ILP 2021 – W2S2
If/Elif/Else statements,
While/Break statements

Matthieu DE MARI – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Outline (Week2, Session2 – W2S2)

- The if statement
- The elif statement
- The else statement
- Dead code and code structure
- Nested ifs
- While statements
- Infinite looks and how to kill them
- The break statement
- (If time allows, recursion!)

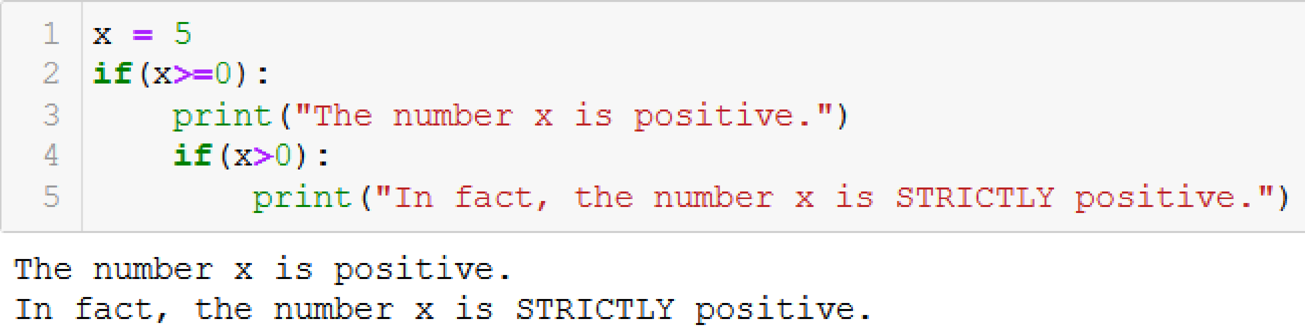
Nested if structures

Definition (nested if structure):
A **nested if structure** is a structure which includes one or multiple **if** statement(s), inside another **if** statement.

Nested if structures

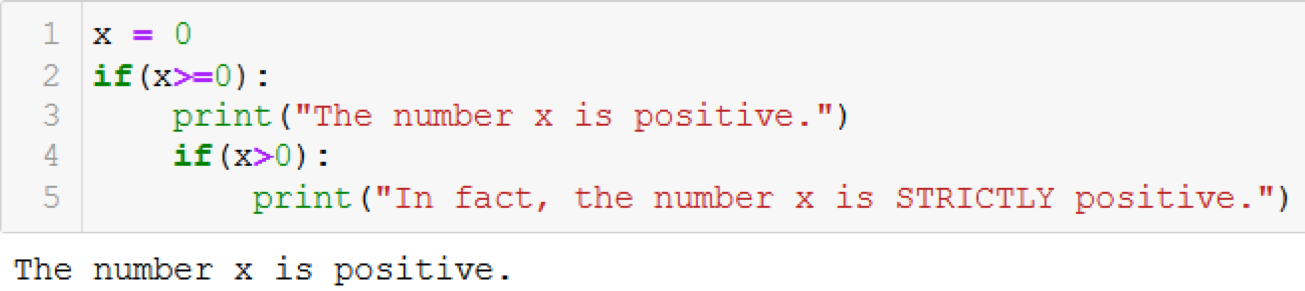
Definition (nested if structure):
A **nested if structure** is a structure which includes one or multiple if statement(s), inside another if statement.

These are typically used to check additional conditions, based on whether another condition has been satisfied or not.



```python
x = 5
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```



```python
x = 0
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

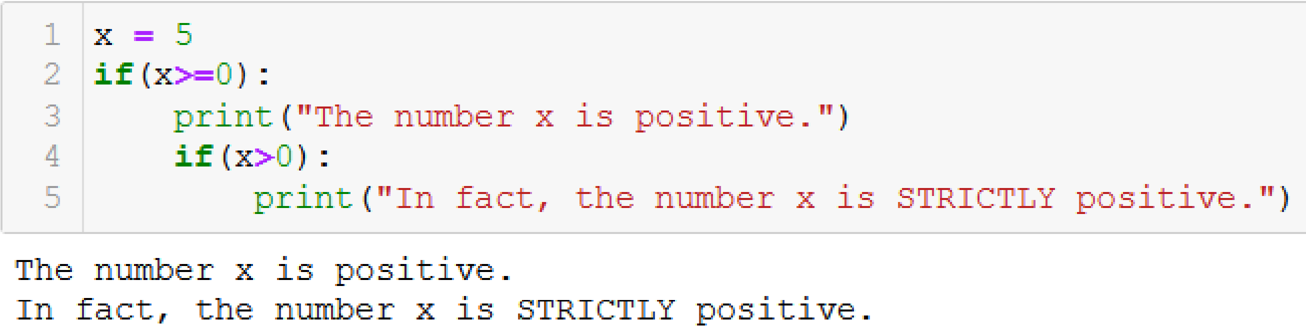
```
The number x is positive.
```

Nested if structures

Definition (nested if structure):
A **nested if structure** is a structure which includes one or multiple if statement(s), inside another if statement.

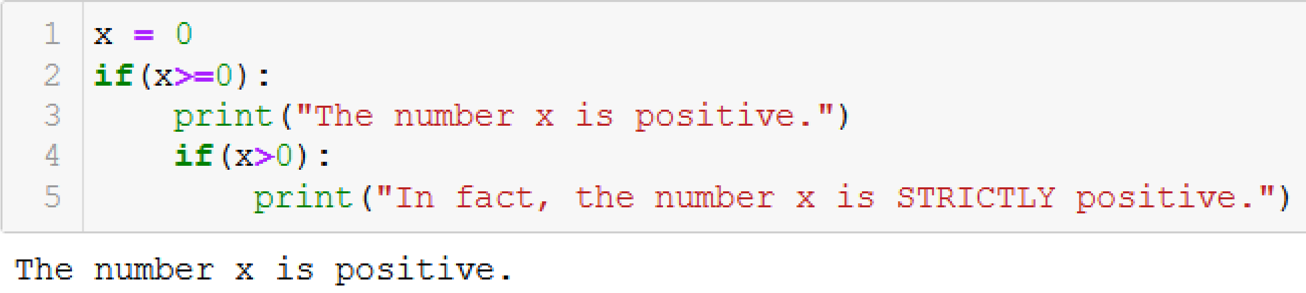
These are typically used to check additional conditions, based on whether another condition has been satisfied or not.

Each if might have its own elif/else statements, placed on the same indentation level.



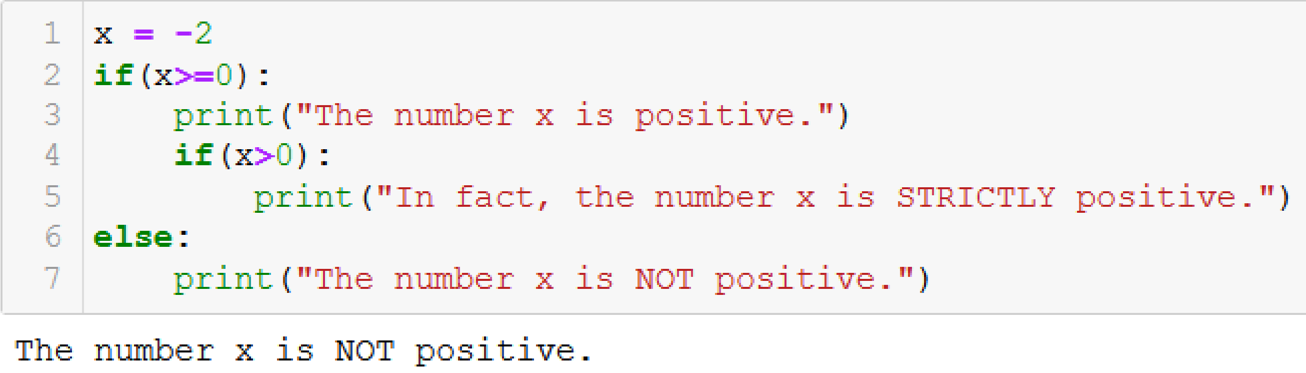
```python
x = 5
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```



```python
x = 0
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

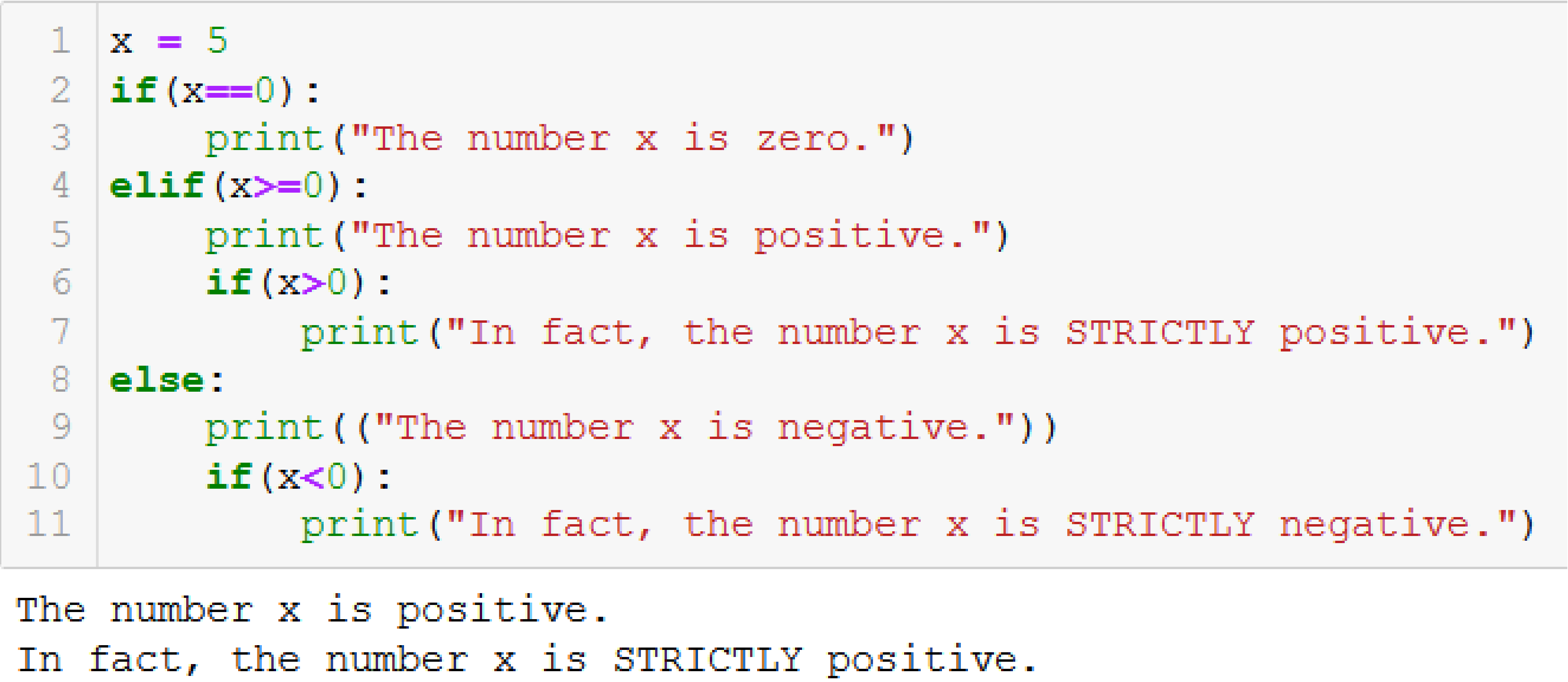
```
The number x is positive.
```



```python
x = -2
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print("The number x is NOT positive.")
```

```
The number x is NOT positive.
```

Nested if structures

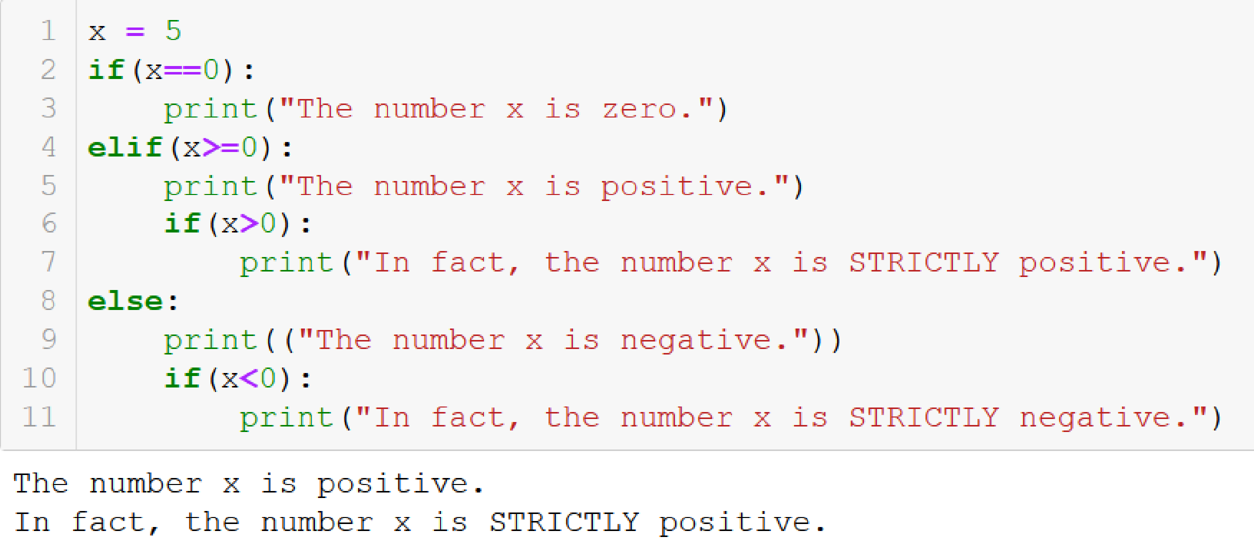


```python
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

Nested if structures vs. combined conditionals

Nested **if** structures can, most of the time, be rewritten with combined conditionals (using **and**/**or** Boolean operators).



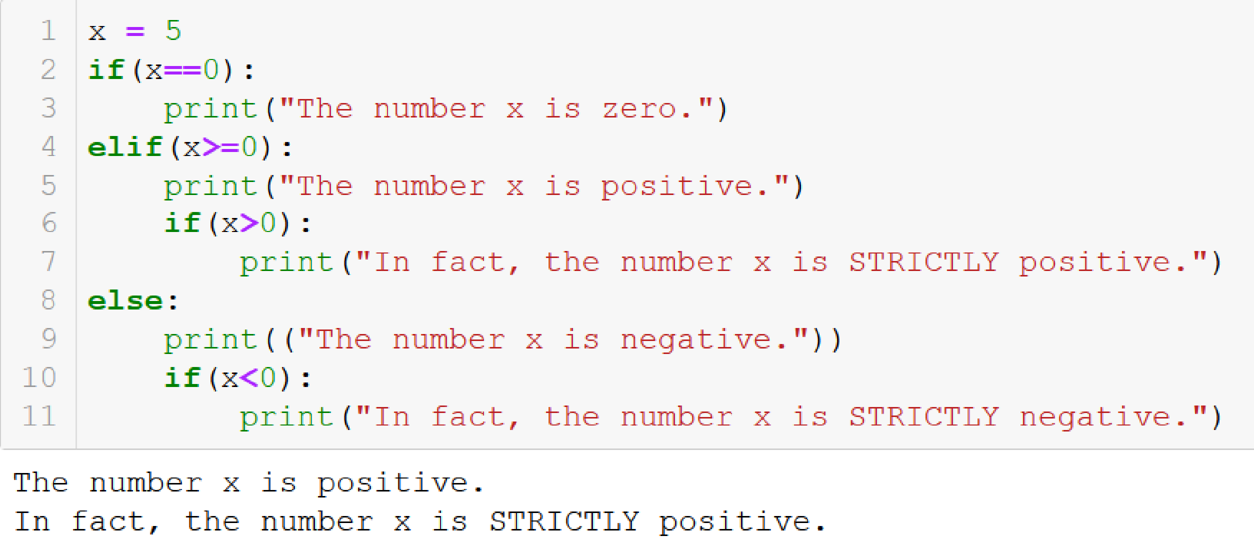
```
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

Nested if structures vs. combined conditionals

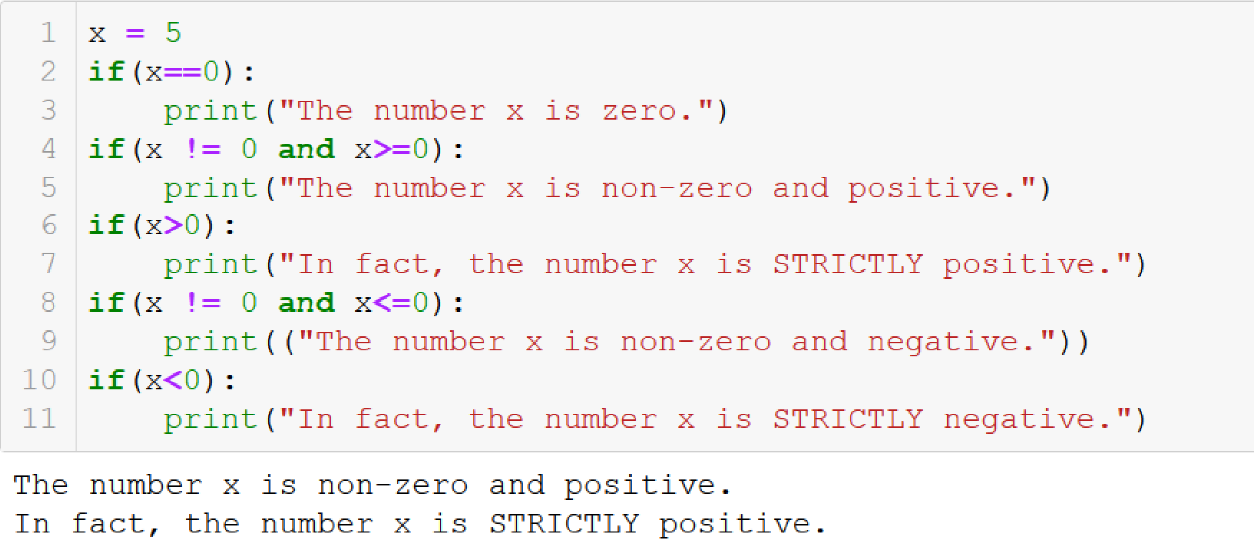
Nested **if** structures can, most of the time, be rewritten with combined conditionals (using **and**/**or** Boolean operators).

For instance, both structures on the right are equivalent.



```
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```



```
x = 5
if(x==0):
    print("The number x is zero.")
if(x != 0 and x>=0):
    print("The number x is non-zero and positive.")
if(x>0):
    print("In fact, the number x is STRICTLY positive.")
if(x != 0 and x<=0):
    print(("The number x is non-zero and negative."))
if(x<0):
    print("In fact, the number x is STRICTLY negative.")
```

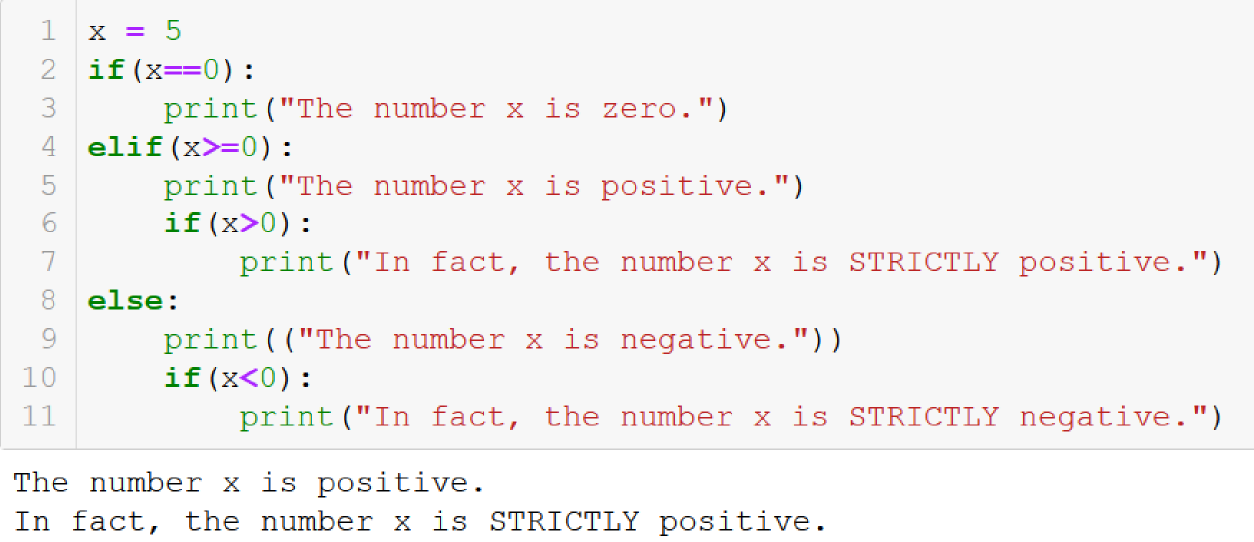
```
The number x is non-zero and positive.
In fact, the number x is STRICTLY positive.
```

Nested if structures vs. combined conditionals

Nested **if** structures can, most of the time, be rewritten with combined conditionals (using **and**/**or** Boolean operators).

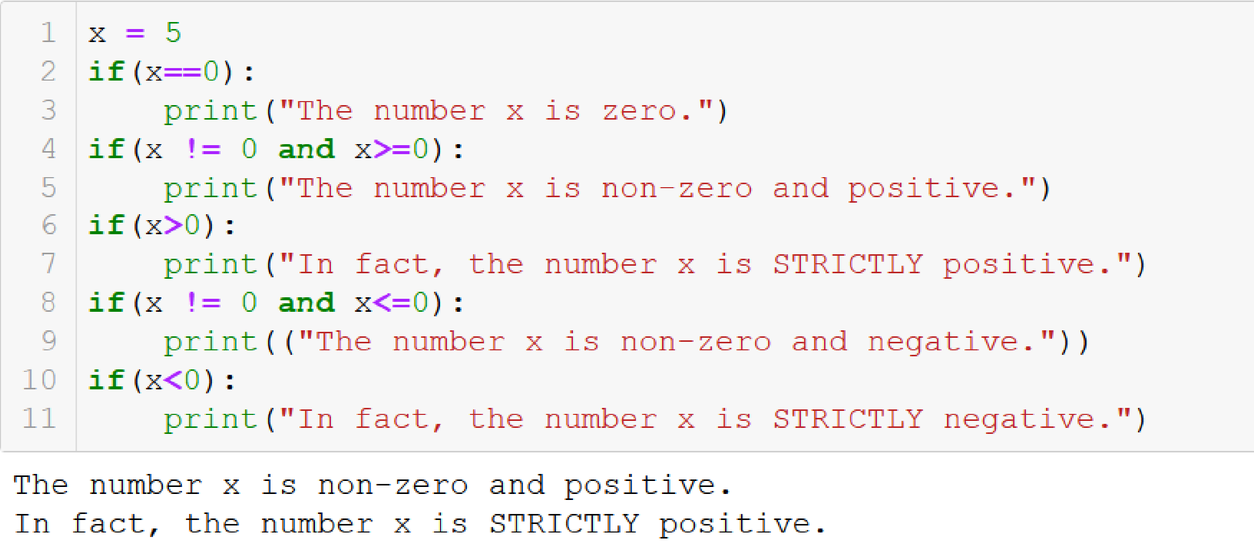
For instance, both structures on the right are equivalent.

Personal preference: Whenever possible, try to avoid the nested **if** structures. They are often overly complicated and prone to errors in designing the code.



```python
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```



```python
x = 5
if(x==0):
    print("The number x is zero.")
if(x != 0 and x>=0):
    print("The number x is non-zero and positive.")
if(x>0):
    print("In fact, the number x is STRICTLY positive.")
if(x != 0 and x<=0):
    print(("The number x is non-zero and negative."))
if(x<0):
    print("In fact, the number x is STRICTLY negative.")
```

```
The number x is non-zero and positive.
In fact, the number x is STRICTLY positive.
```

Activity 3 - Race and class check

Write a function **character_creation()**, according to the following requirements.

- The function will **receive two parameters**: **user_race** and **user_class**.
- For simplicity, only **three races** are available: **Human, Elf,** and **Dwarf**.
- For simplicity, **only four classes** are available: **Warrior, Hunter, Mage** and **Priest**.
- **Humans** can play **all classes**.
- **Elves** cannot be **warriors**.
- **Dwarves** cannot be **mages or priests**.
- The function should **not return anything.**
- It should **print** "You cannot play a character that is ...{race} and ...{class}.", with **blanks filled** accordingly, **if the combination of user_race and user_class is not acceptable**.
- Not acceptable here means that its race and/or class is not among the ones listed above, or the combination is not permitted, as listed above.
- **If the combination is valid**, it should **print** "Your character's race is ...{race} and your character's class is ...{class}.", with blanks filled accordingly.

The while statement

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

The while statement

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

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.

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.**

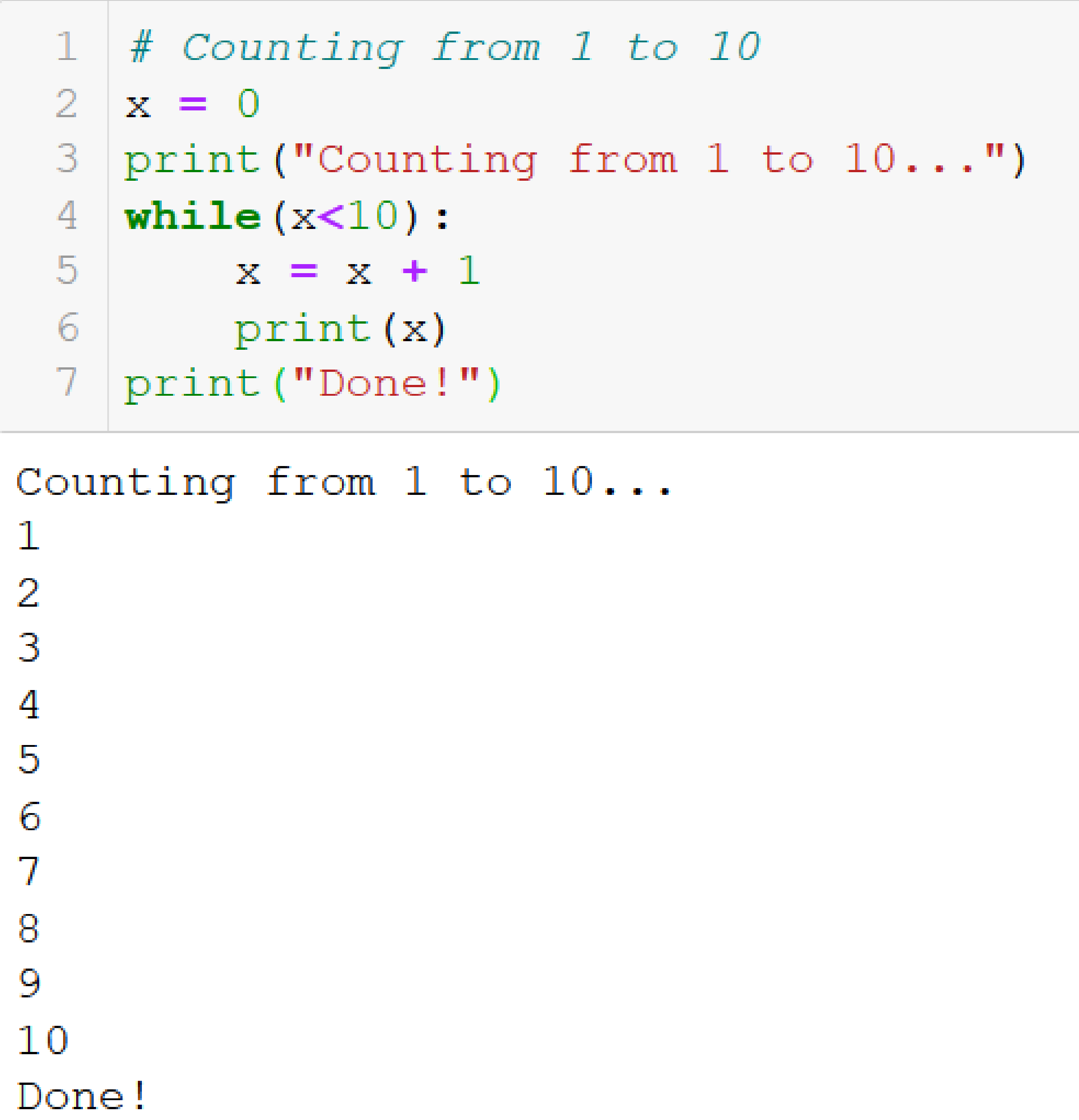
The **if** statement is the simplest **conditional structure**.

- **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.**

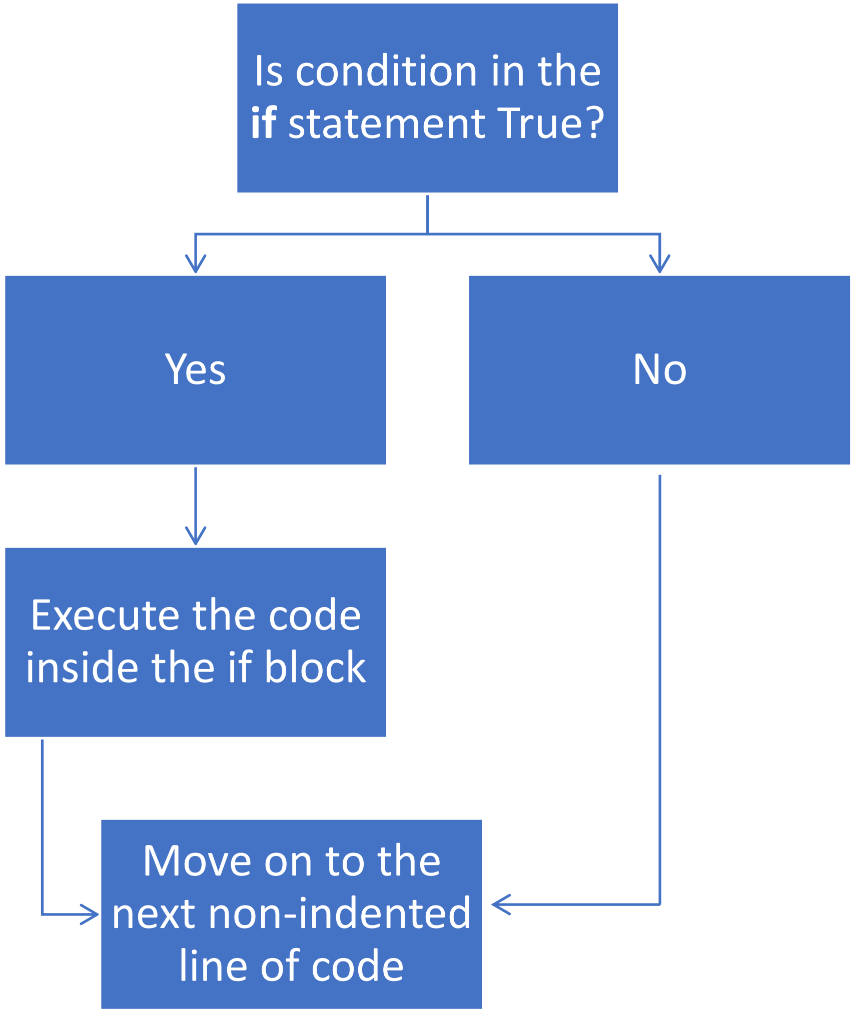
- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in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



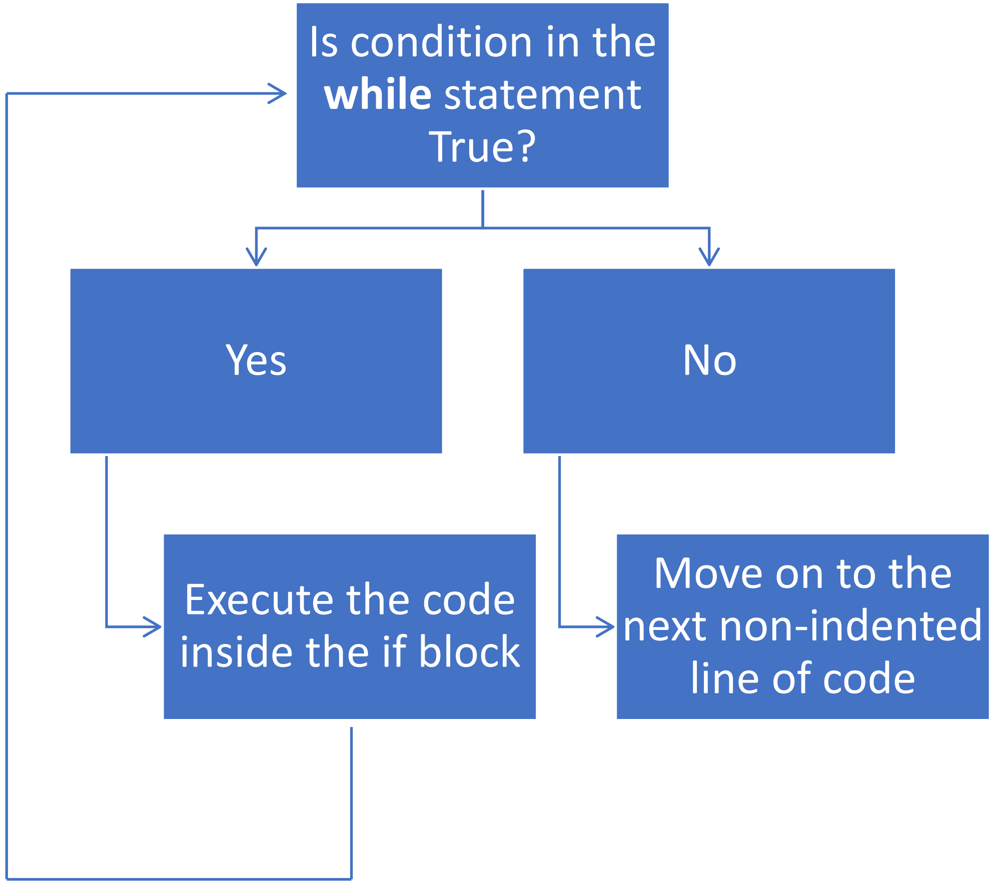
Is condition in the **if** statement True?

Yes

No

Execute the code inside 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 the if block

Move on to the next non-indented line of code

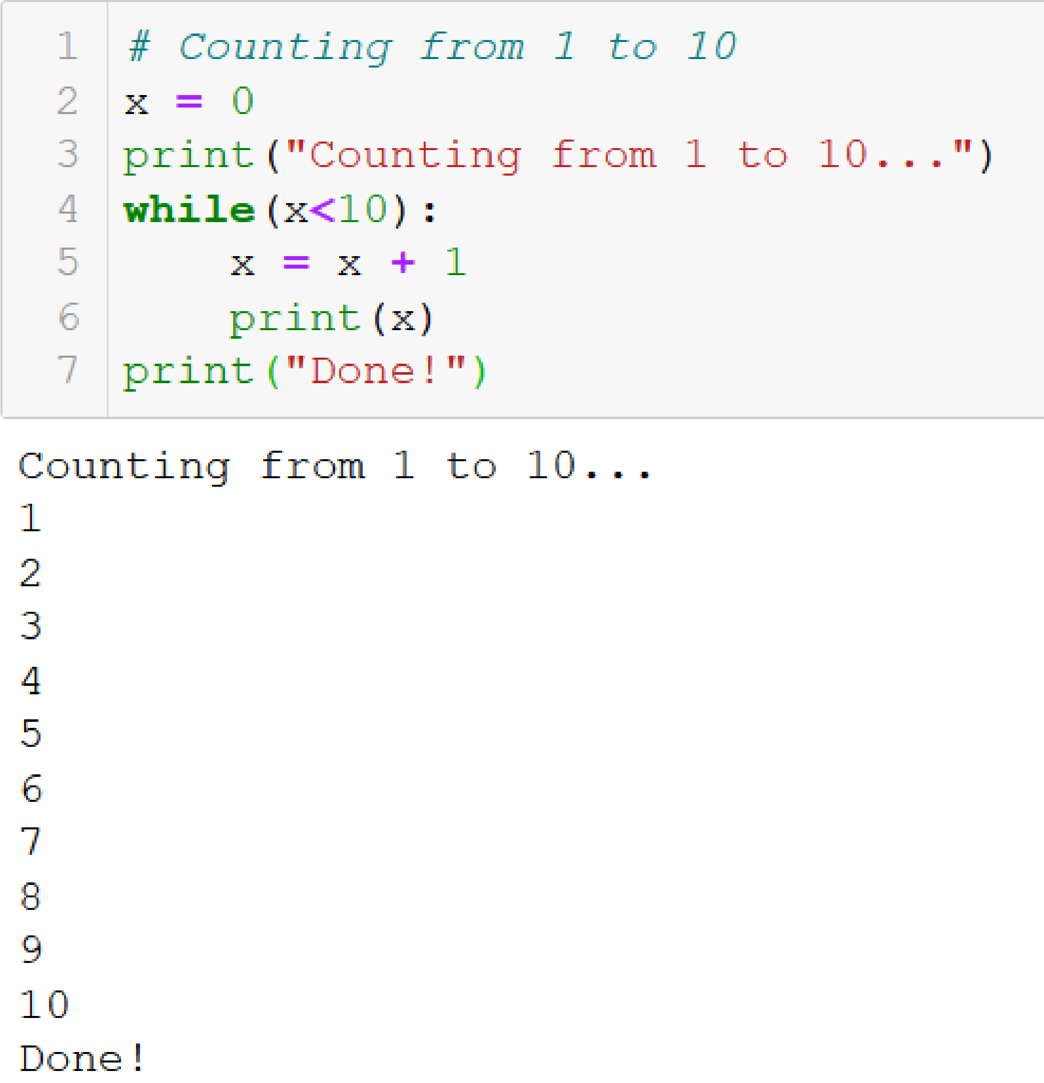
Infinite loops

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!")
```

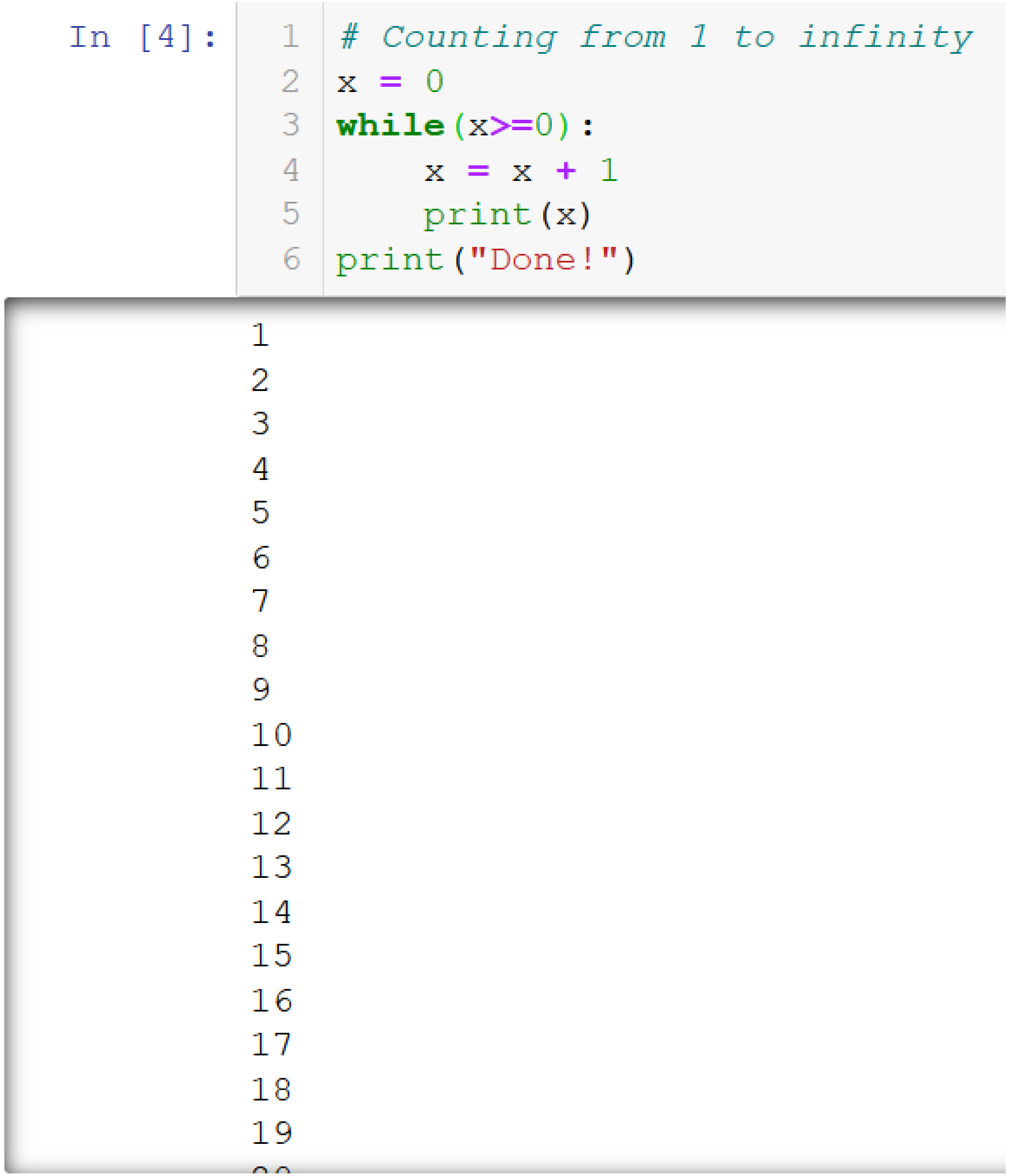
```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

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**.



```python
# 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

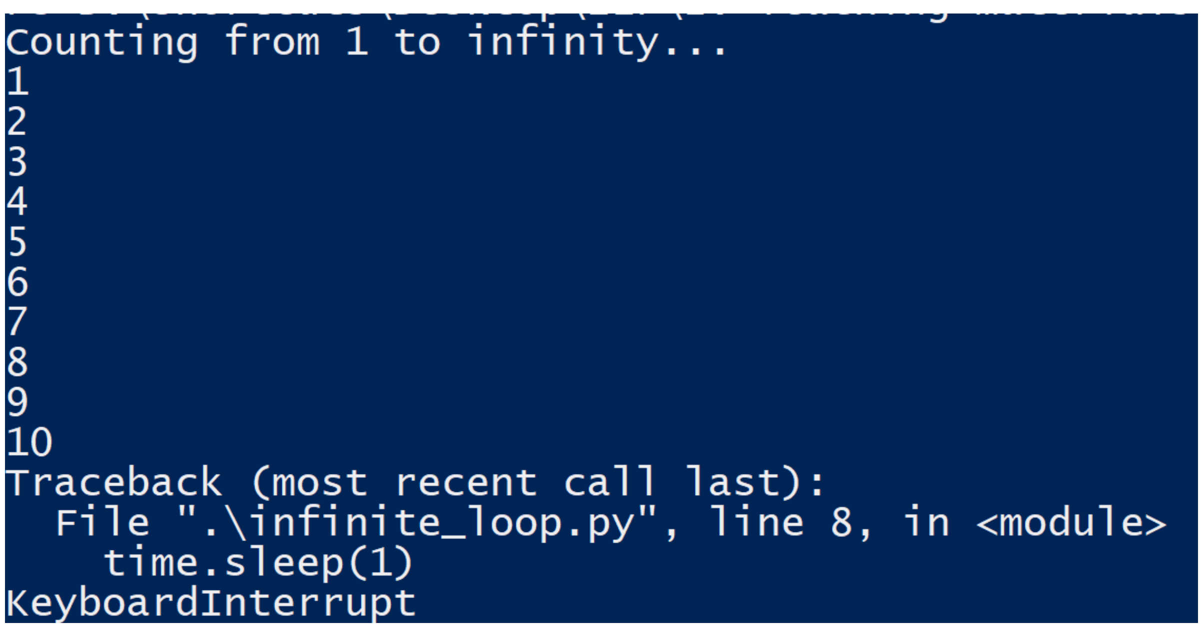
1. You decide to crash the program on purpose and kill the loop manually.

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
```

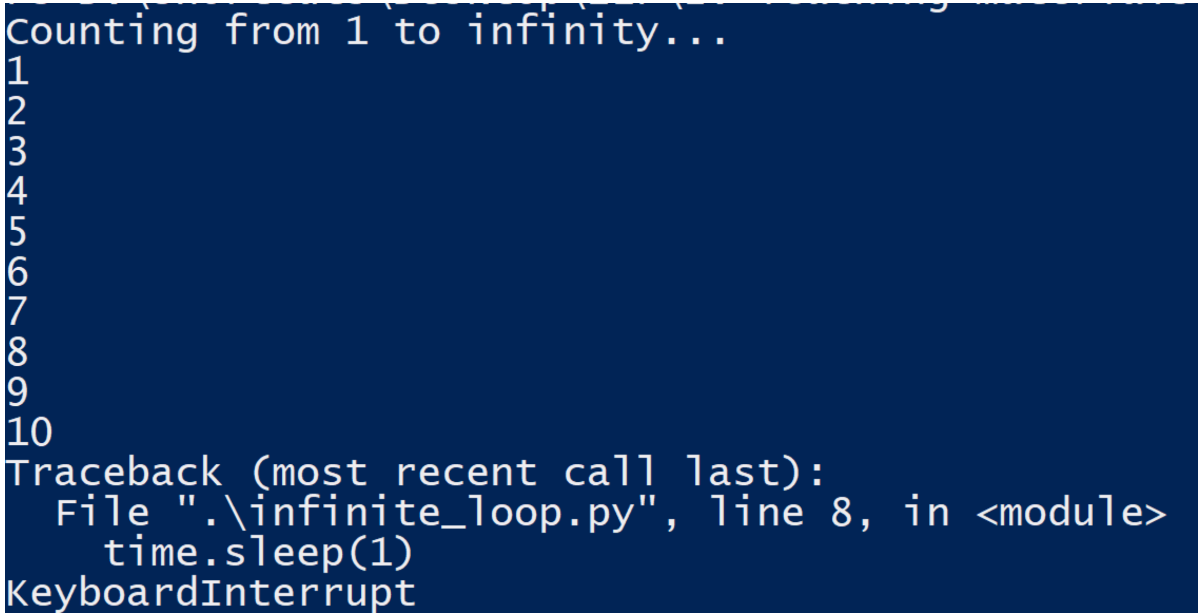
Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

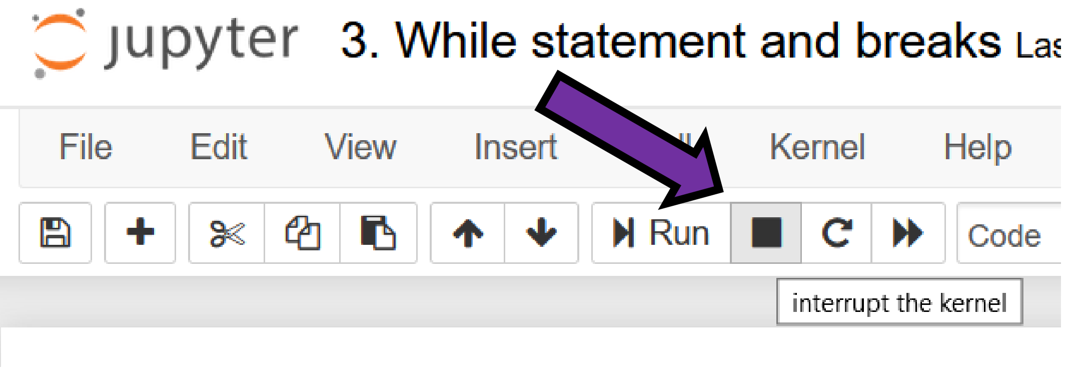
1. You decide to crash the program on purpose and kill the loop manually.

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
```



3. While statement and breaks Las

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), if 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.

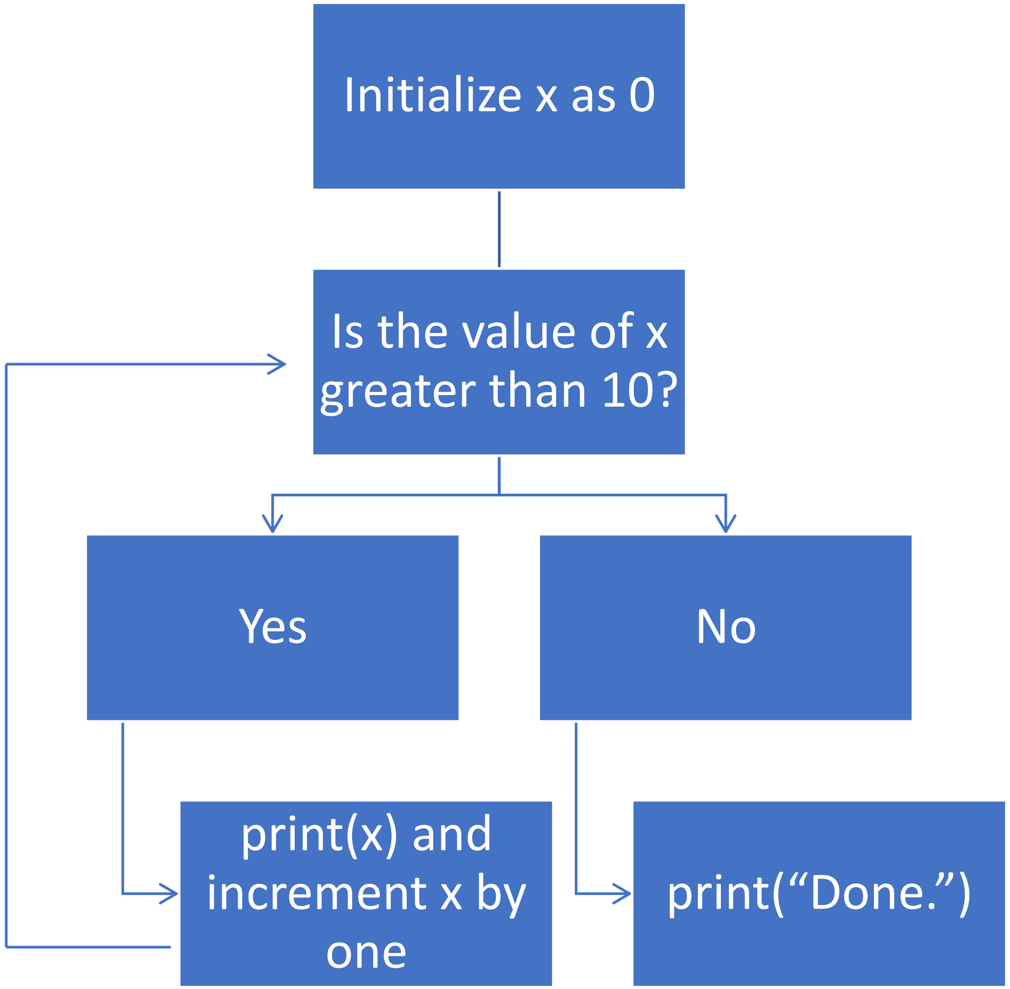
Drawing a **structural diagram**, **<u>before coding</u>**, greatly helps figuring out the right structure for your code.

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**, **before coding**, 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.

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.

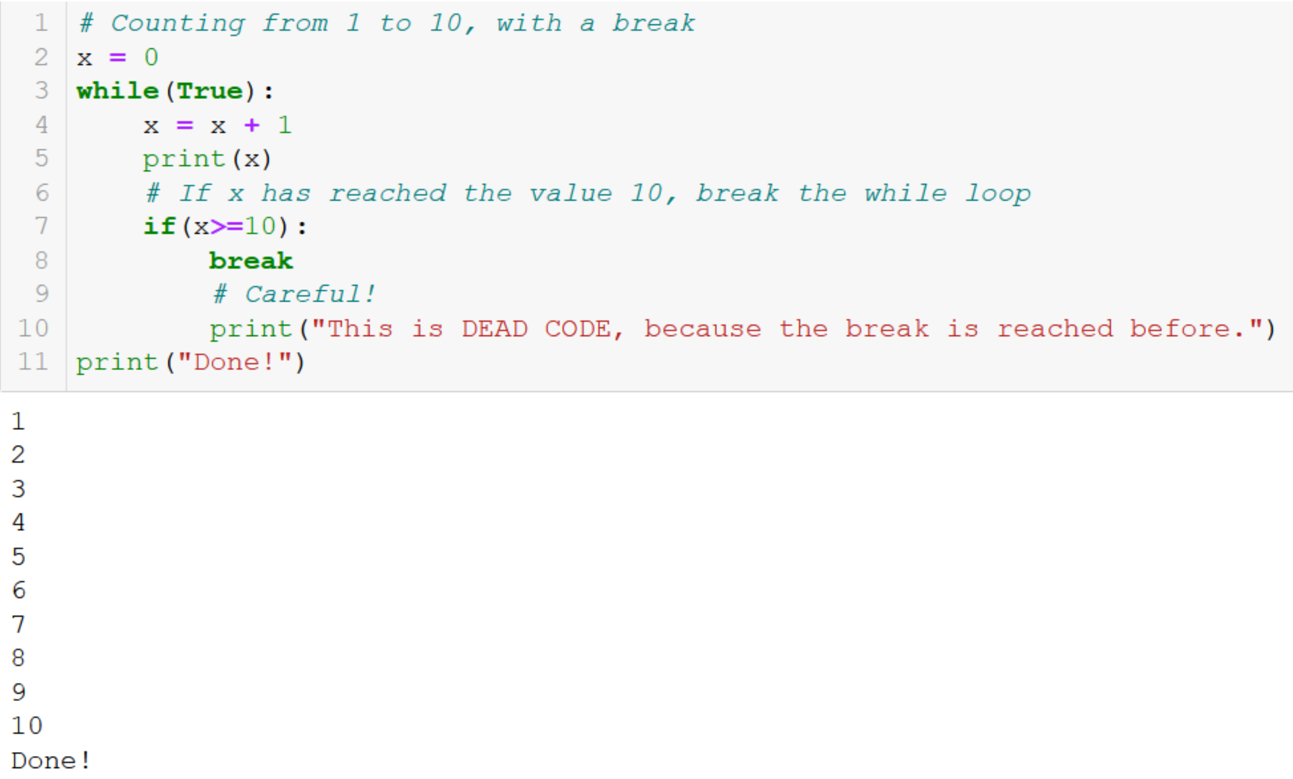
Infinite loops: the break statement

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.

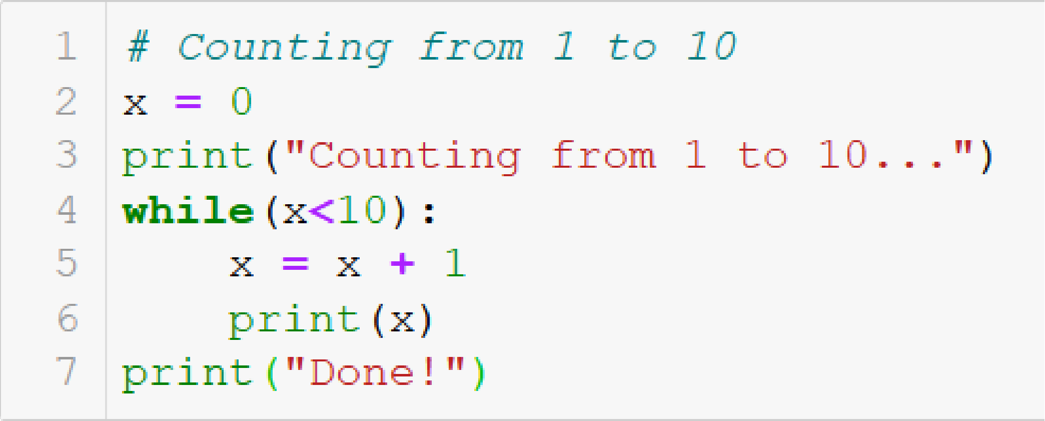


```python
# Counting from 1 to 10, with a break
x = 0
while(True):
    x = x + 1
    print(x)
    # If x has reached the value 10, break the while loop
    if(x>=10):
        break
        # Careful!
        print("This is DEAD CODE, because the break is reached before.")
print("Done!")
```

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

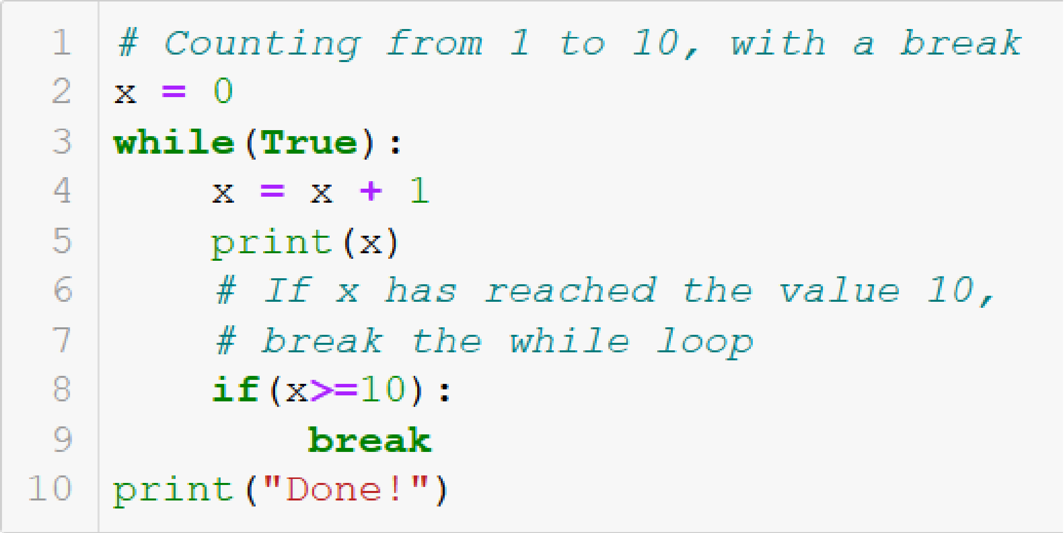
Standard **while** vs. infinite **while** + **break**

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



```
# 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.



```
# 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.

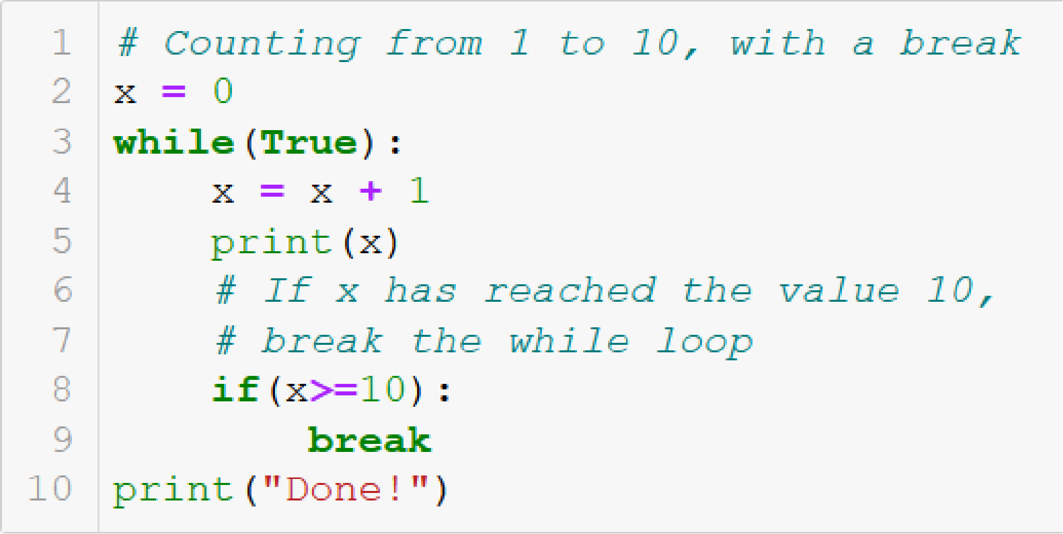
Relying on an **infinite while** loop with a **break** is **risky**, and should be avoided when possible.

Matt’s Great advice #8

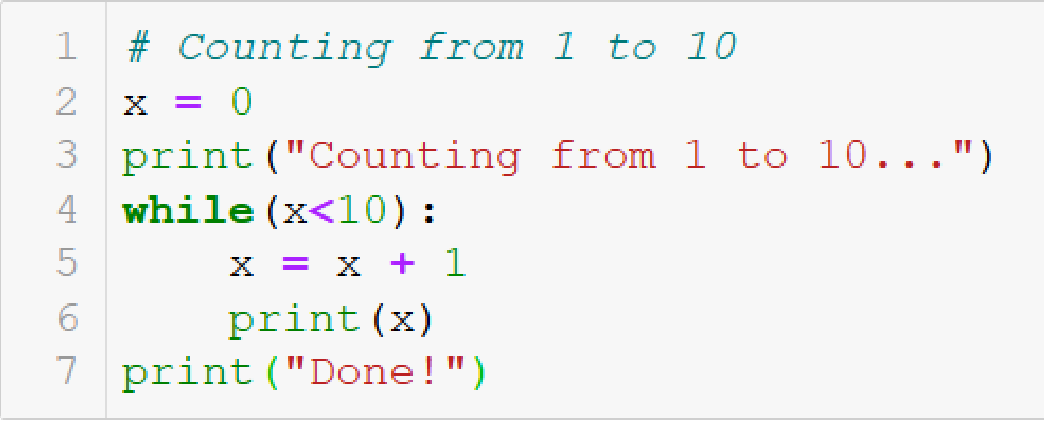
Matt’s Great Advice #8: Avoid the infinite loops, if possible.

Relying on an **infinite** **while** loop with a **break** is **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.



```
# 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!")
```



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

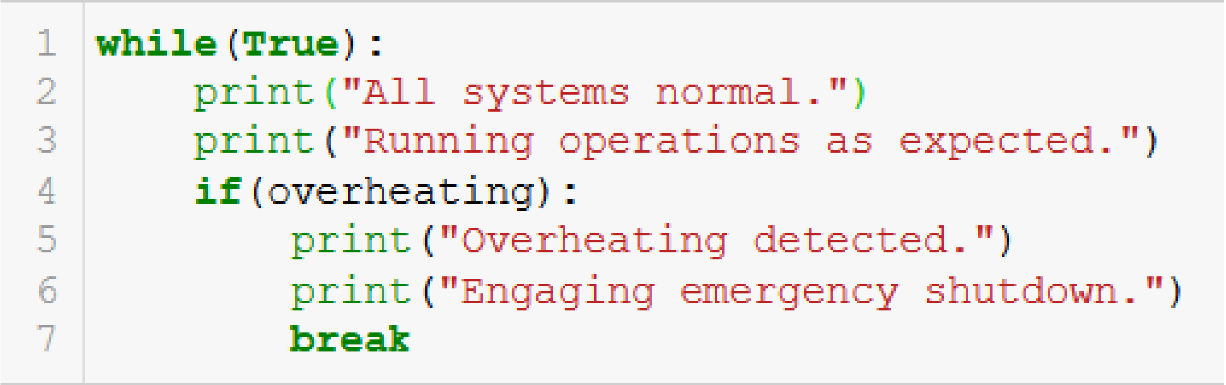
Matt’s Great advice #8

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**.



```
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 two activities.

Activity 1 – How many hits can you take.ipynb

Activity 1 - How many hits can you take

Your main character currently has a number of lifepoints, stored in **lifepoints_number**.

Your mentor gives you the following challenge: he will hit you, for a given number of times $\boldsymbol{n}$.

- The **first hit** will make you **lose one lifepoint**,
- the **second**, **two lifepoints**,
- the **third**, **three lifepoints**,
- and **so on**.
- **If** you take too many hits and **your lifepoints fall at or below zero, you fail the challenge.**
- Assuming you survive $\boldsymbol{n}$ hits, your mentor will give you $\boldsymbol{n^2}$ coins.

Write a function, named **maximal_coins_number()**, which

- **receives** your current number of **lifepoints**, as the variable **lifepoints_number**,
- and **returns** the **maximal number of coins** you can hope to obtain from the challenge,
- as well the **number of lifepoints** that will be **remaining after taking this maximal number of hits**.

Activity 2 – Guess the number game v2

Remember the guess the number game in W3S1, Activity 1? Back then, we had defined a function **guess_the_number()**,

- which received a **hidden number that the user had to guess** (passed as input **hidden_number**),
- **asked the user to input a number**, via the **input()** method and would store it in a variable **guessed_number**,
- and based on the two numbers would **display two messages**, reading:
    - "You have found the hidden number: True/False."
    - "Your number in guessed_number is lower than the hidden number: True/False.“

Your task is to write a **second version** (v2!) of this function, called **guess_the_number_v2().**

Activity 2 – Guess the number game v2

This v2 function will have the following features, replacing the previous ones:

- The game will **keep on asking the user to input()** values, **until the right number is found**.
- It will **display the message** "Your have found the right number!", **once the user has found the right number.**
- When that happens, it **also displays** "It only took you ... tries!" with the blank filled with the number of times the user had to type a number via input().
- Once the number has been found, the function no longer asks the user for inputs and stops.
- **While the user has not found the right number**, the game will **display either**
    - "Your number is lower than the hidden number." (if the last number entered by the user is lower than the hidden number)
    - or "Your number is higher than the hidden number." (if the last number entered by the user is higher than the hidden number).

Conclusion

- The if statement
- The elif statement
- The else statement
- Dead code and code structure
- Nested ifs
- While statements
- Infinite looks 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!