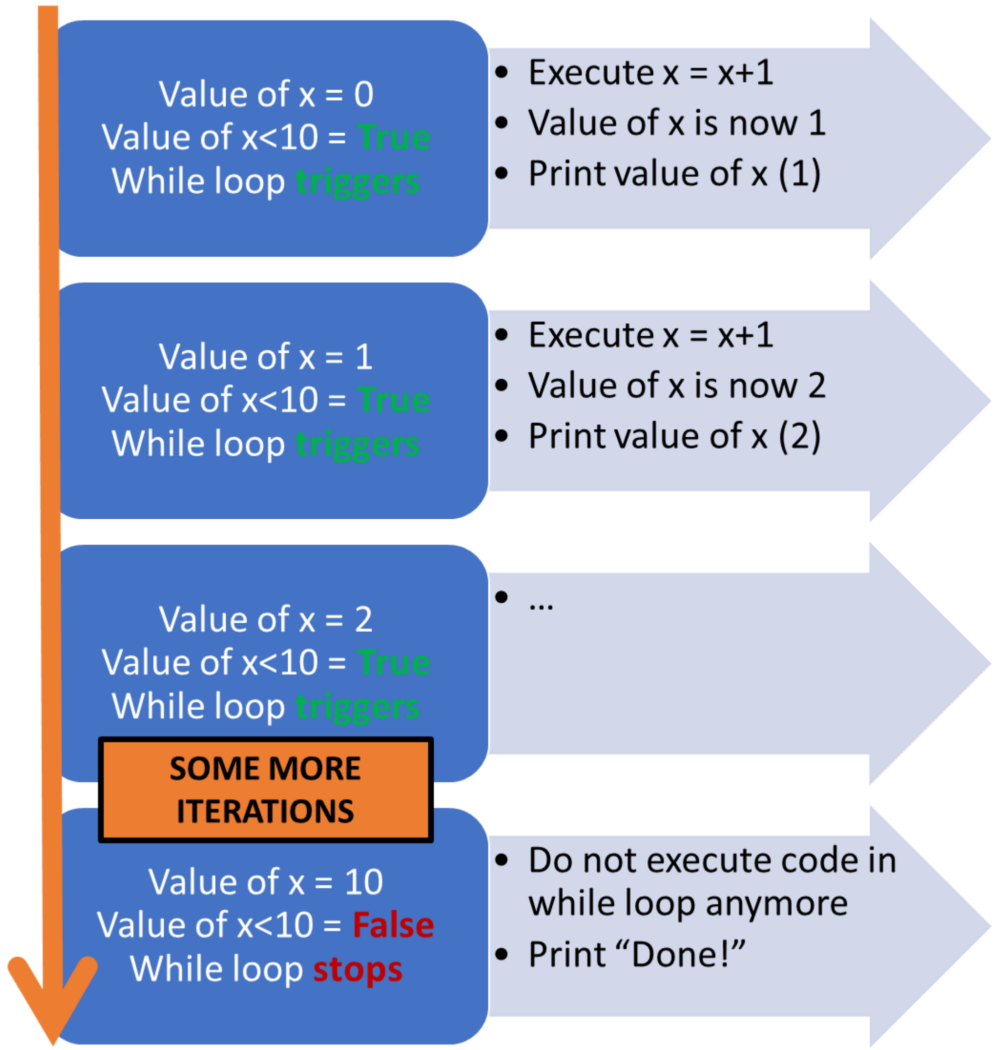
Matt’s Great Advice: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose (!), usually follow from a **poor design** in your code.

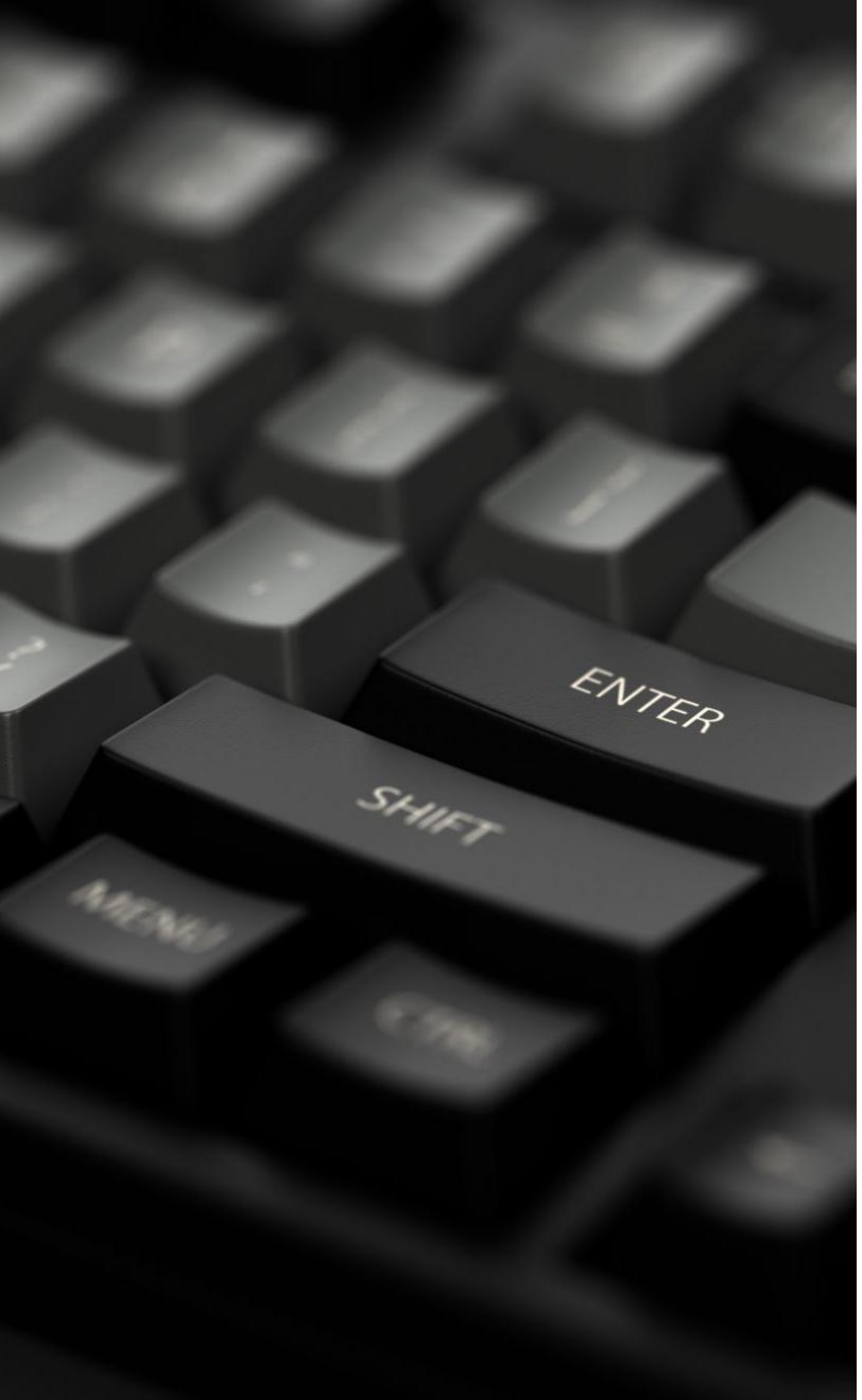
Drawing a **structural diagram**, **<u>before coding</u>**, greatly helps figuring out the right structure for your code.

Use prints in your loops to see them in action and confirm their behavior!

Use pen and paper as well if it helps!



Example: diagram for our while loop, counting from 1 to 10.



Conclusion (Chapter 5)

- What is the **while** statement?
- What are **infinite loops**, why are they problematic and **how to stop them**?
- What is the **break** statement and how to use it in while statements?
- What are **good practices** when it comes to using while loops?
- (If time allows, let us discuss recursion!)

Quiz time!

Let’s go!

slido

Please download and install the Slido app on all computers you use

What will be printed?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

What will happen here?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

Which statement should we use in the while loop, in order NOT to get stuck in a while loop?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

Which of the following statements is NOT true about break statements in while loops?

ⓘ Start presenting to display the poll results on this slide.

Practice activities for while/break

Let us practice the **while**/**break** concepts a bit, with three activities.

- Activity 1: How much experience needed for level n?
- Activity 2: How many hits can you take?
- Activity 3: Guess the card game, v2.

(And as usual, some extra practice/extra challenges!)