ILP 2023 – W2S3
While/Break statements

Matthieu DE MARI – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Outline (Week2, Session3 – W2S3)

- While statements
- Infinite loops and how to kill them
- The break statement
- (If time allows, recursion!)

The while statement

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

The while statement

The **while** statement is another type of **<u>conditional structure.</u>**

The **if** statement is the simplest **<u>conditional structure</u>**.

- **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.

⟺

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.**

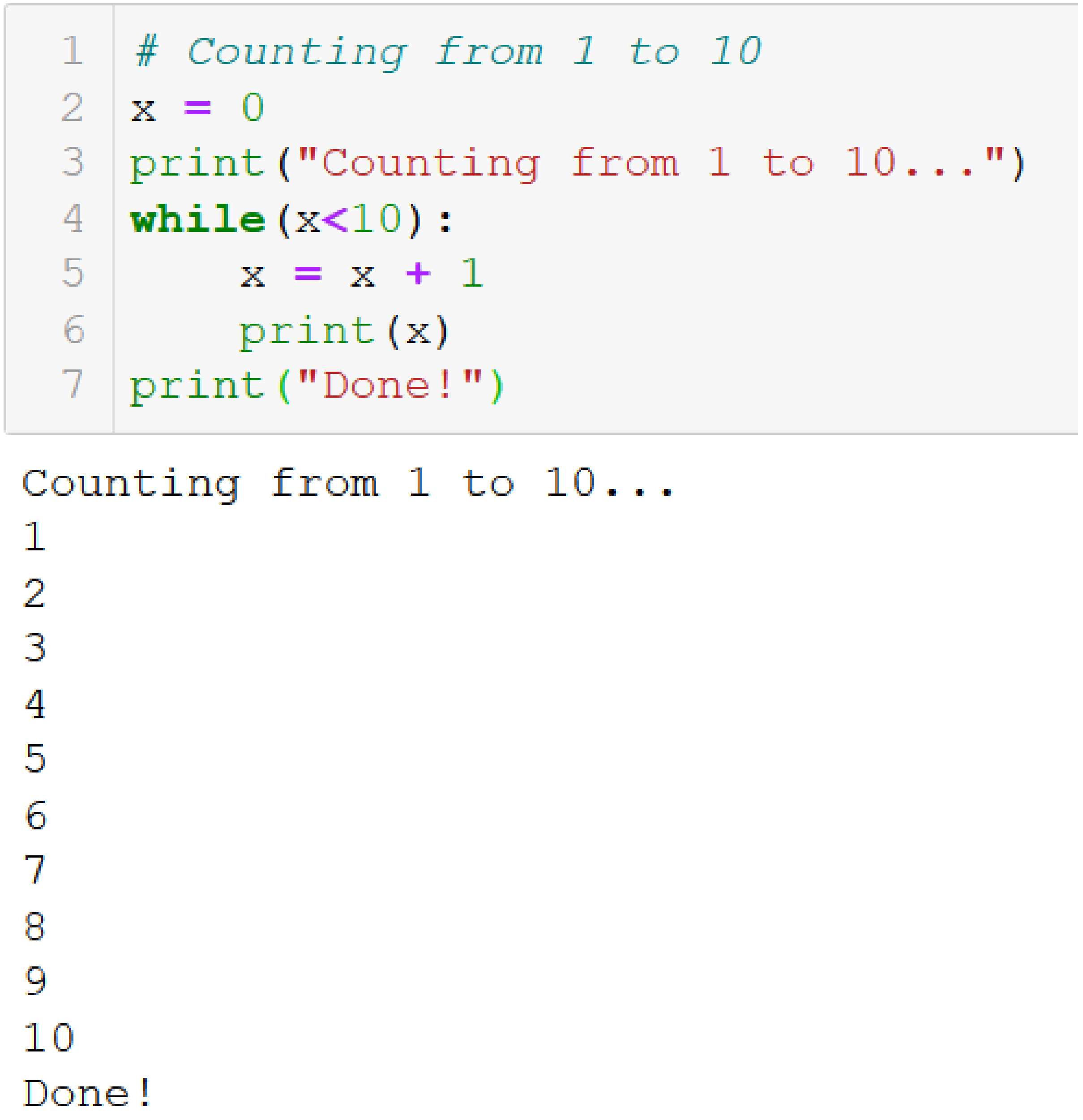
The **if** statement is the simplest **conditional structure**.

- **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.**

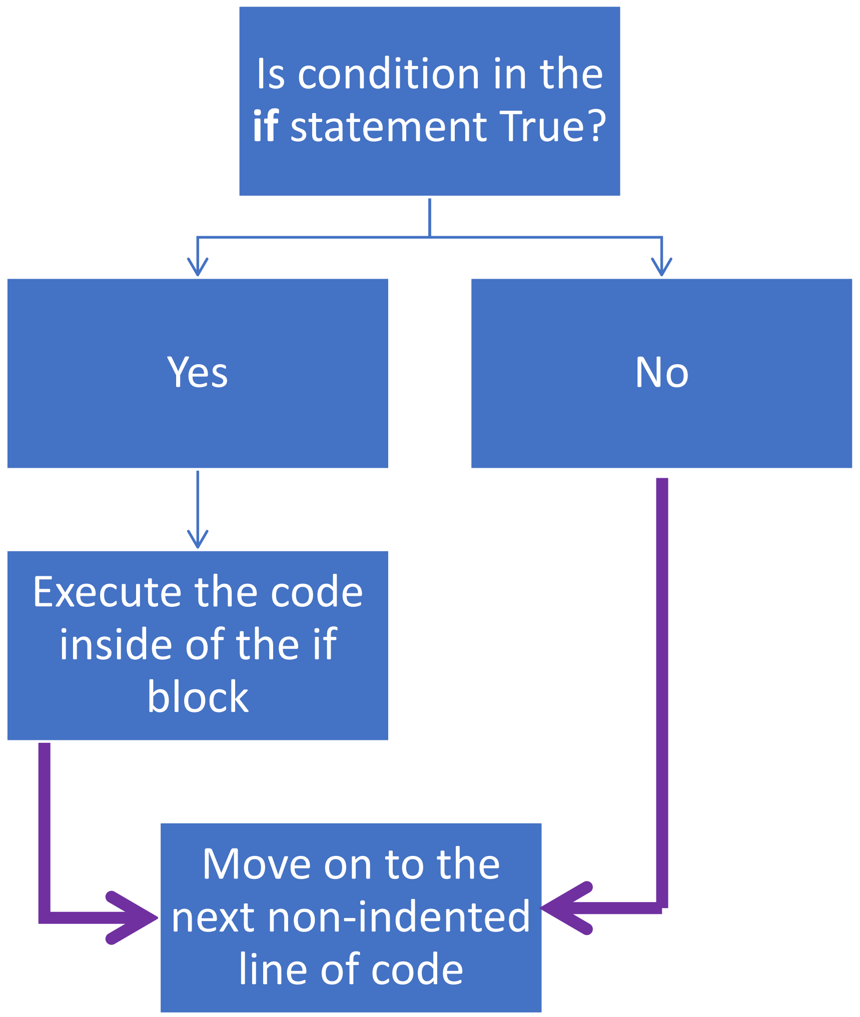
- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in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

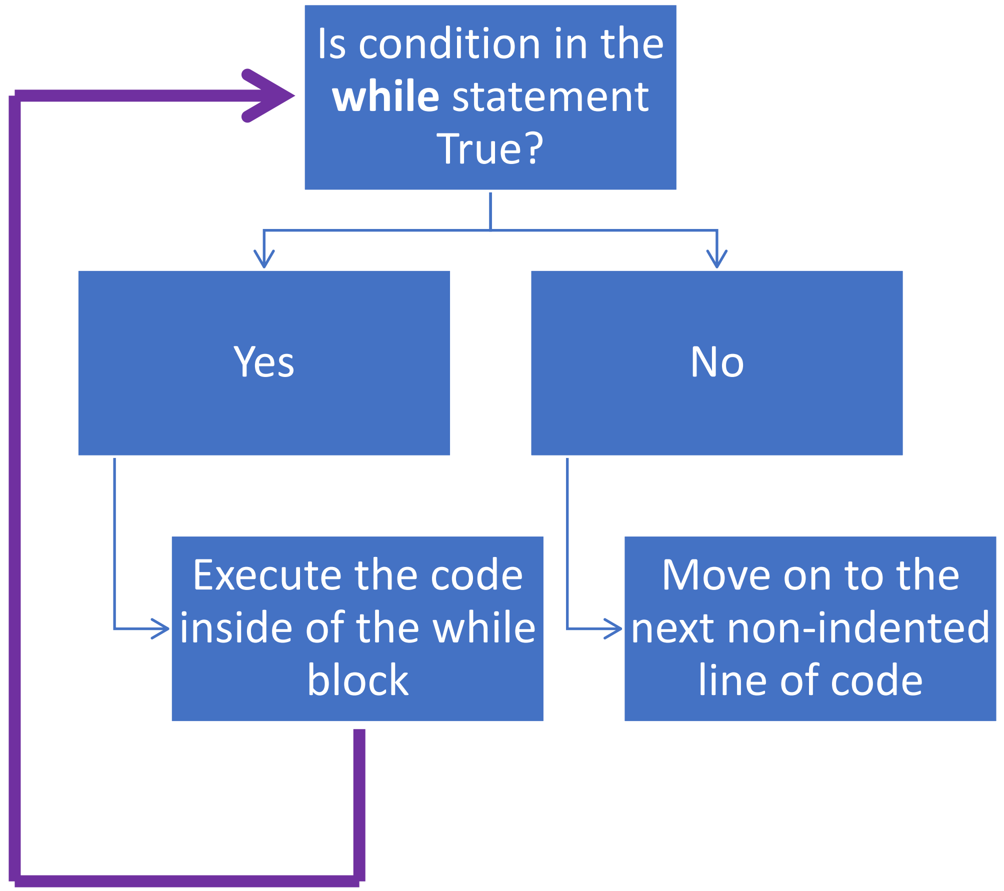


```python
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

```
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
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

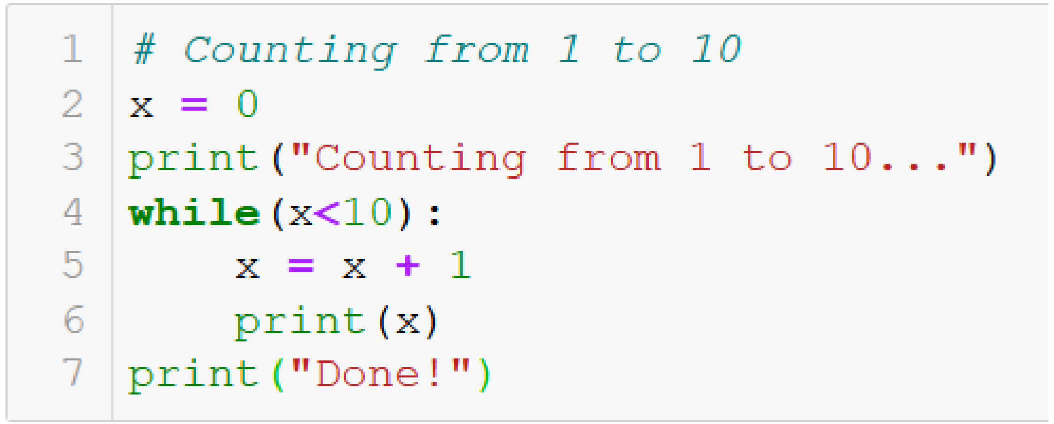
Infinite loops

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

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.



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

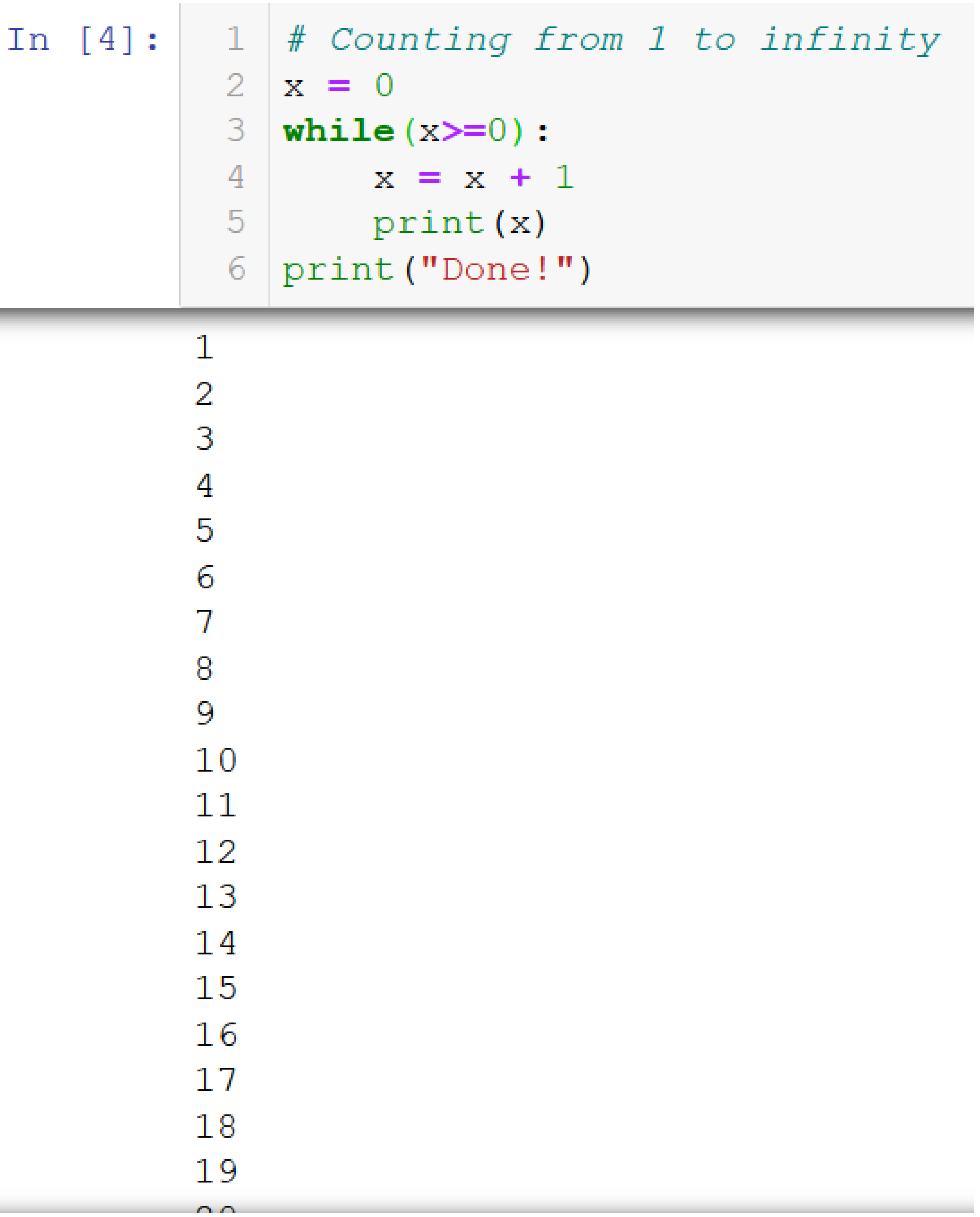
```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

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.

Otherwise, 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 and performs an emergency shutdown before exploding (**bad thing to do**),

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

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

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

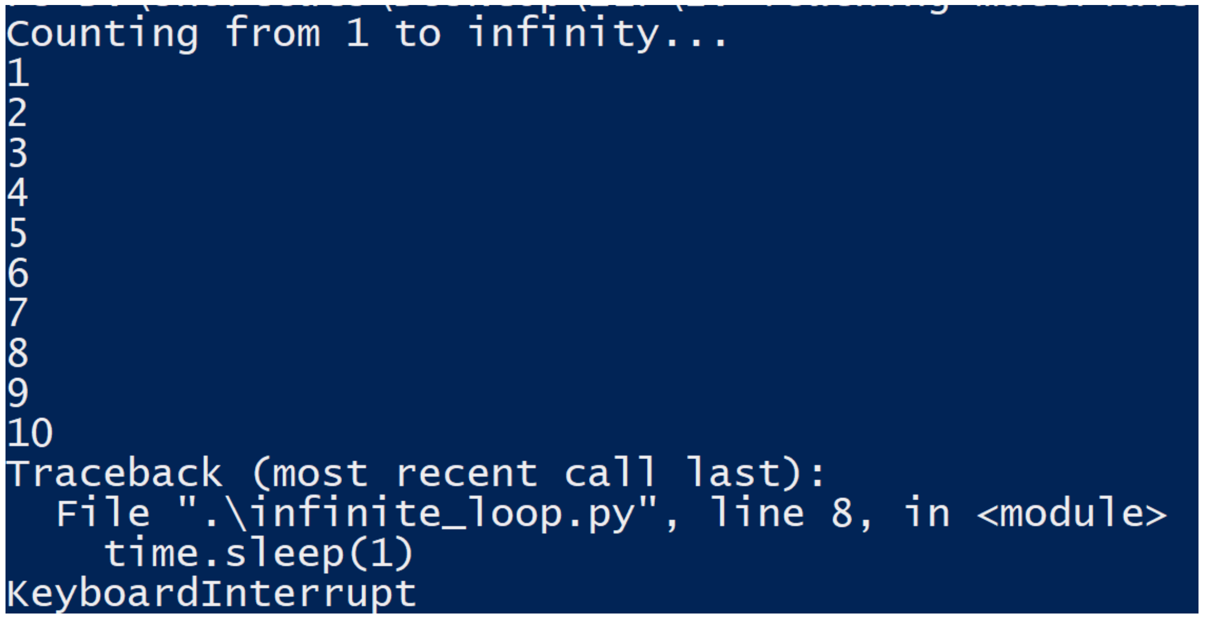
1. You decide to crash the program on purpose and kill the loop manually.

Infinite loops 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 with **CTRL+C** (or **CMD+C** on mac), in console mode and most IDEs.



```
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
```

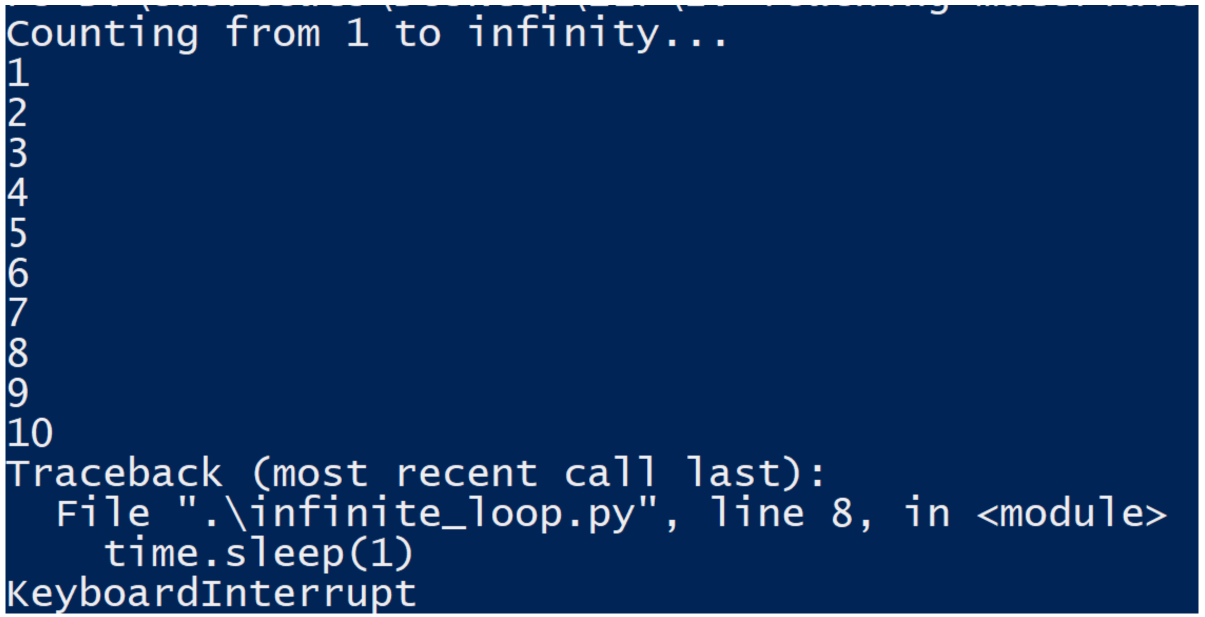
Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

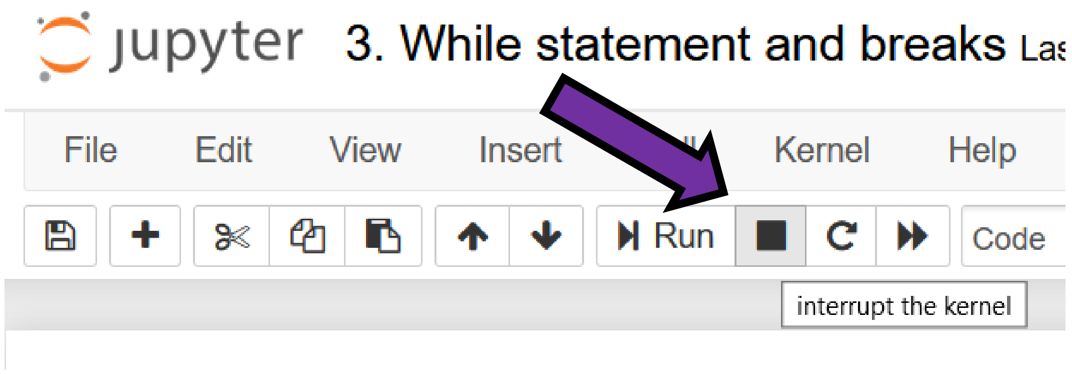
1. You decide to crash the program on purpose and kill the loop manually.

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

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

Matt’s Great advice #7

Matt’s Great Advice #7: 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.

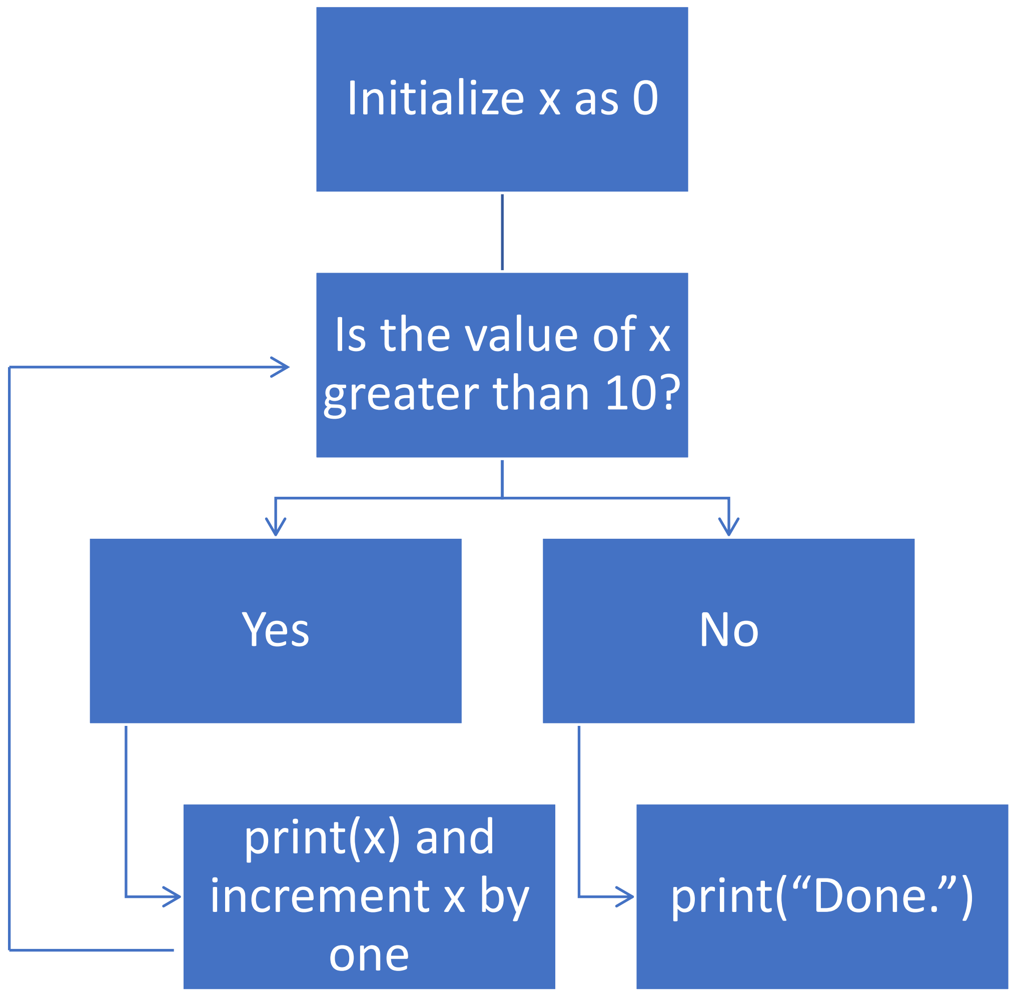
Drawing a **structural diagram**, **before coding**, greatly helps figuring out the right structure for your code.

Matt’s Great advice #7

Matt’s Great Advice #7: 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.


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.

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.

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.

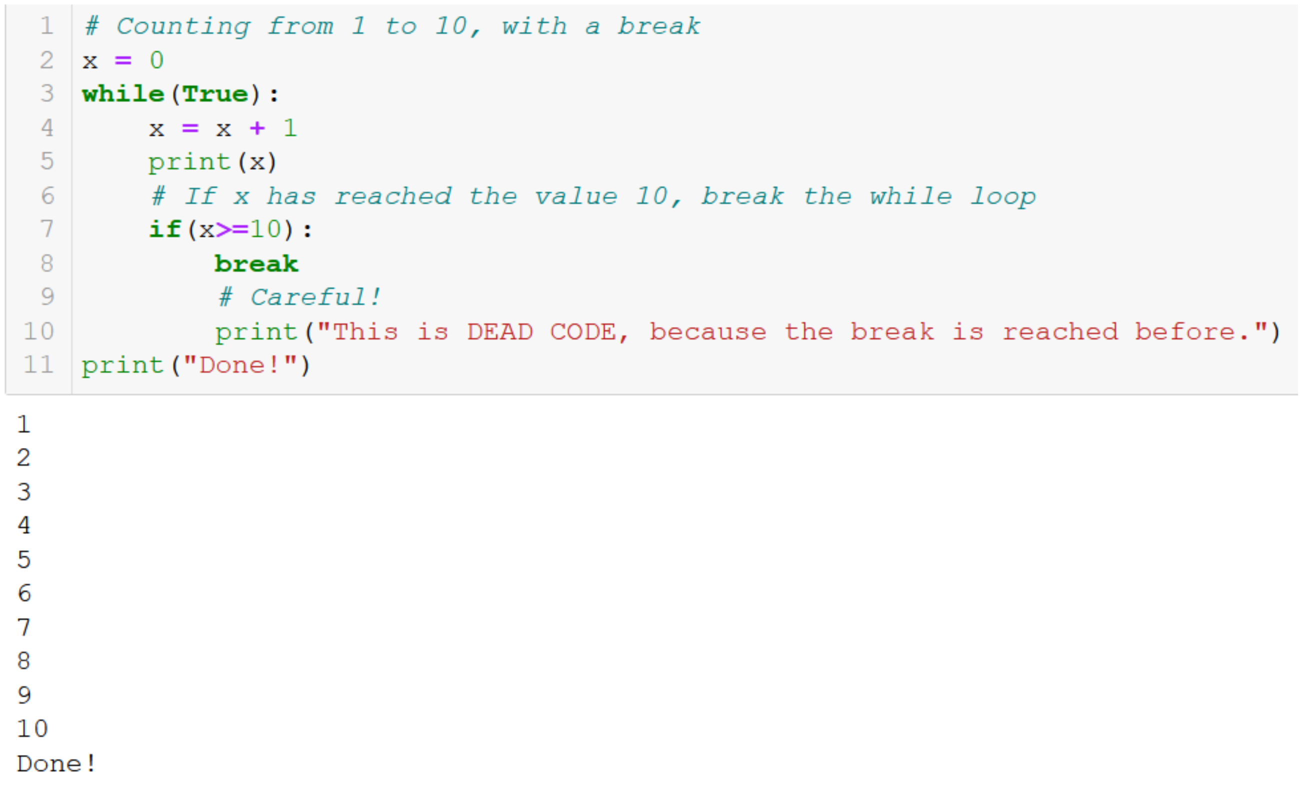
Infinite loops: the **break** statement

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.

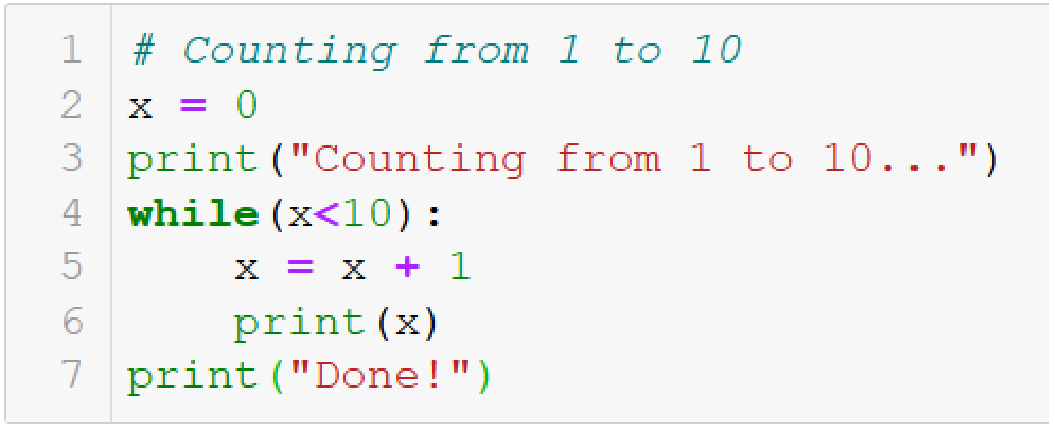


```python
# Counting from 1 to 10, with a break
x = 0
while(True):
    x = x + 1
    print(x)
    # If x has reached the value 10, break the while loop
    if(x>=10):
        break
        # Careful!
        print("This is DEAD CODE, because the break is reached before.")
print("Done!")
```

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

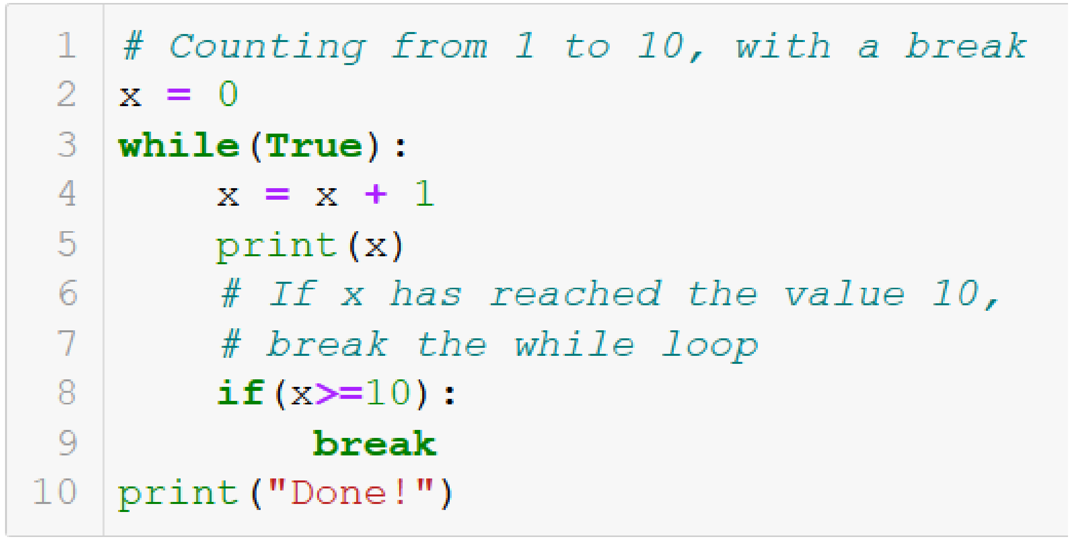
Standard **while** vs. infinite **while** + **break**

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 #8

Matt’s Great Advice #8: Avoid the infinite loops, if possible.

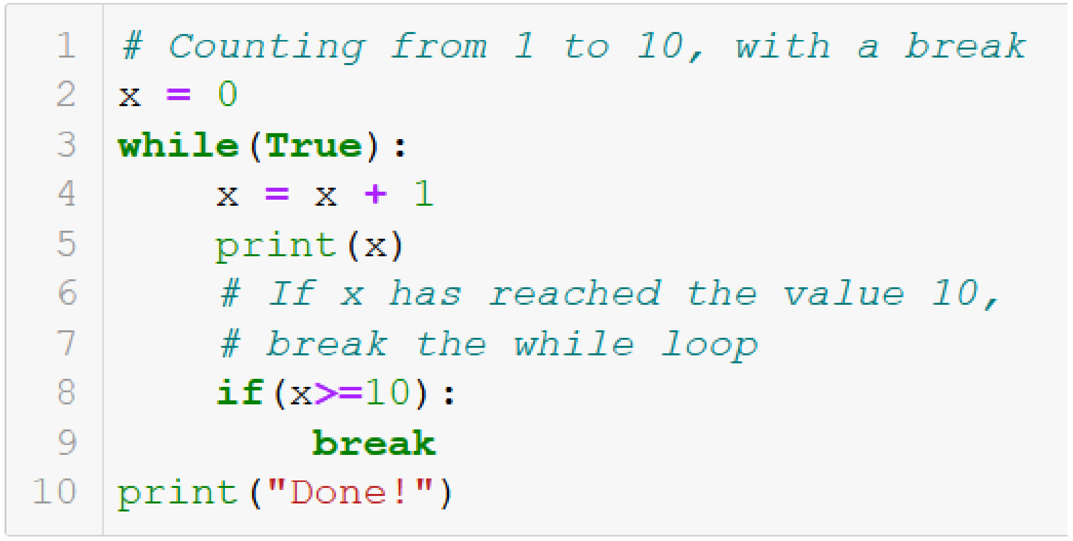
Relying on an **infinite while** loop with a **break** is **risky**, and should be avoided when possible.

Matt’s Great advice #8

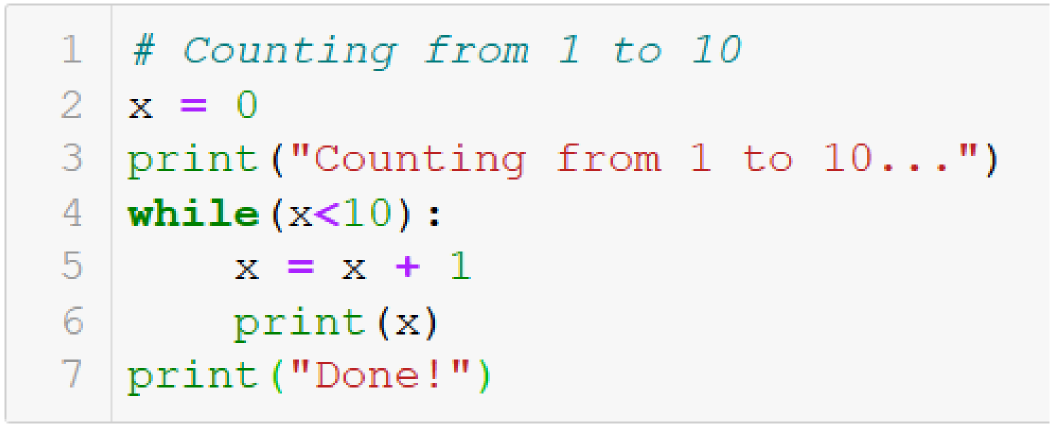
Matt’s Great Advice #8: Avoid the infinite loops, if possible.

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!")
```

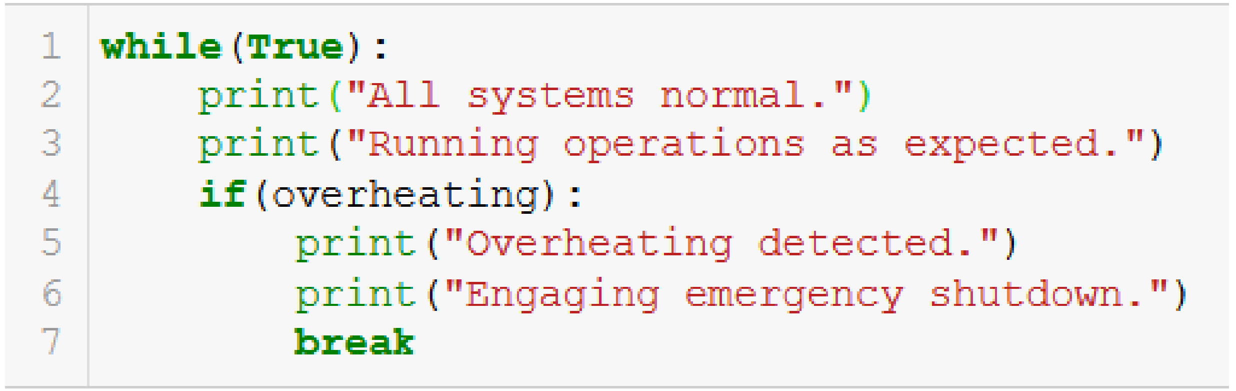
Matt’s Great advice #8

Matt’s Great Advice #8: Avoid the infinite loops, if possible.

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**.



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

Practice activities for **while**/**break**

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

Conclusion

- While statements
- Infinite loops and how to kill them
- The break statement
- (If time allows, recursion!)

Up for a challenge?
(in the Extra challenges folder)

Challenge: Activity 1+ - How many hits can you take (extra challenge).ipynb

- Similarly, as in other challenges…
- Do not use any conditional statement (**if**/**while**)
- **Hint:** use a bit of maths on sequences!