APS106

UNIVERSITY OF TORONTO

while loops.

Week 4 | Lecture 1 (4.1)

if nothing else, write #cleancode

This Week's Content

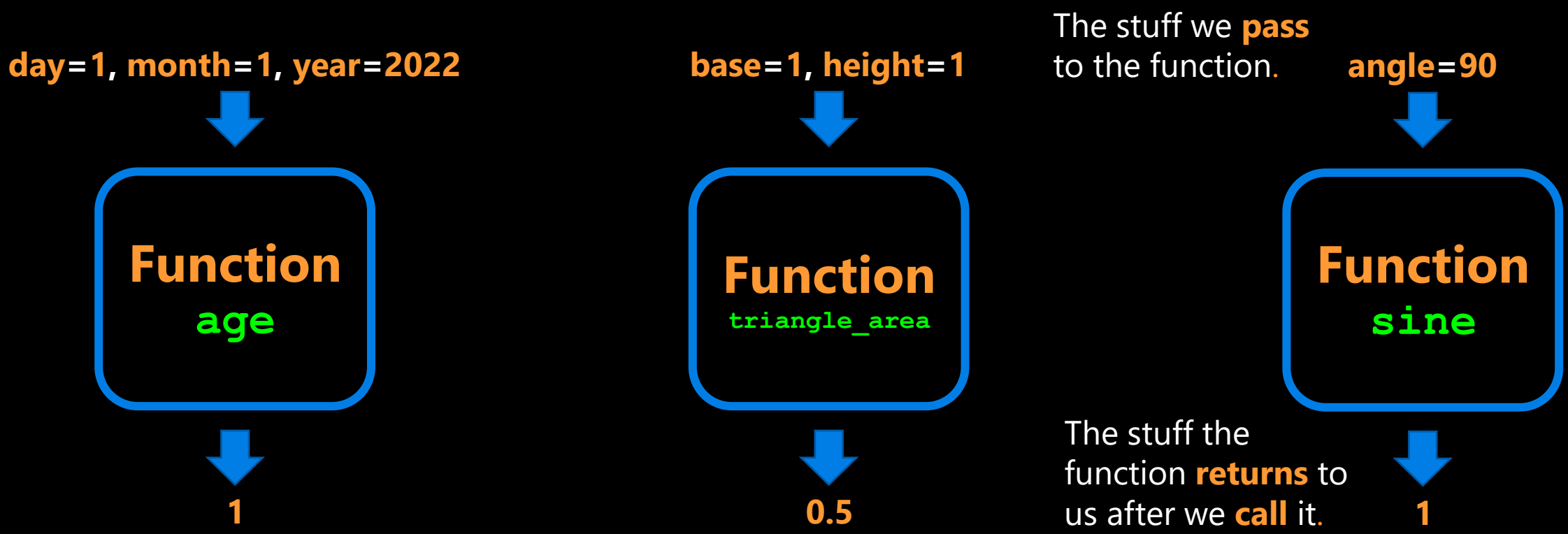
- **Lecture 4.1**
  - **function review, while loops**
  - **Reading: Chapter 9**
- **Lecture 4.2**
  - More while loops
  - Reading: Chapter 9
- **Lecture 4.3**
  - Midterm review

`function` confusion

- Review.
- `parameters` and `arguments`.
- `print` and `return`.
- When is a function done?

function, what are they?

- A function is best explained as a self-contained piece of code that has inputs and an output.



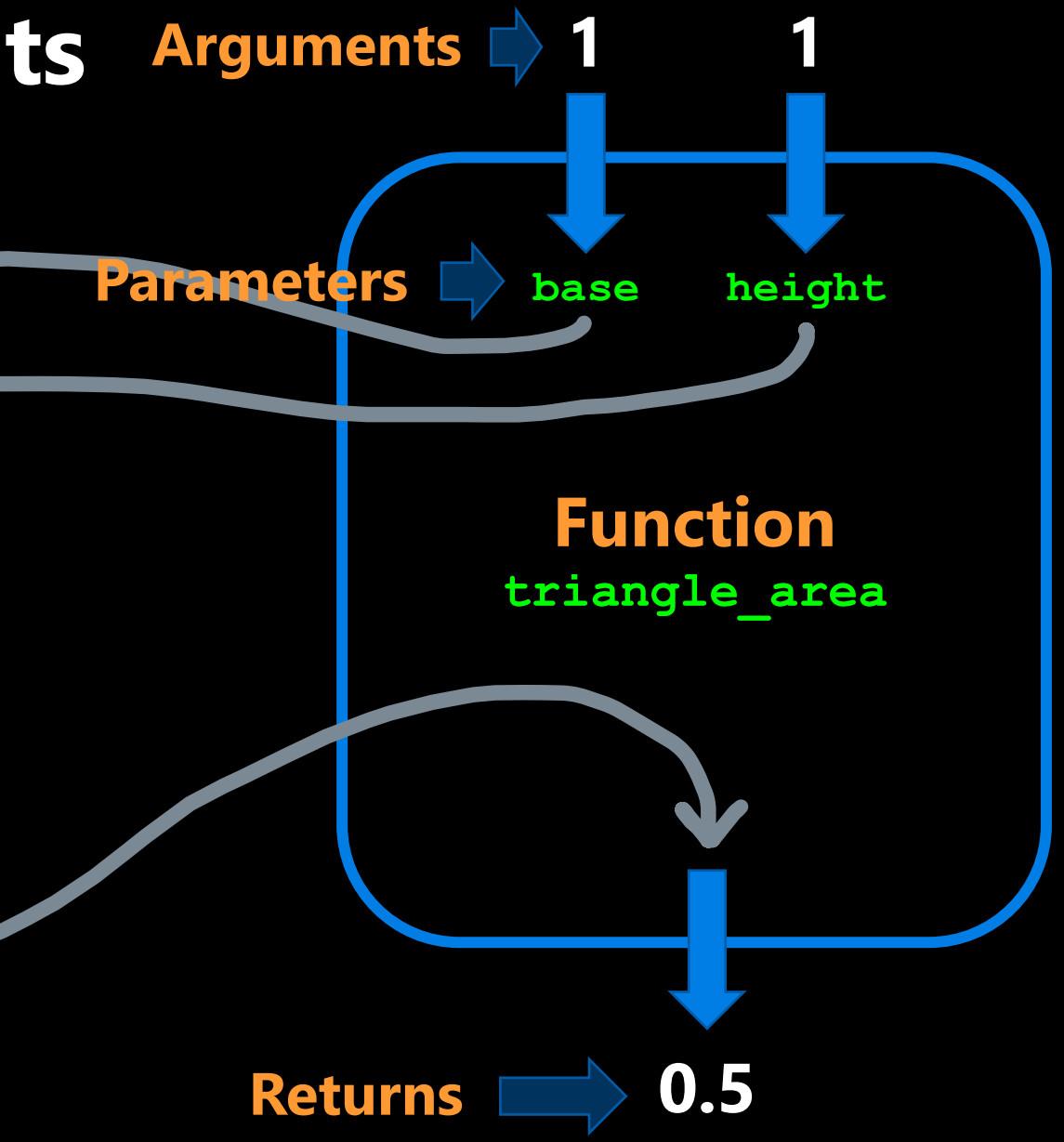
function, what are they?

- Let's look at a real example of using function.

Open your notebook

Click Link:
1. Function use cases

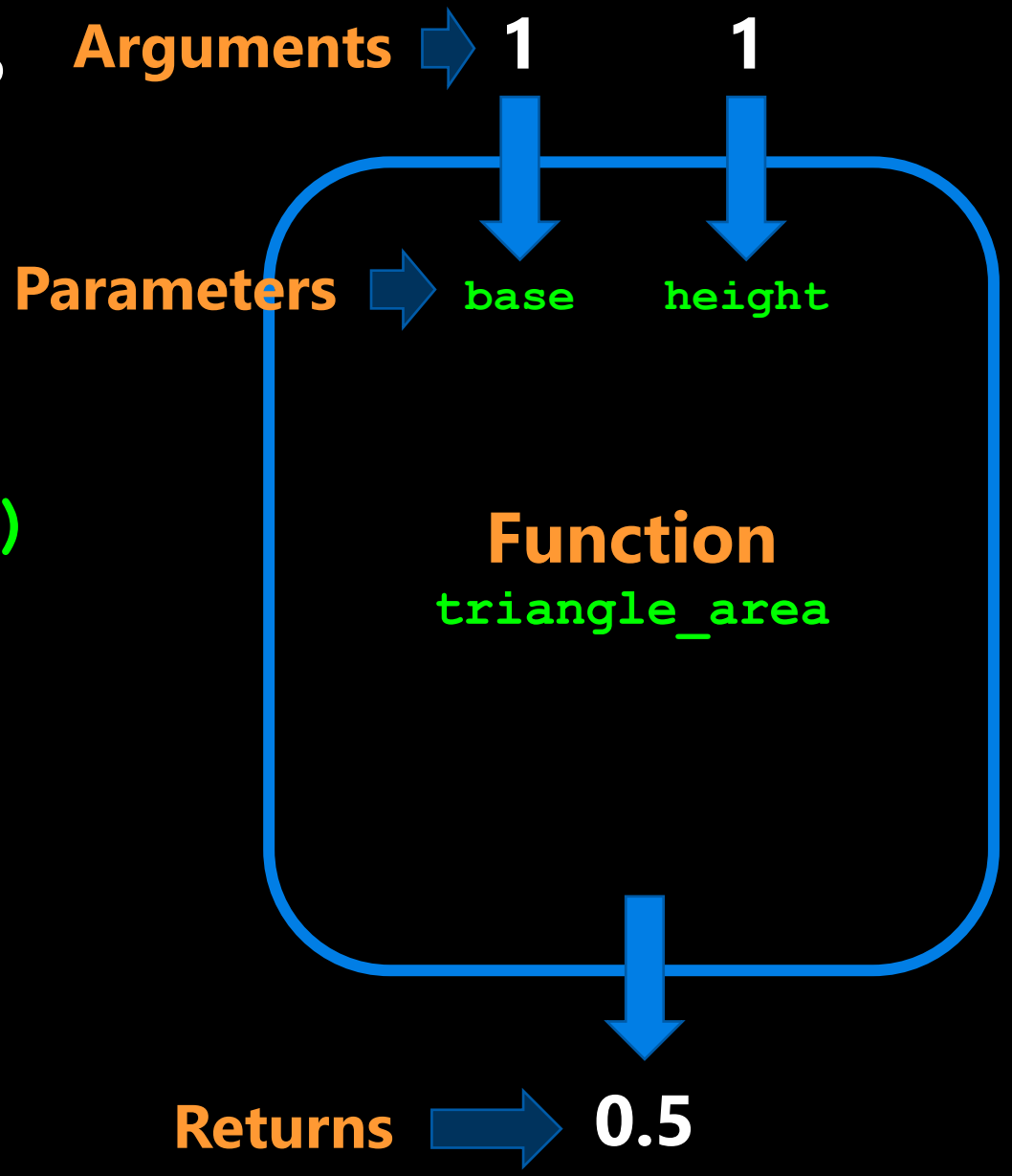
parameters & arguments




```
def triangle_area(base, height):
    """
    (number, number) -> number
    """
    area = 0.5 * base * height
    return area
```

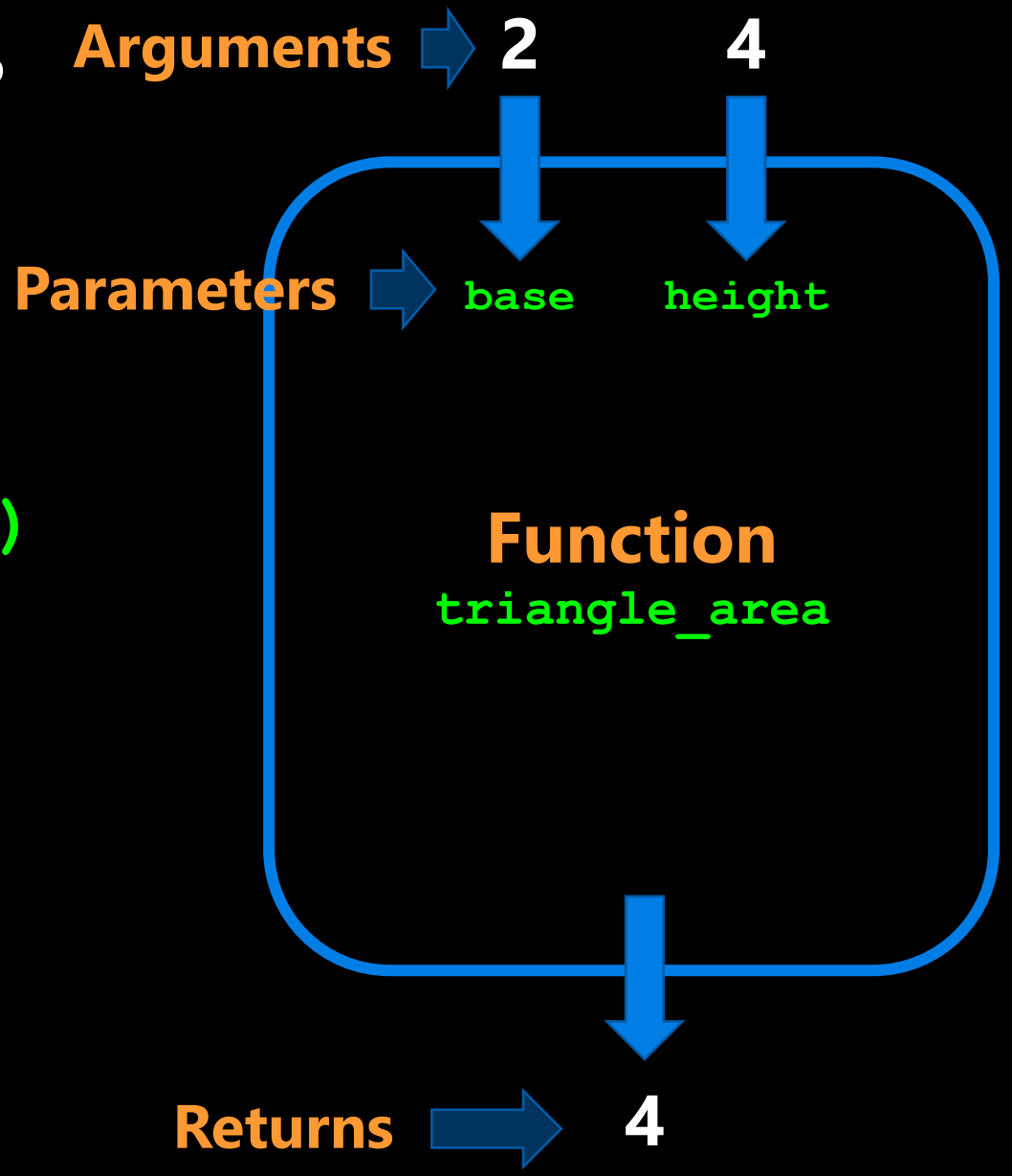
parameters & arguments

```
>>> area = triangle_area(1, 1)
>>> print(area)
0.5
```



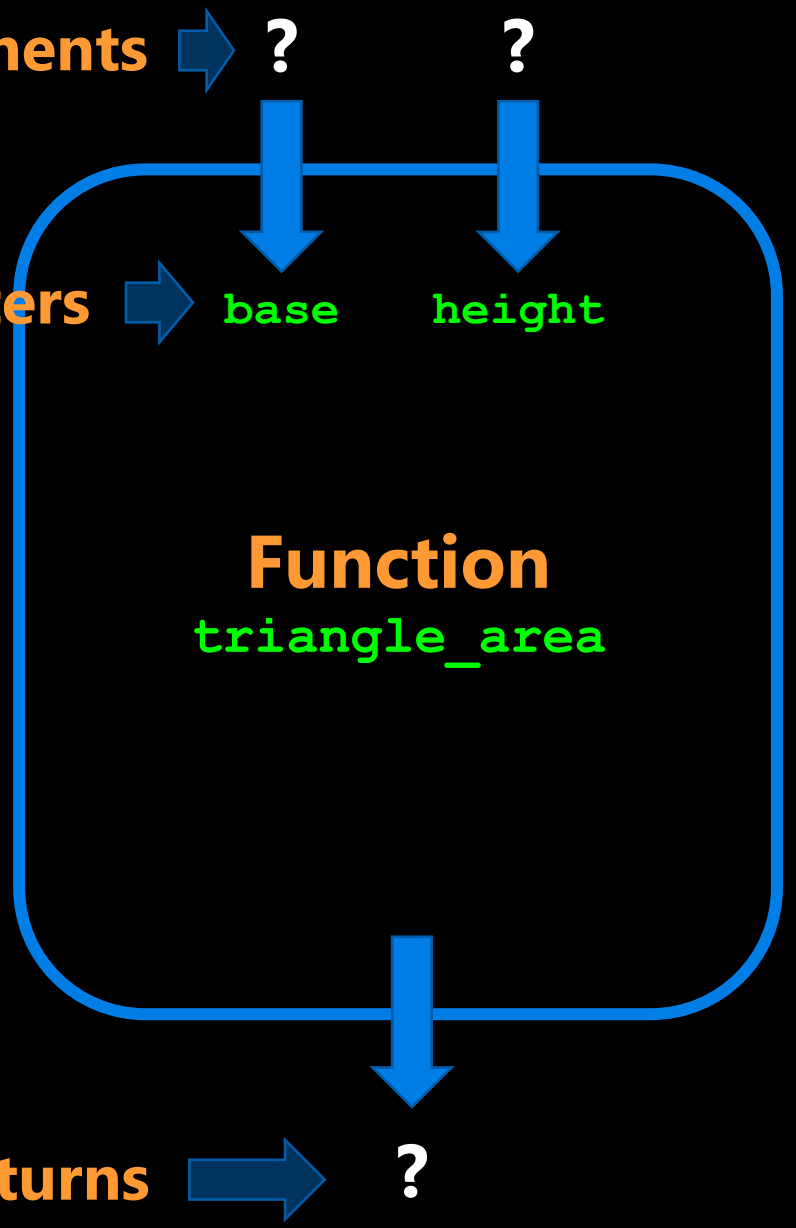
parameters & arguments

```
>>> area = triangle_area(2, 4)
>>> print(area)
4
```



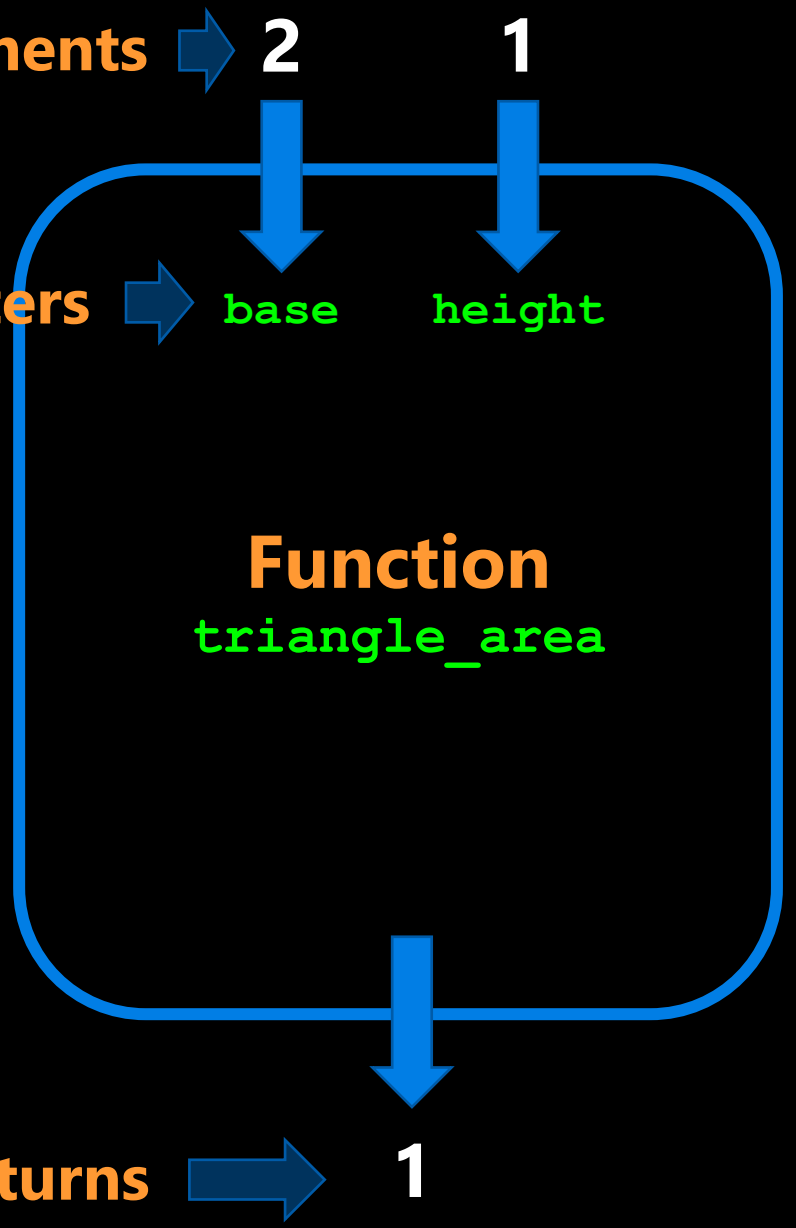
parameters & arguments

```
>>> area = triangle_area(1+1, 2/2)
>>> print(area)
?
```



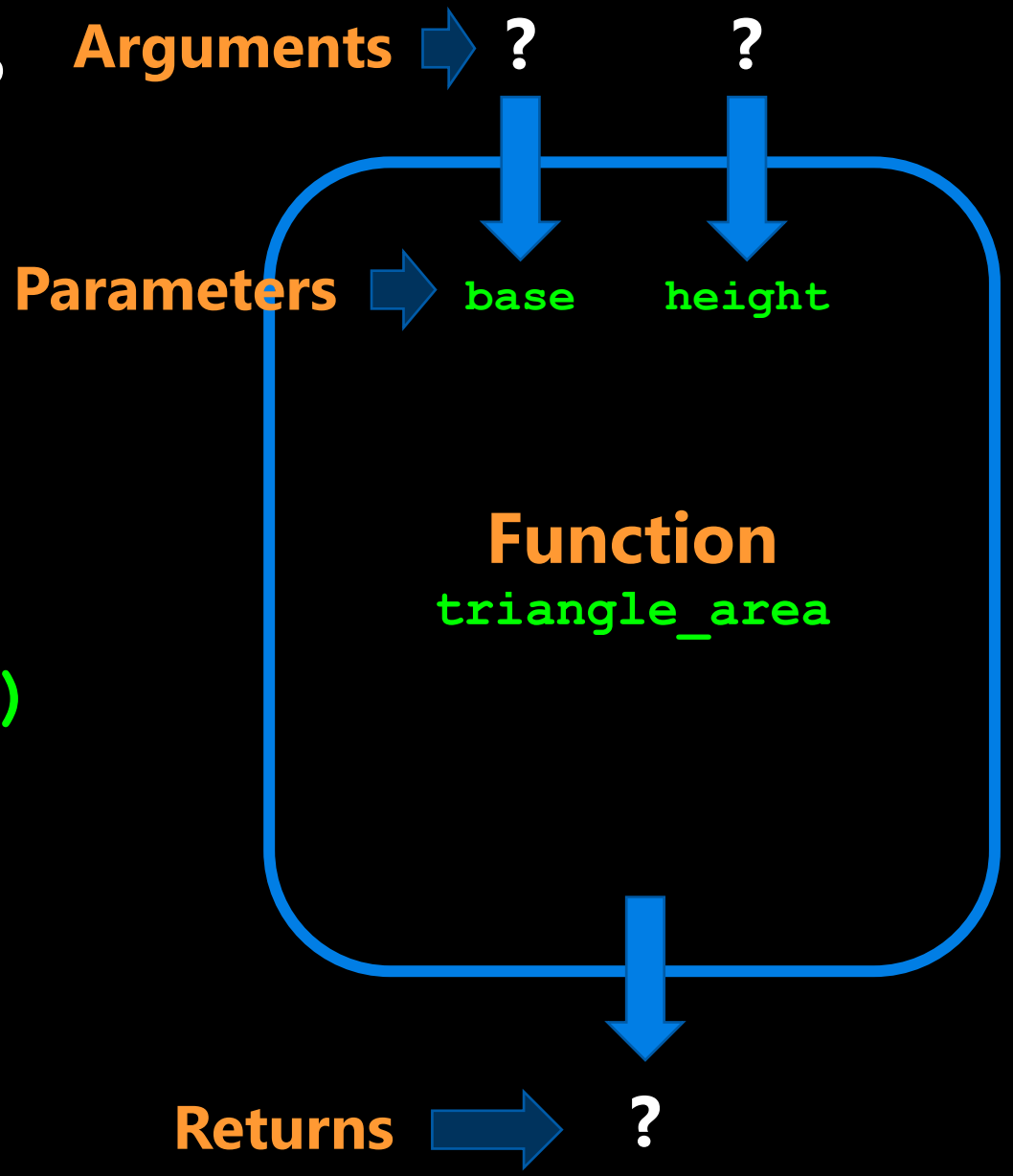
parameters & arguments

```
>>> area = triangle_area(1+1, 2/2)
>>> print(area)
1
```



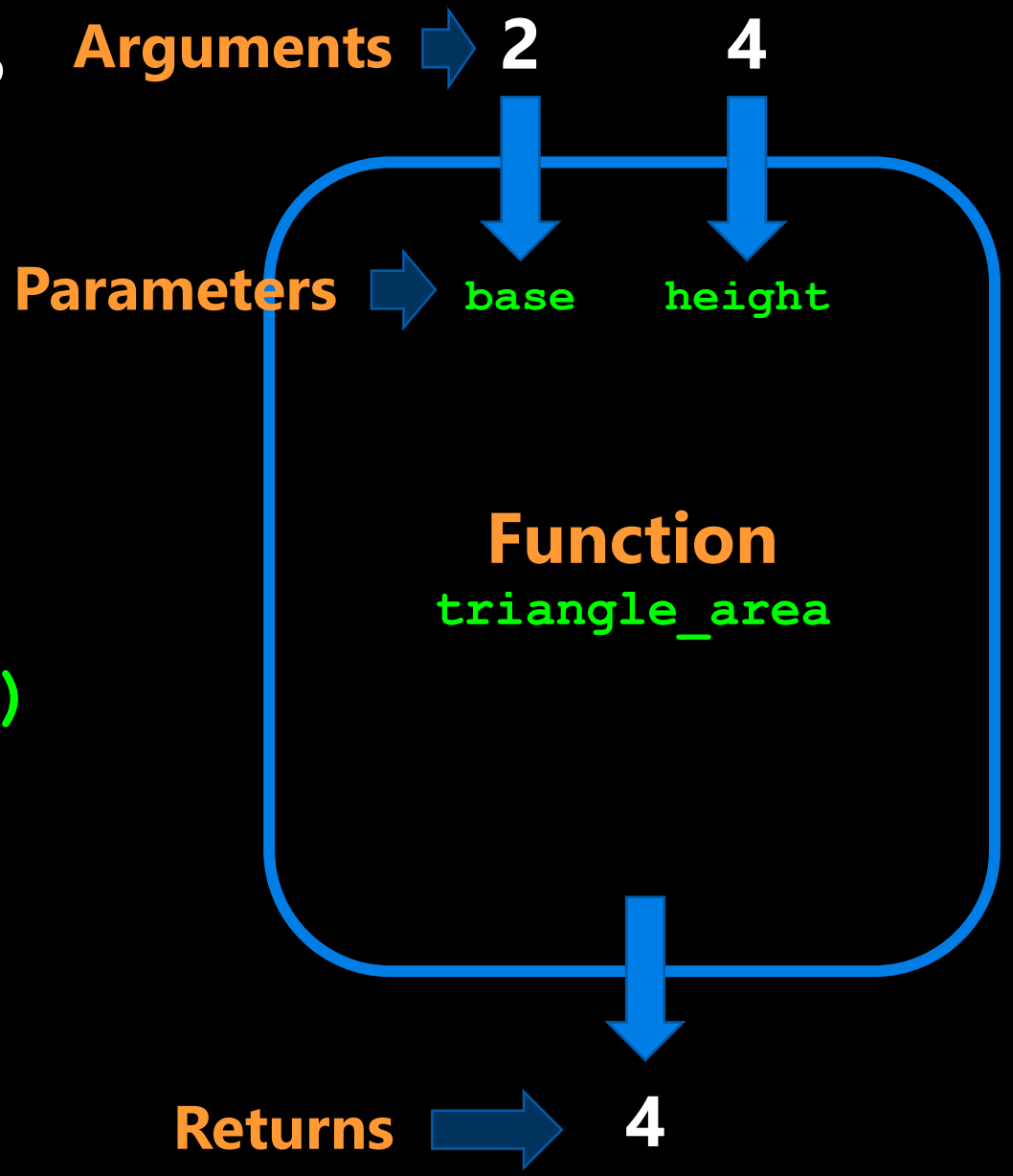
parameters & arguments

```
>>> x = 2
>>> y = 4
>>> area = triangle_area(x, y)
>>> print(area)
?
```



parameters & arguments

```
>>> x = 2
>>> y = 4
>>> area = triangle_area(x, y)
>>> print(area)
4
```

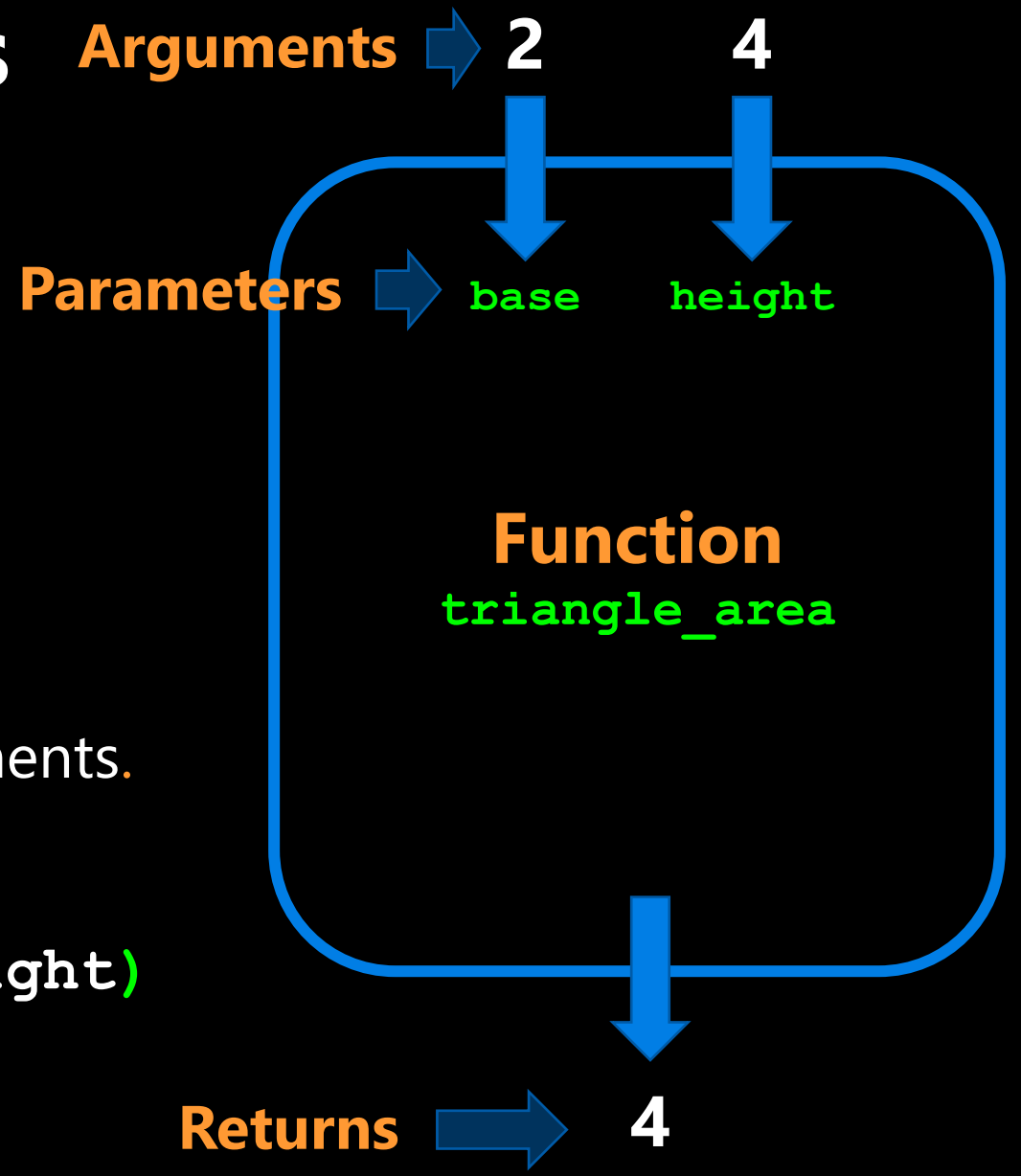


parameters & arguments

```
>>> x = 2
>>> y = 4
>>> area = triangle_area(x, y)
>>> print(area)
4
```

Same arguments.

```
>>> base = 2
>>> height = 4
>>> area = triangle_area(base, height)
>>> print(area)
4
```



parameters & arguments

- Let's look at some examples.

Open your notebook

Click Link:
2. Parameters & Arguments

`print` v.s. `return`

- The difference between print and return is a point of confusion year after year.
- So, let's be proactive and address this.

`print`

- **Use cases**
- **Debugging**.
- Displaying messages to users.

`return`

- **Use cases**
- Used to end the execution of the function call and "return" the result.

print

```
def square(x):
    output = x * x
    print(output)
```

```
>>> square(2)
4
```

return

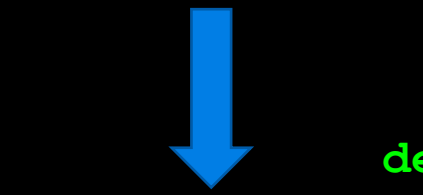
```
def square(x):
    output = x * x
    return output
```

```
>>> square(2)
4
```

print

The stuff we **pass** to the function.

Arguments: 2



```
def square(x):
    output = x * x
    print(output)
```

Function
`square`

The stuff the function **returns** to us after we **call** it.

Returns: `None`

return

The stuff we **pass** to the function.

Arguments: 2

```
def square(x):
    output = x * x
    return output
```

Function
`square`

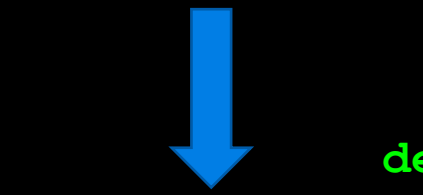
The stuff the function **returns** to us after we **call** it.

Returns: `4`

print

The stuff we **pass** to the function.

Arguments: `2`



```
def square(x):
    output = x * x
    print(output)
```

Function `square`

The stuff the function **returns** to us after we **call** it.

Returns: `None`

Standard Out.

`4`

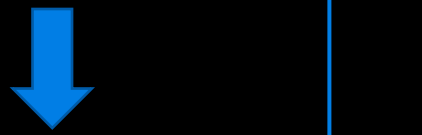
return

The stuff we **pass** to the function.

Arguments: `2`

```
def square(x):
    output = x * x
    return output
```

Function `square`



The stuff the function **returns** to us after we **call** it.

Returns: `4`

Standard Out.

`print` v.s. `return`

- Let's look at some examples.

Open your notebook

Click Link:
3. `print` v.s. `return`

When is a function done?

- A function is done executing if one of the following things occurs:

1. All the indented code finishes running.

2. A return statement is encountered.

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```

```
>>> out = func(2)
>>> print(out)
?
```

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```

```
>>> out = func(2)
>>> print(out)
?
```

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```
end.

```
>>> out = func(2)
>>> print(out)
14
```

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```

end.

```
>>> out = func(2)
>>> print(out)
14
```

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```
end.

```
>>> out = func(2)
>>> print(out)
14
```

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```
end.
(end of indented code)

```
>>> out = func(2)
>>> print(out)
None
```

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
?
```

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```
end.

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```
end. (end of indented code)

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```

```
>>> out = func(2)
>>> print(out)
14
```

```
>>> out = func(2)
>>> print(out)
None
```

```
>>> out = func(2)
>>> print(out)
?
```

When is a function done?

```
def func(x):
    output = x * x
    output += 10
    return output
```
end.

```
>>> out = func(2)
>>> print(out)
14
```

```
def func(x):
    output = x * x
    output += 10
    output /= 2
```
end. (end of indented code)

```
>>> out = func(2)
>>> print(out)
None
```

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
```
end.

```
>>> out = func(2)
>>> print(out)
4
```

When is a function done?

- Let's look at some examples.

Open your notebook

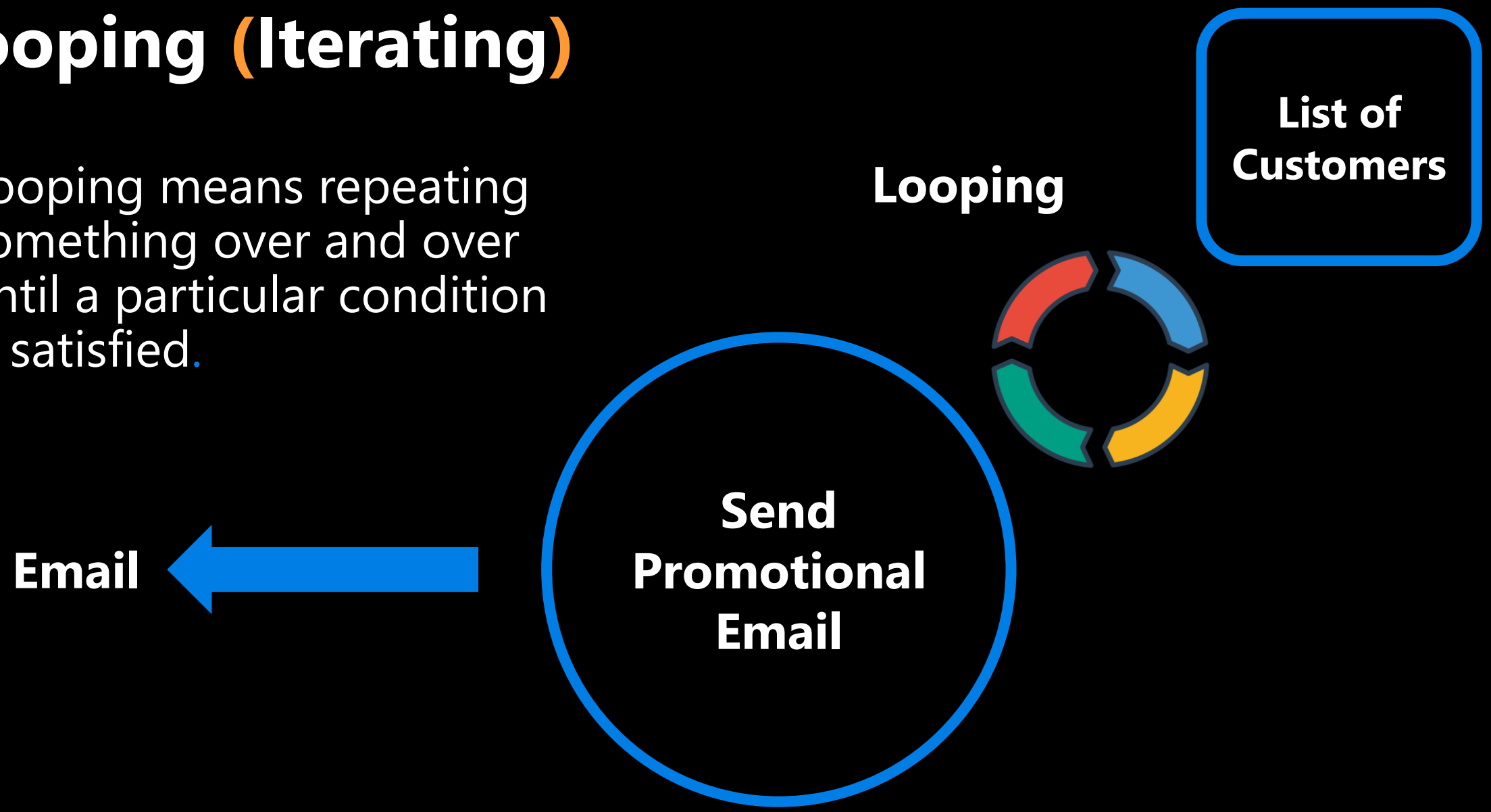
Click Link:
4. When is a function done?

Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.
- Looping (aka iteration) is the second key control structure in programming (if-statements/branching was the first).

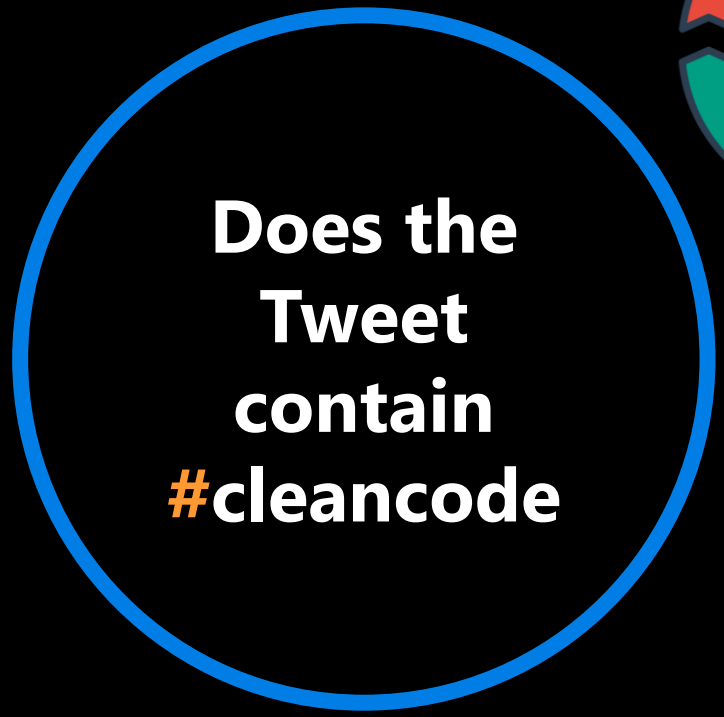
Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.



Looping (Iterating)

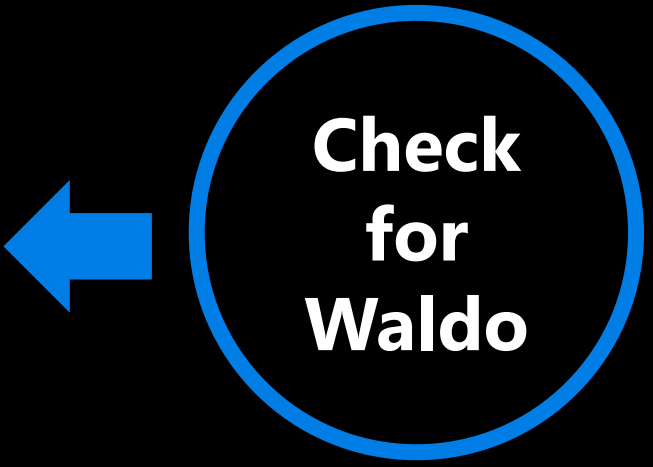
- Looping means repeating something over and over until a particular condition is satisfied.

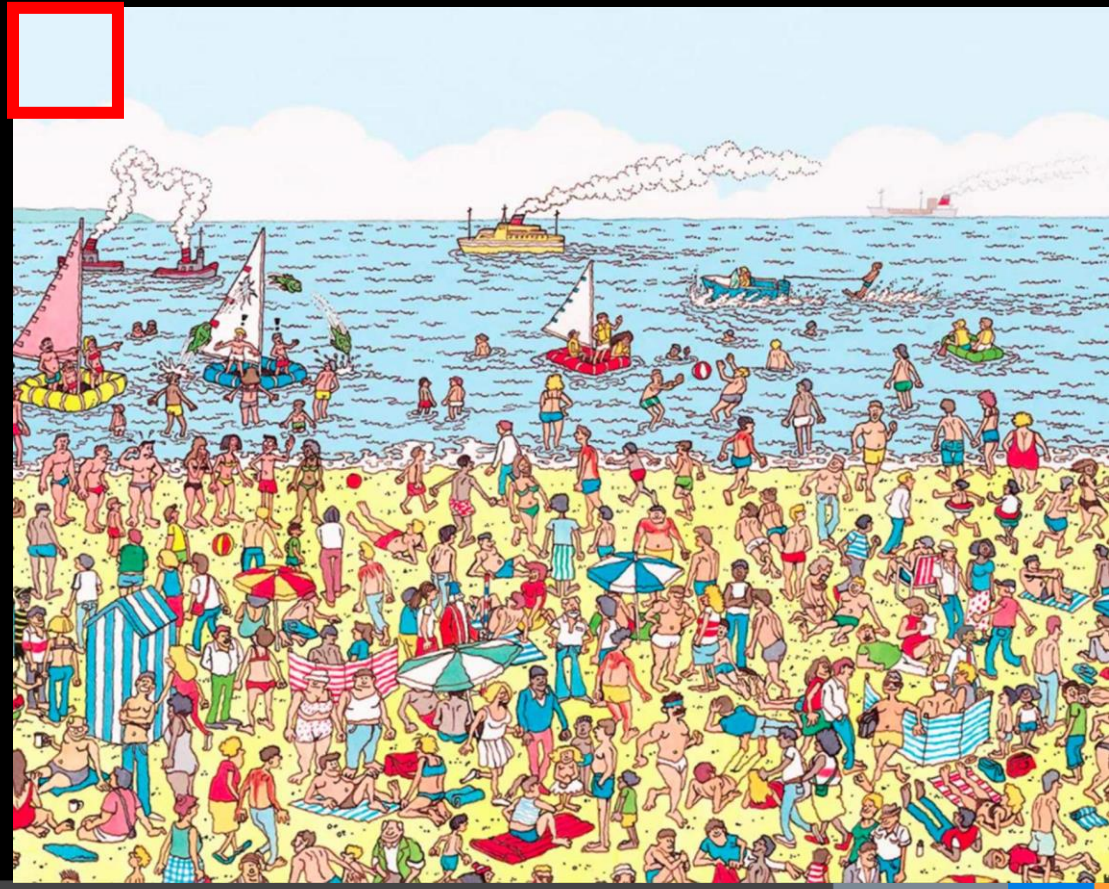


Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Looping

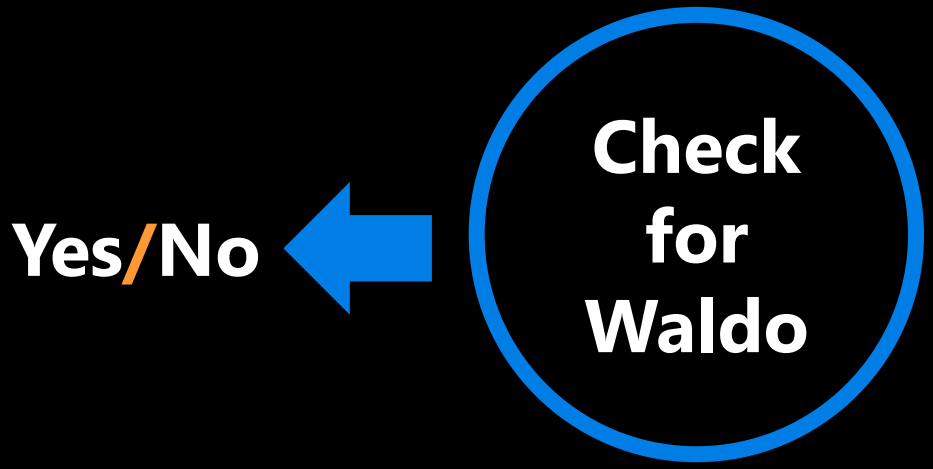


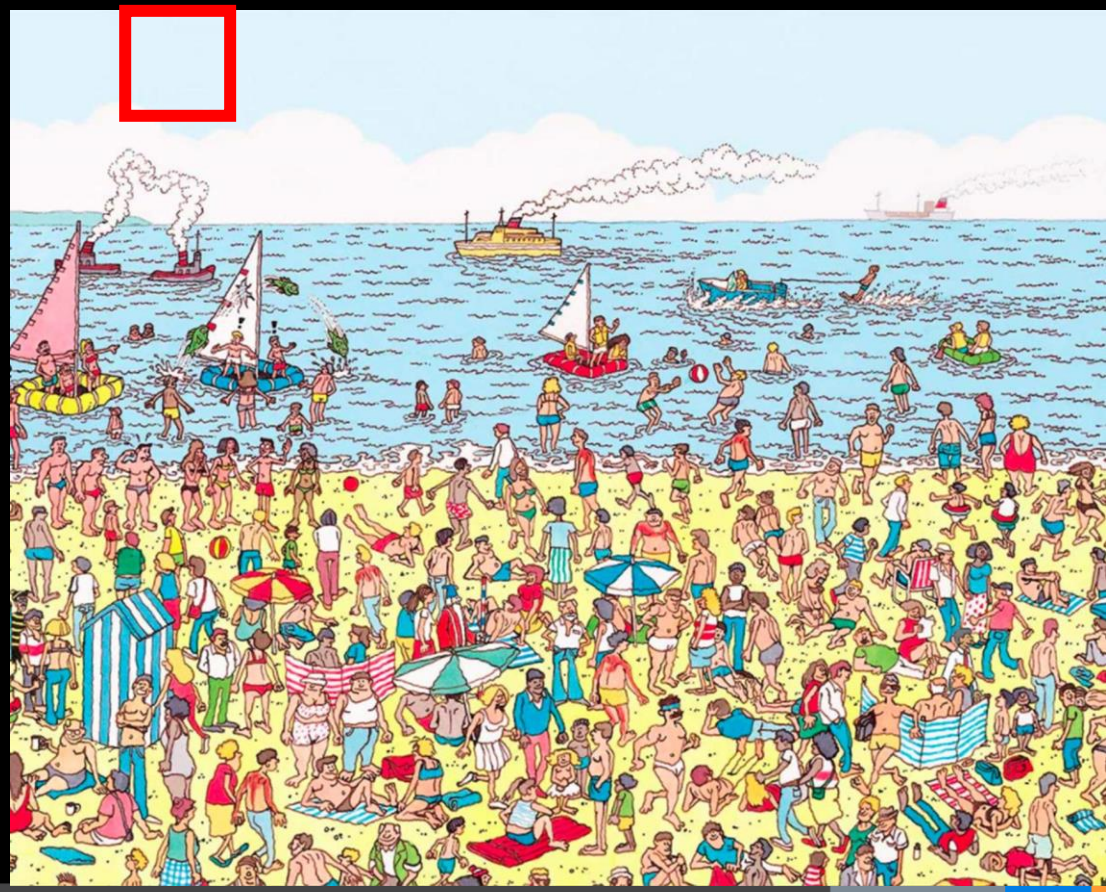


Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Looping

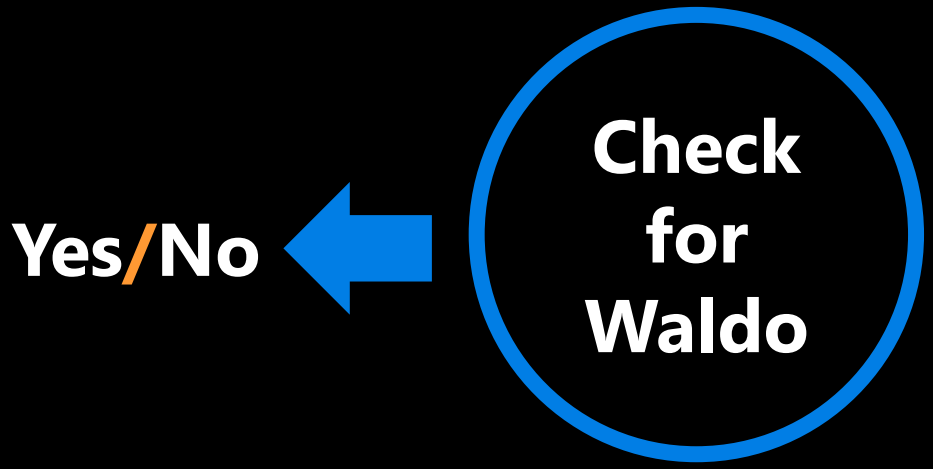


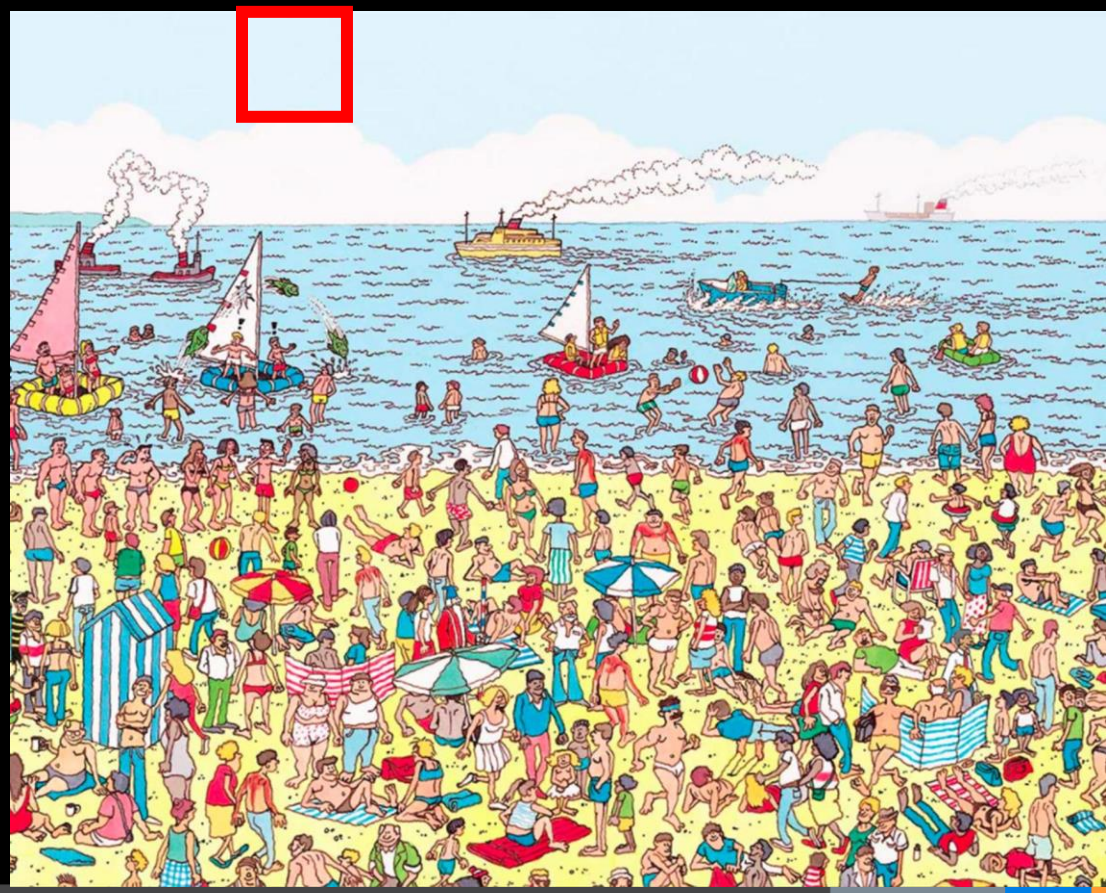


Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Looping

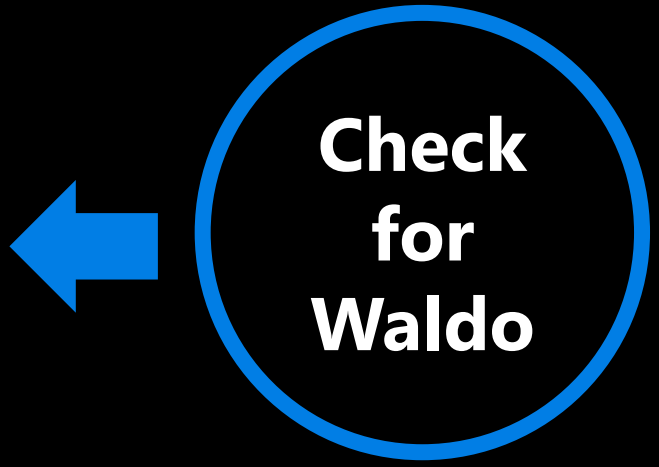


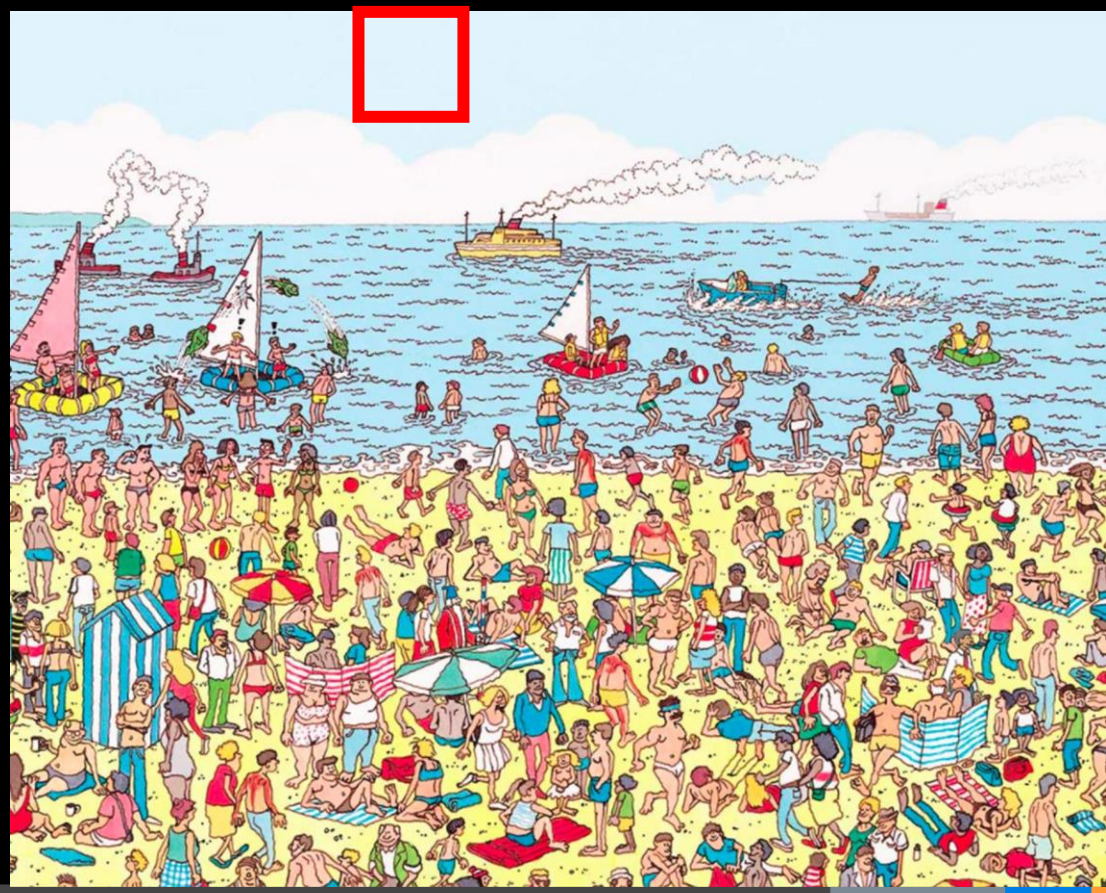


Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Looping

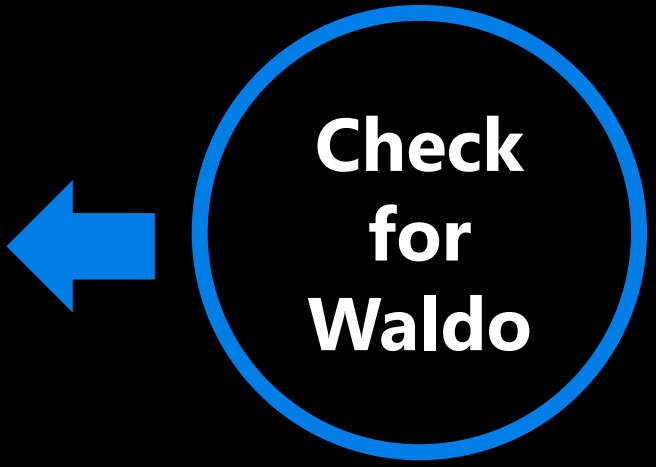


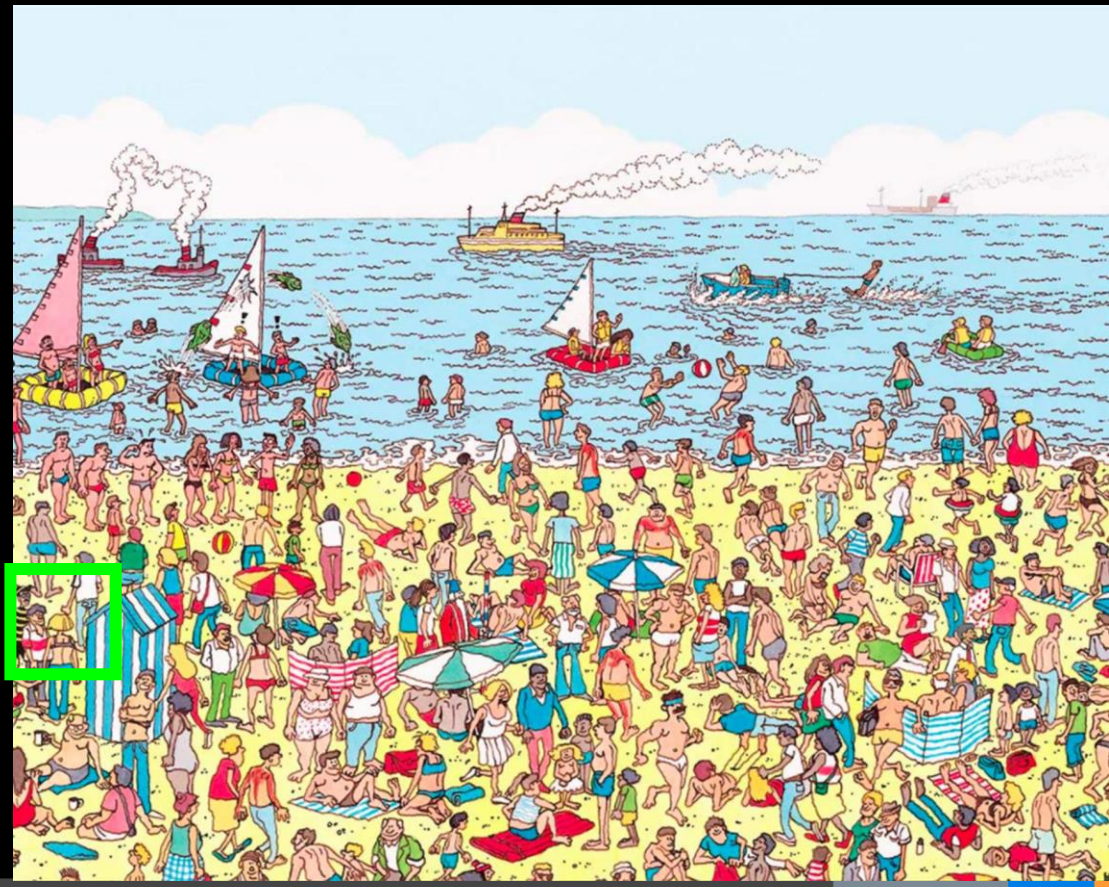


Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Looping





While Loops

- Sometimes we need to keep looping as long as some condition is **True**, and stop when it becomes **False**.
- Let's say you want to ask the user a question.
  - "Do you think the Toronto Maple Leafs will win the Stanley Cup in your lifetime?"
- If the user answers 'y', print out "You are going to live for a very long time." If the user answers 'n', print out "Well, sometimes miracles happen."

Open your notebook

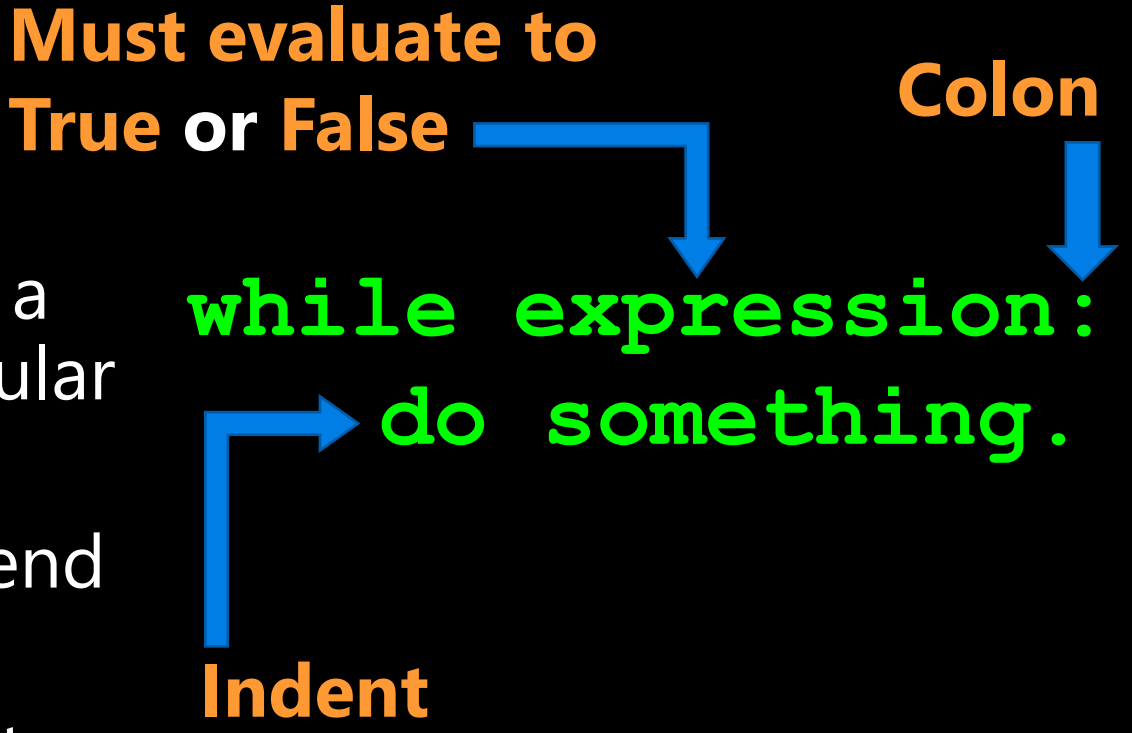
Click Link:
5. Asking the User a Question

While Loops

- Our code kinda worked but if the user makes a typo, they can't participate in the questionnaire.
- The general solution is to loop: to execute the same lines of code more than once. This is also called iteration.
- We're going to talk about one loop construct today: the while-loop where you loop while some boolean expression is True.

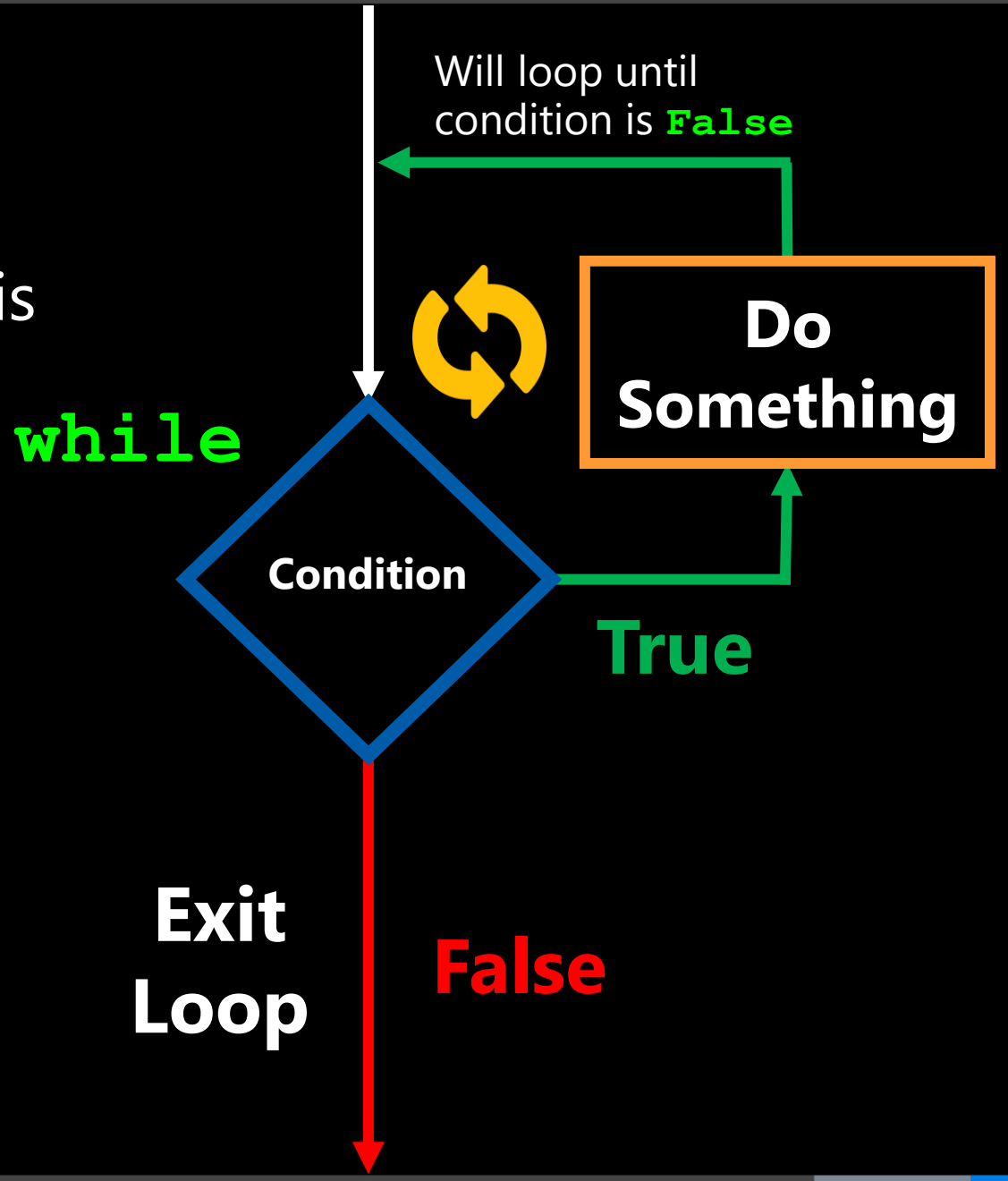
While Loops

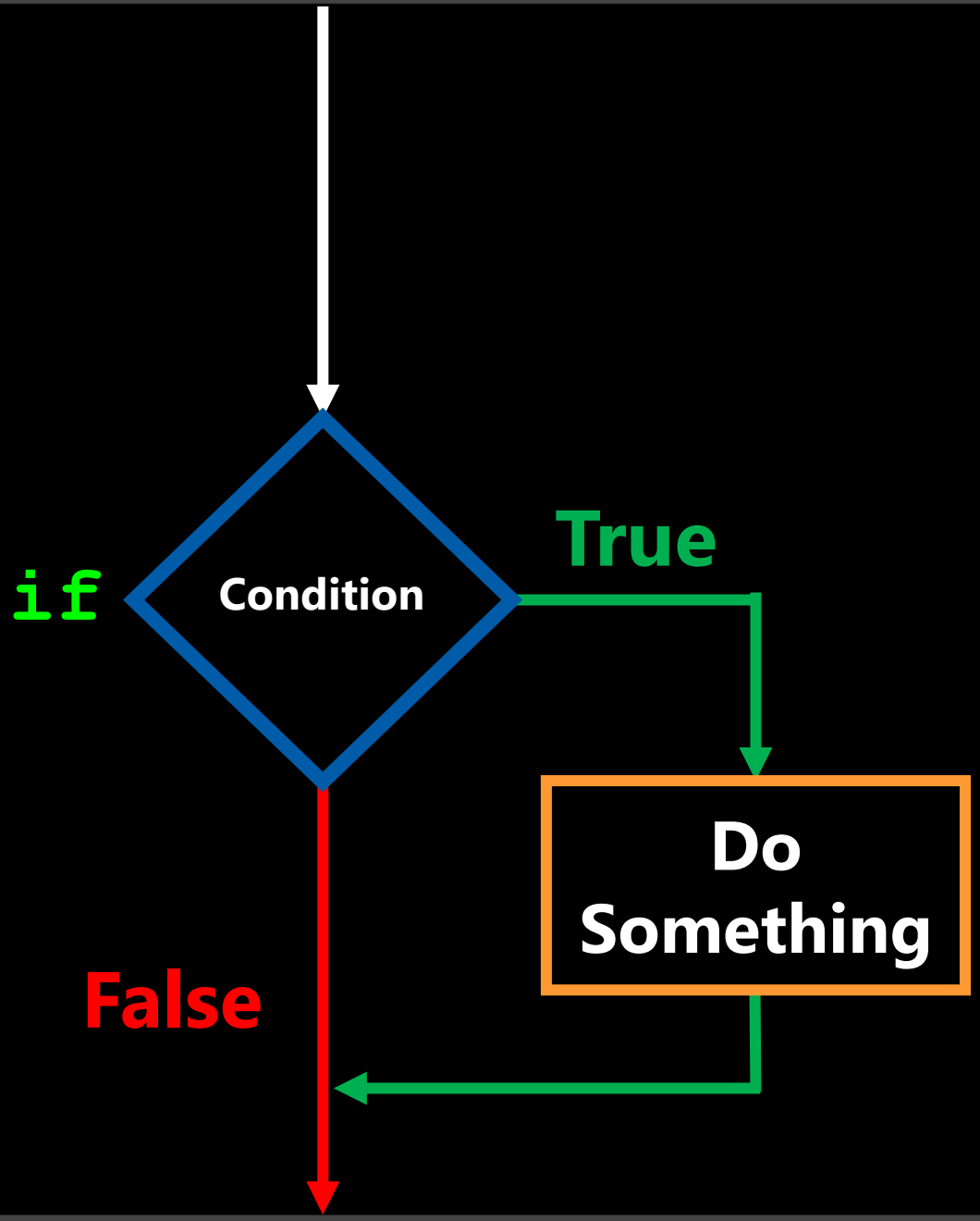
- The **while loop** keeps executing a piece of code as long as a particular condition is **True**.
- There must be a colon (:) at the end of the while statement.
- The action to be performed must be indented.

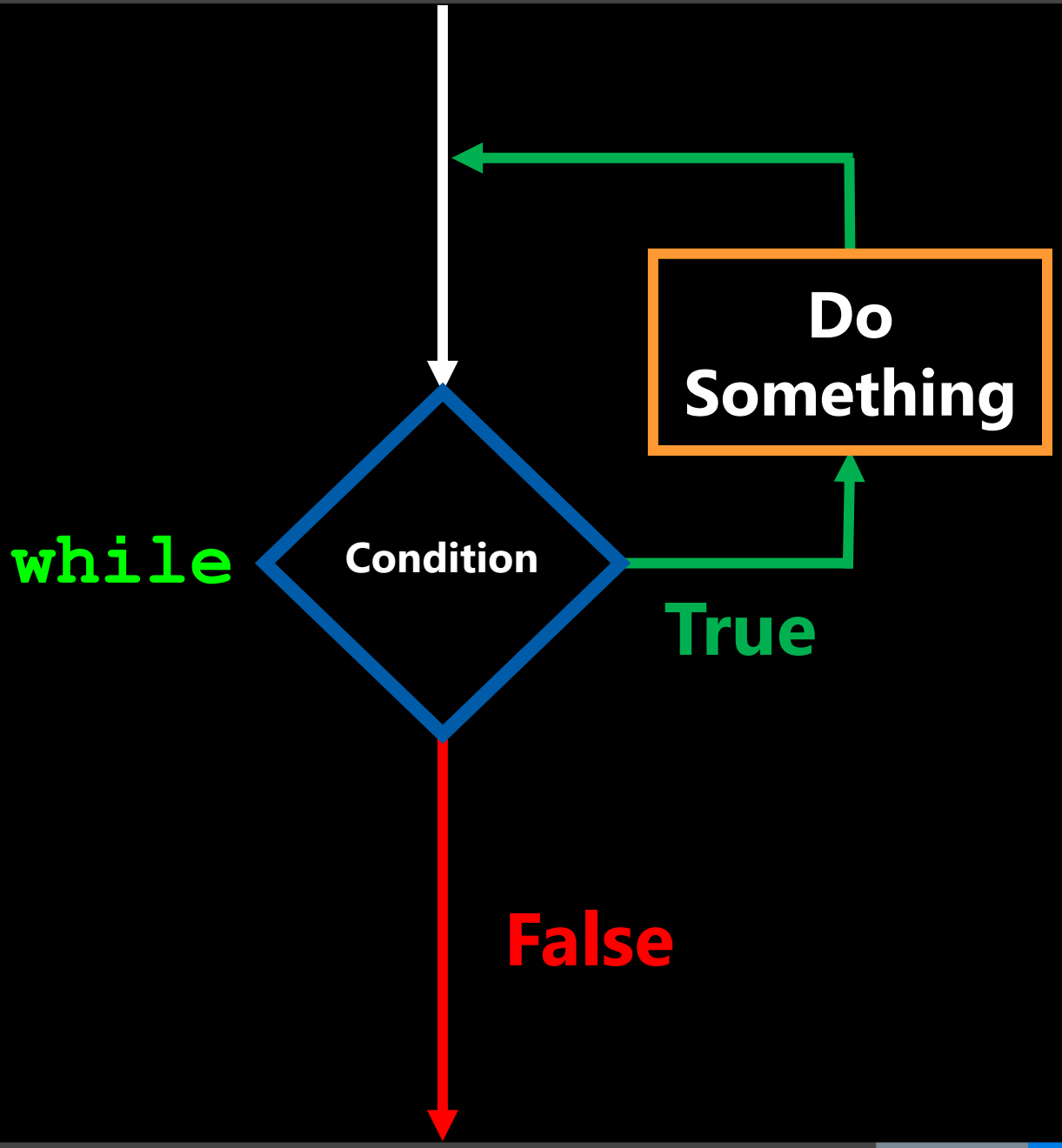


While Loops

- The condition that gets evaluated is just a boolean expression.
- In particular it can include:
  - Something that evaluates to **True** or **False**.
  - logical operators (**and**, **or**, **not**)
  - comparison operators
  - function calls
- … really anything that evaluates to **True** or **False**.

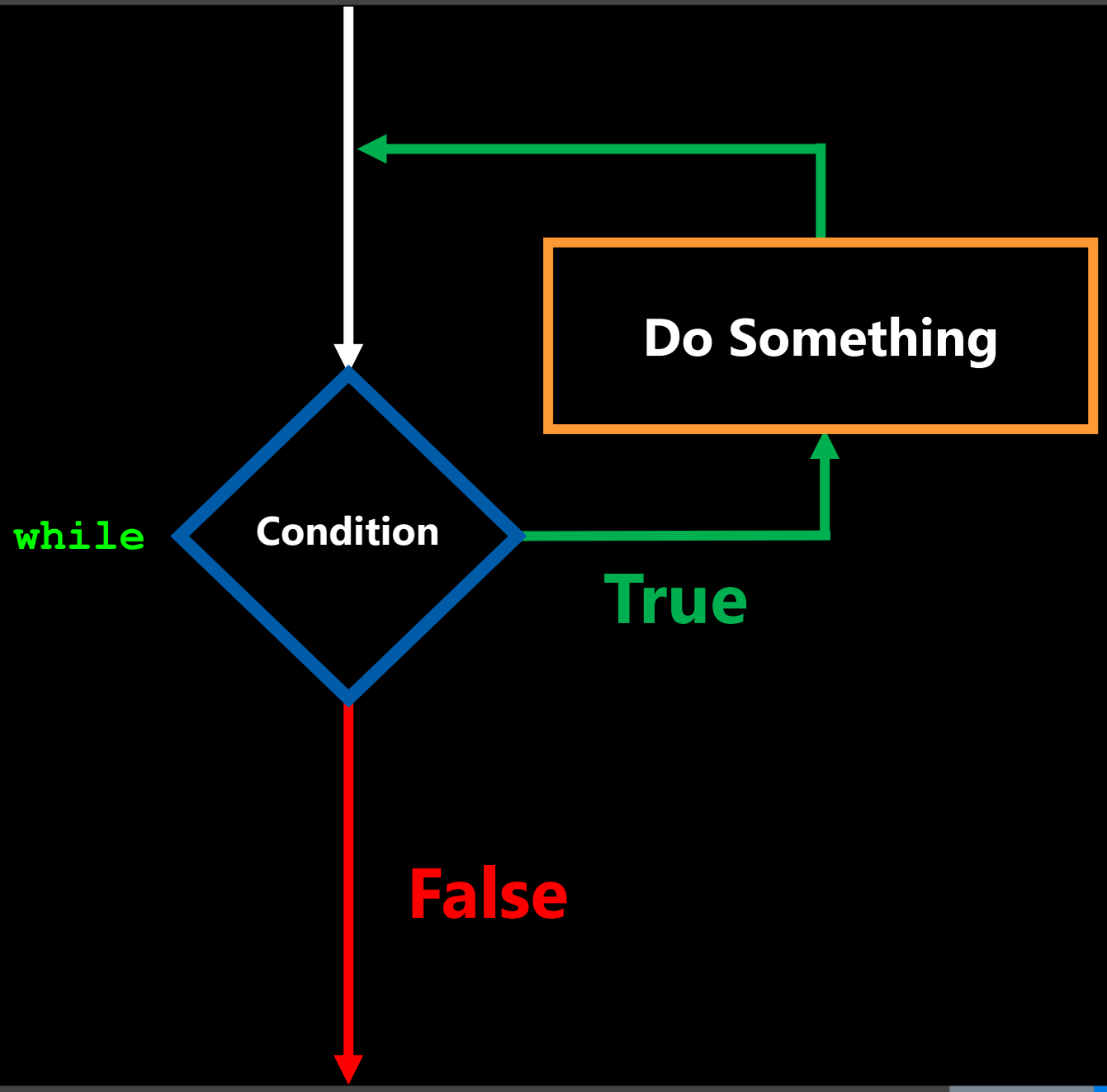



if
Condition
True
Do Something
False


while
Condition
Do Something
True
False

While Loops

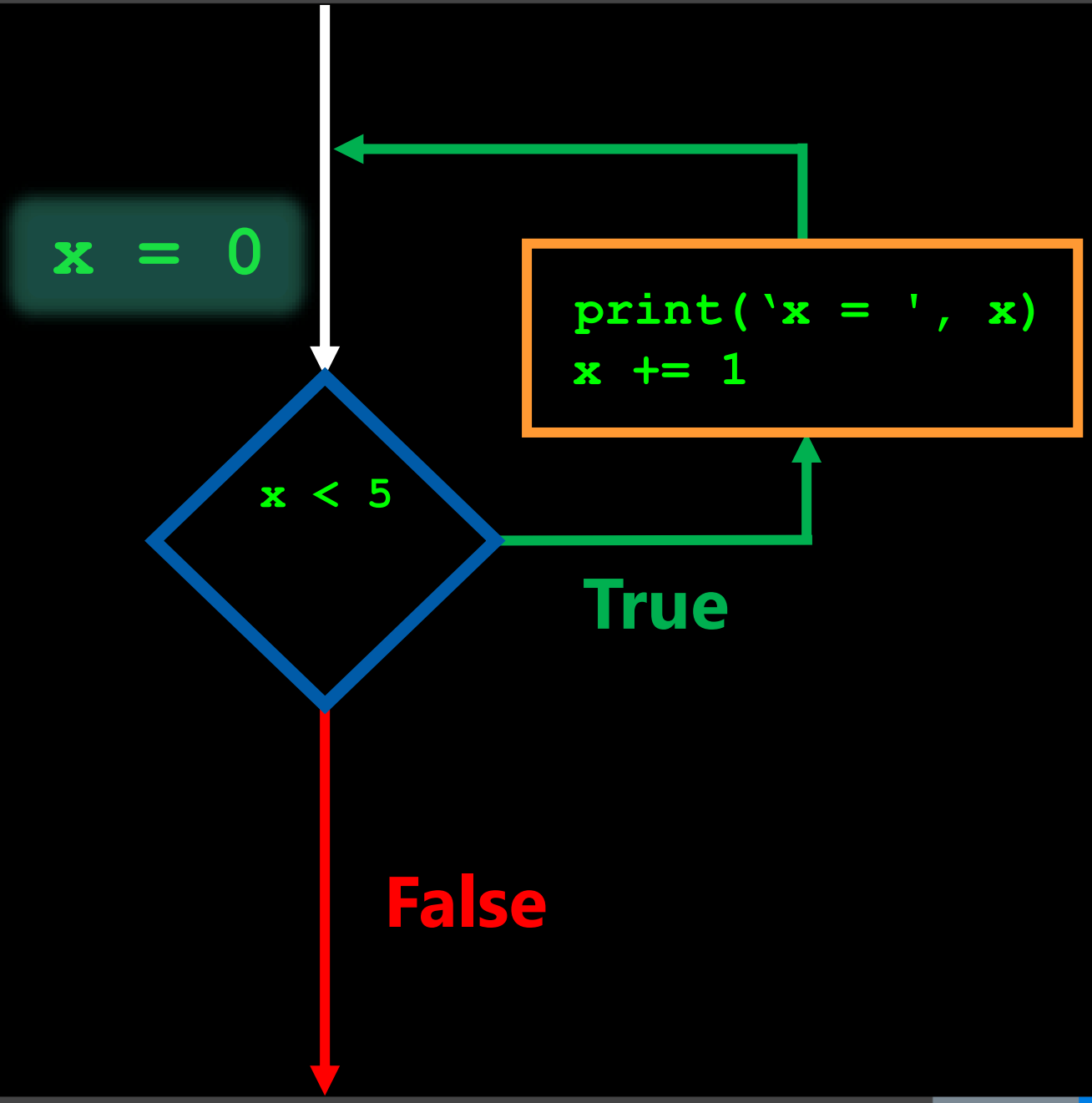
```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```



While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

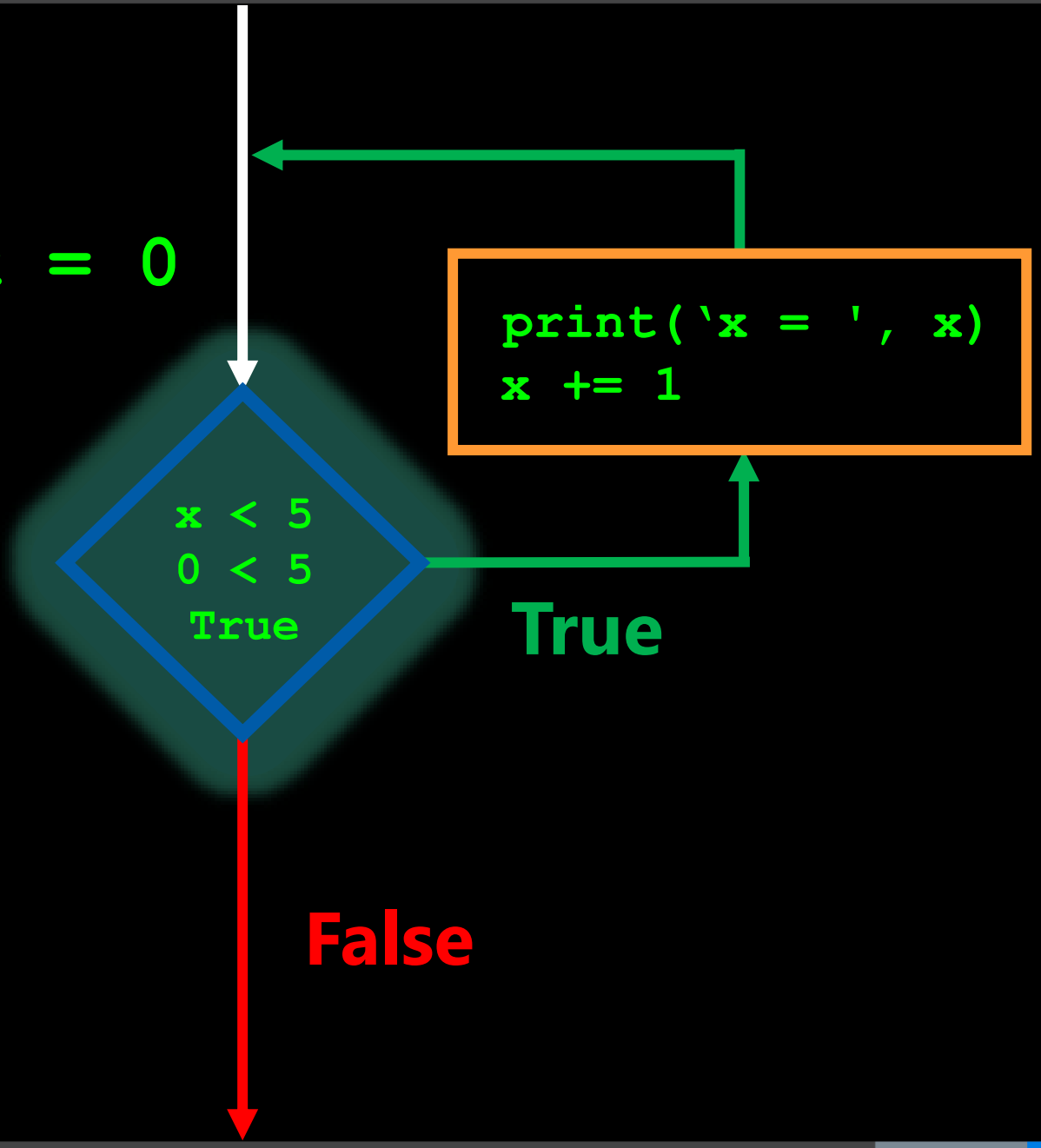
Standard Out.



While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

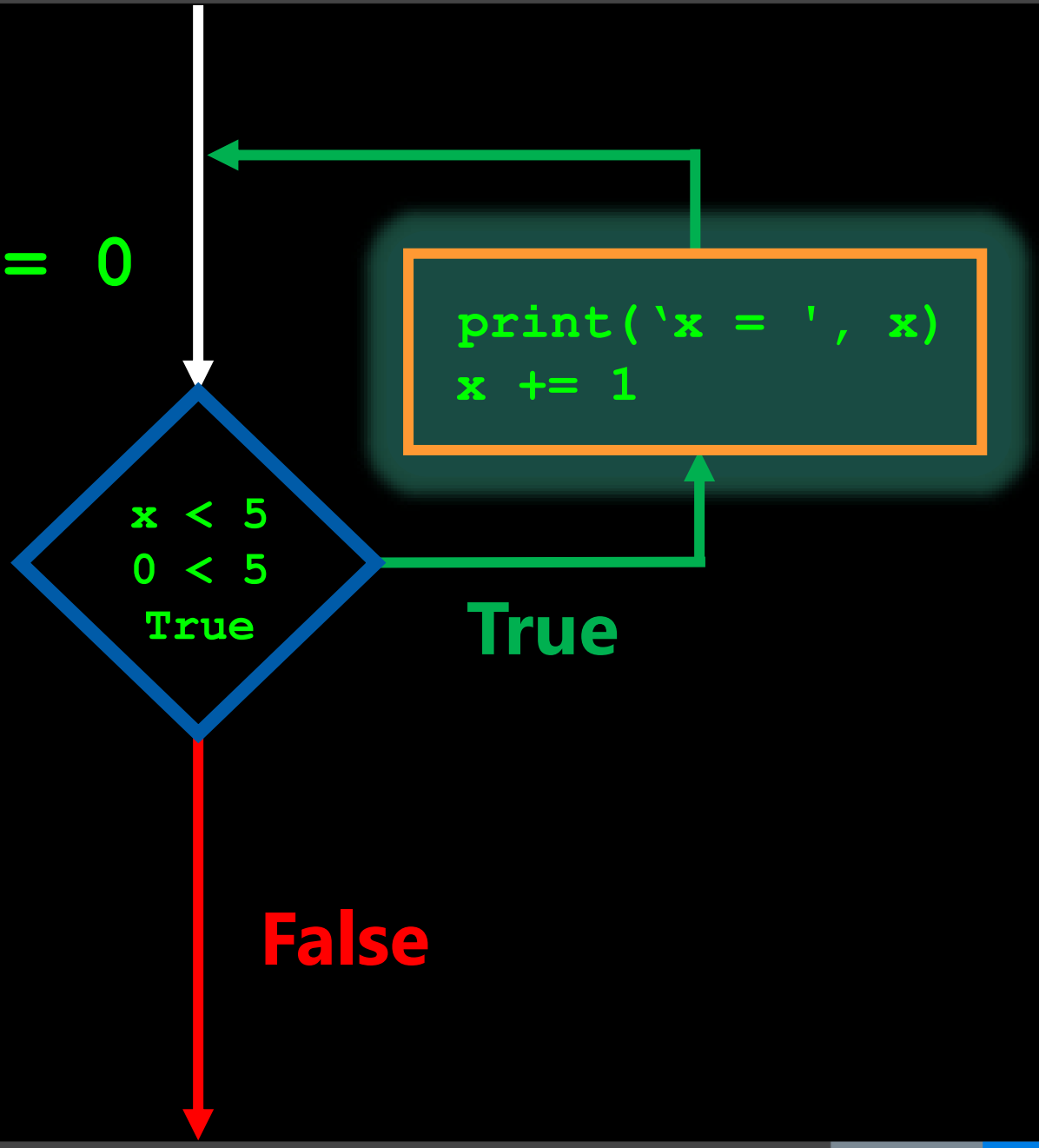


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
```

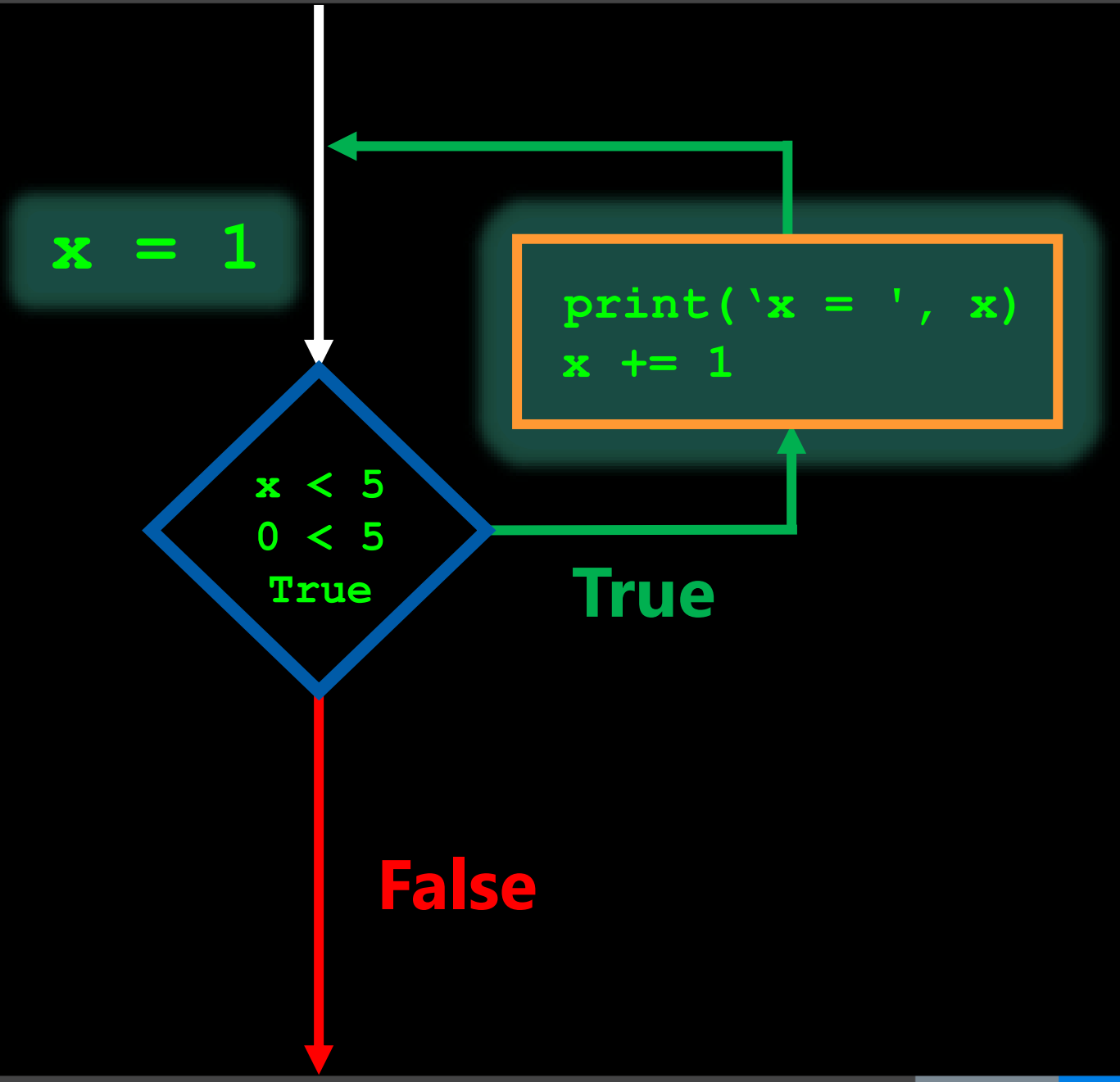


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
```

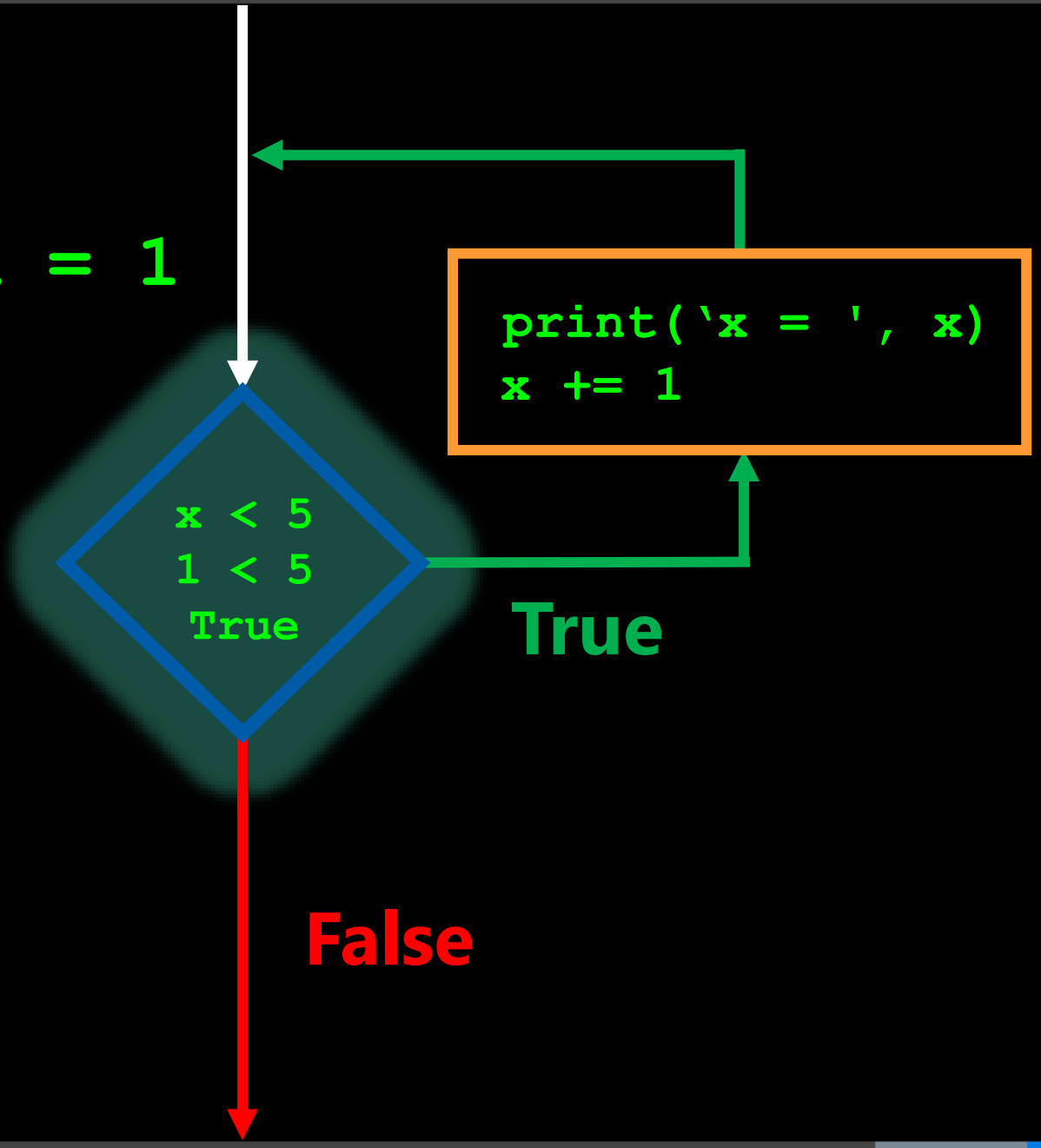


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
```

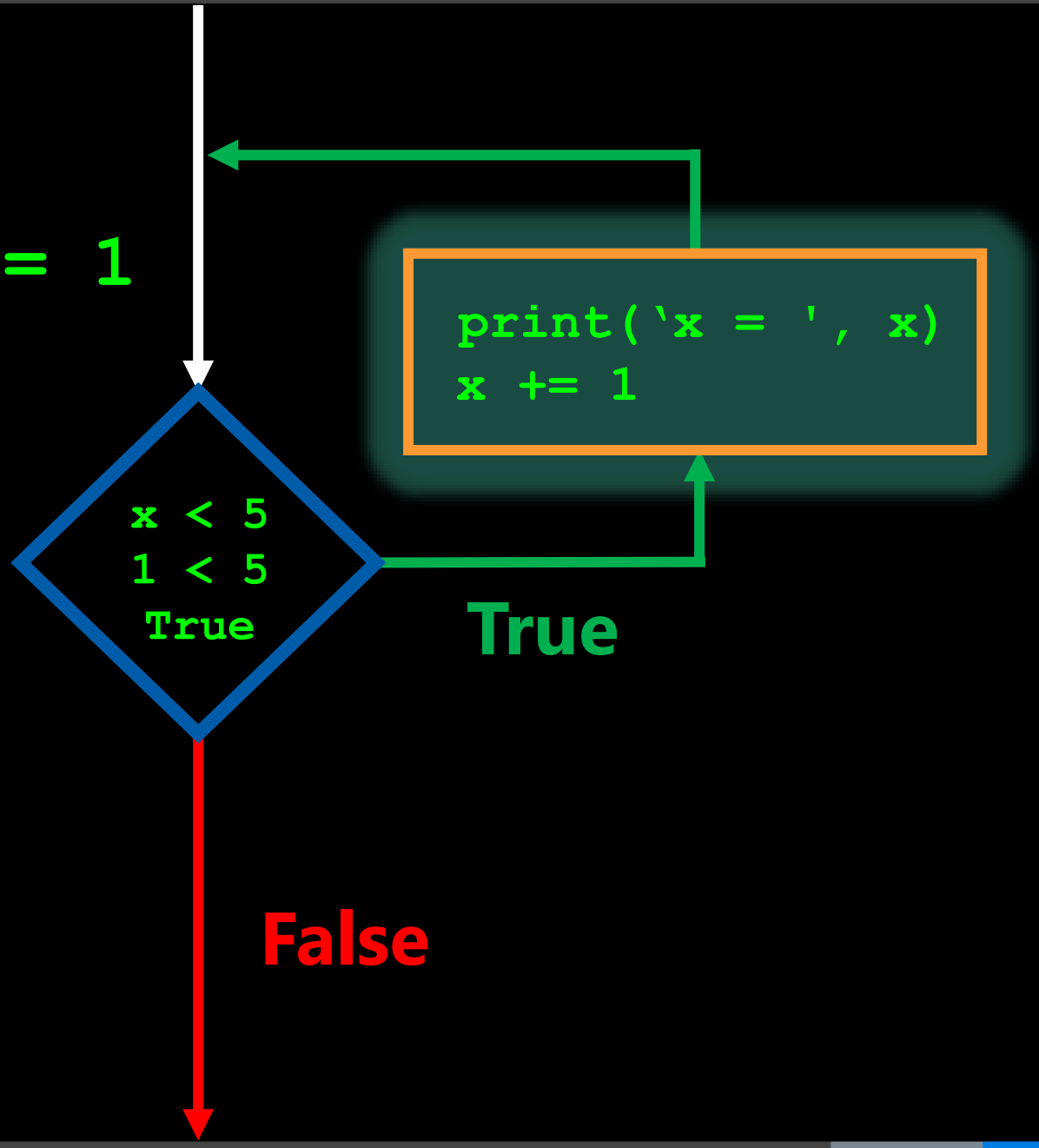


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
```

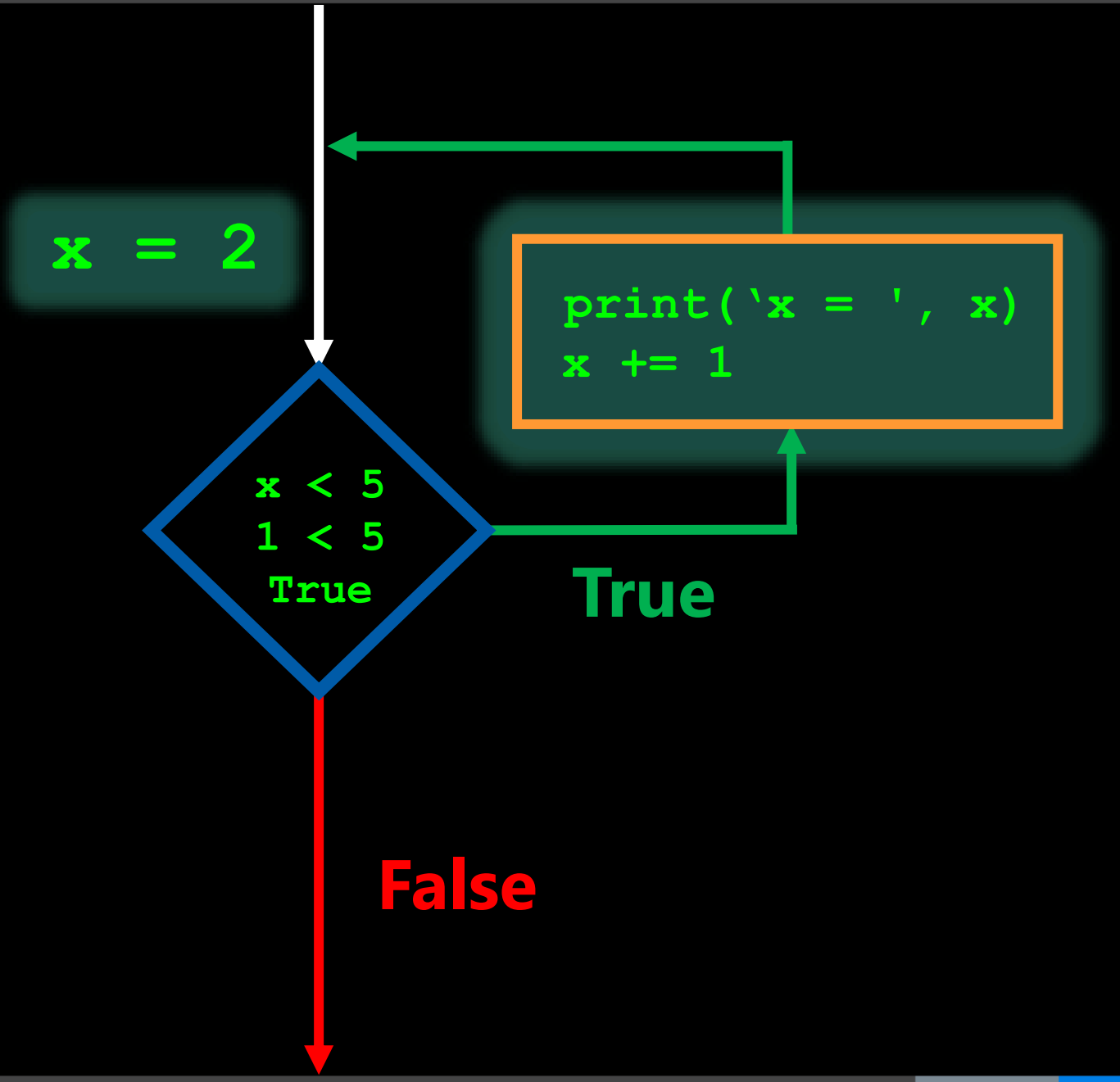


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
```

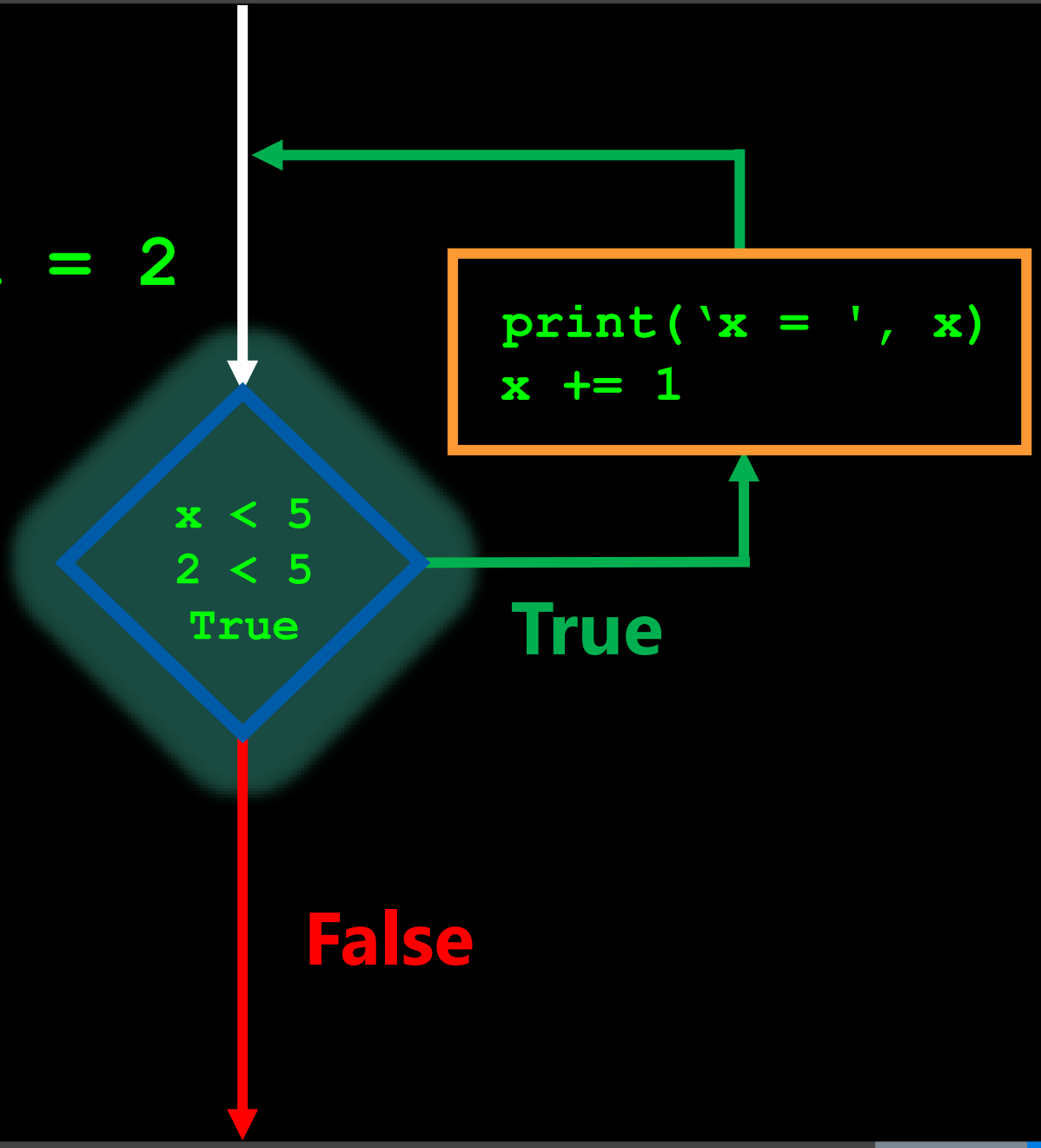


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
```

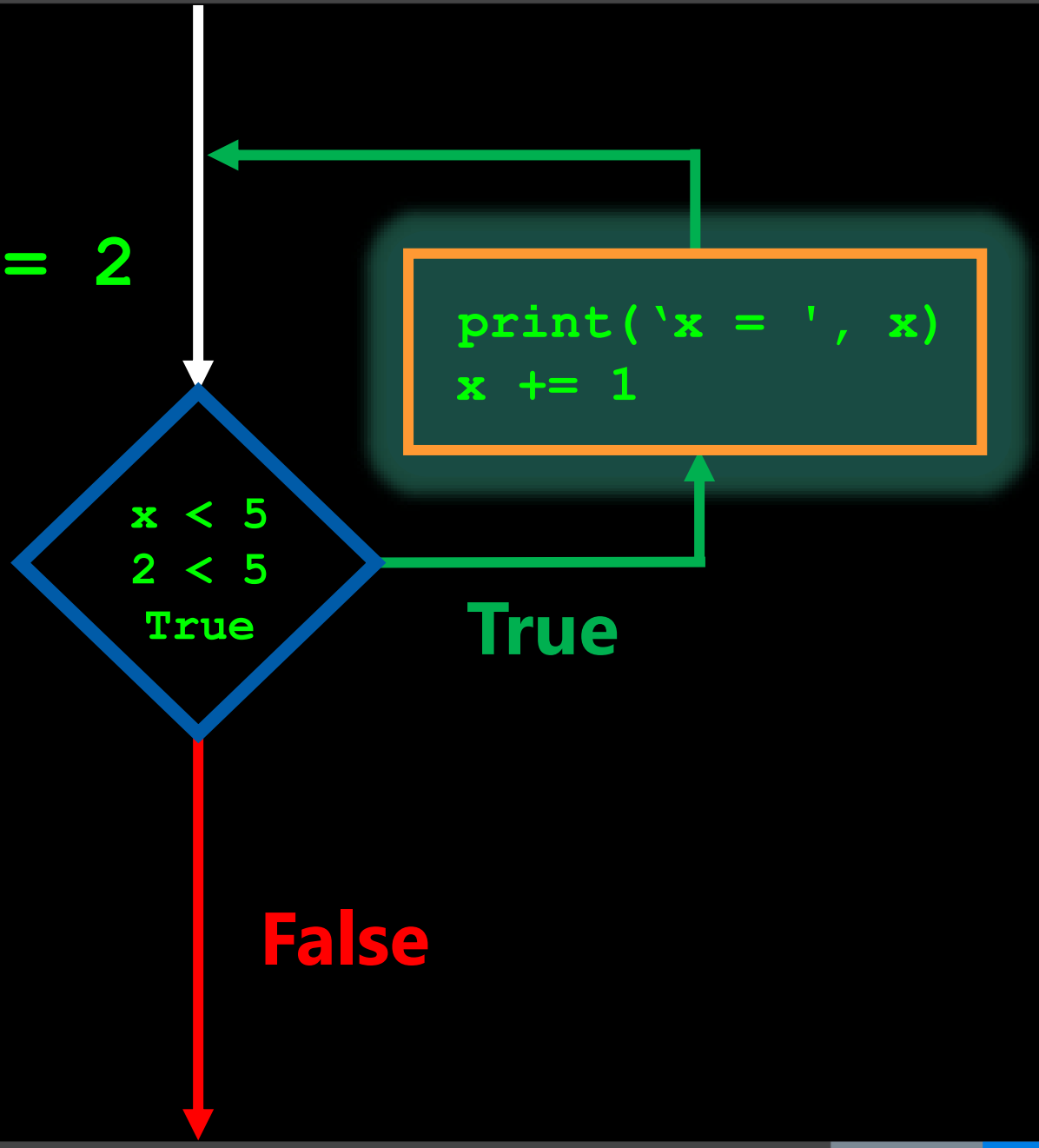


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
x = 2
```

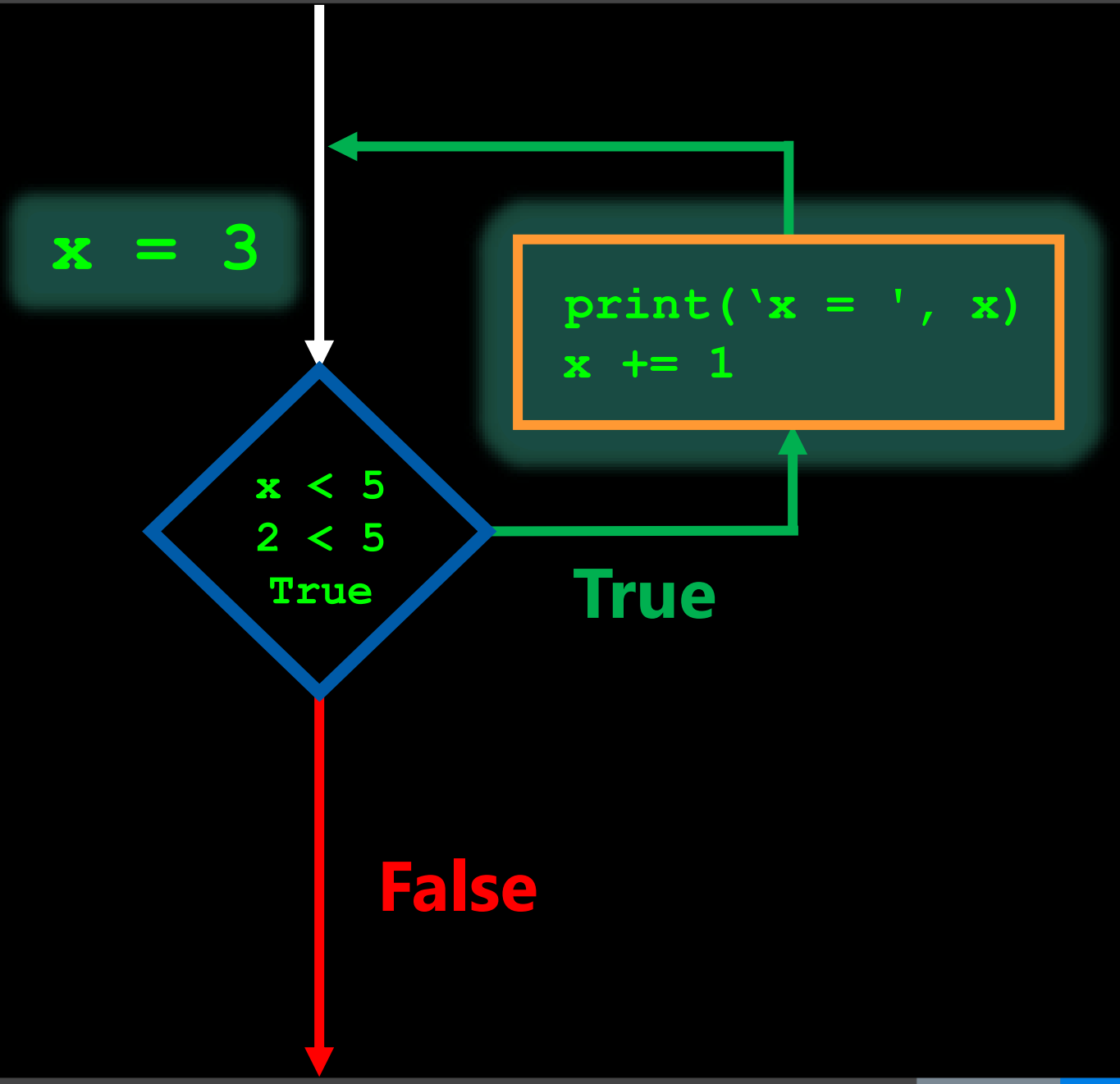


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
x = 2
```

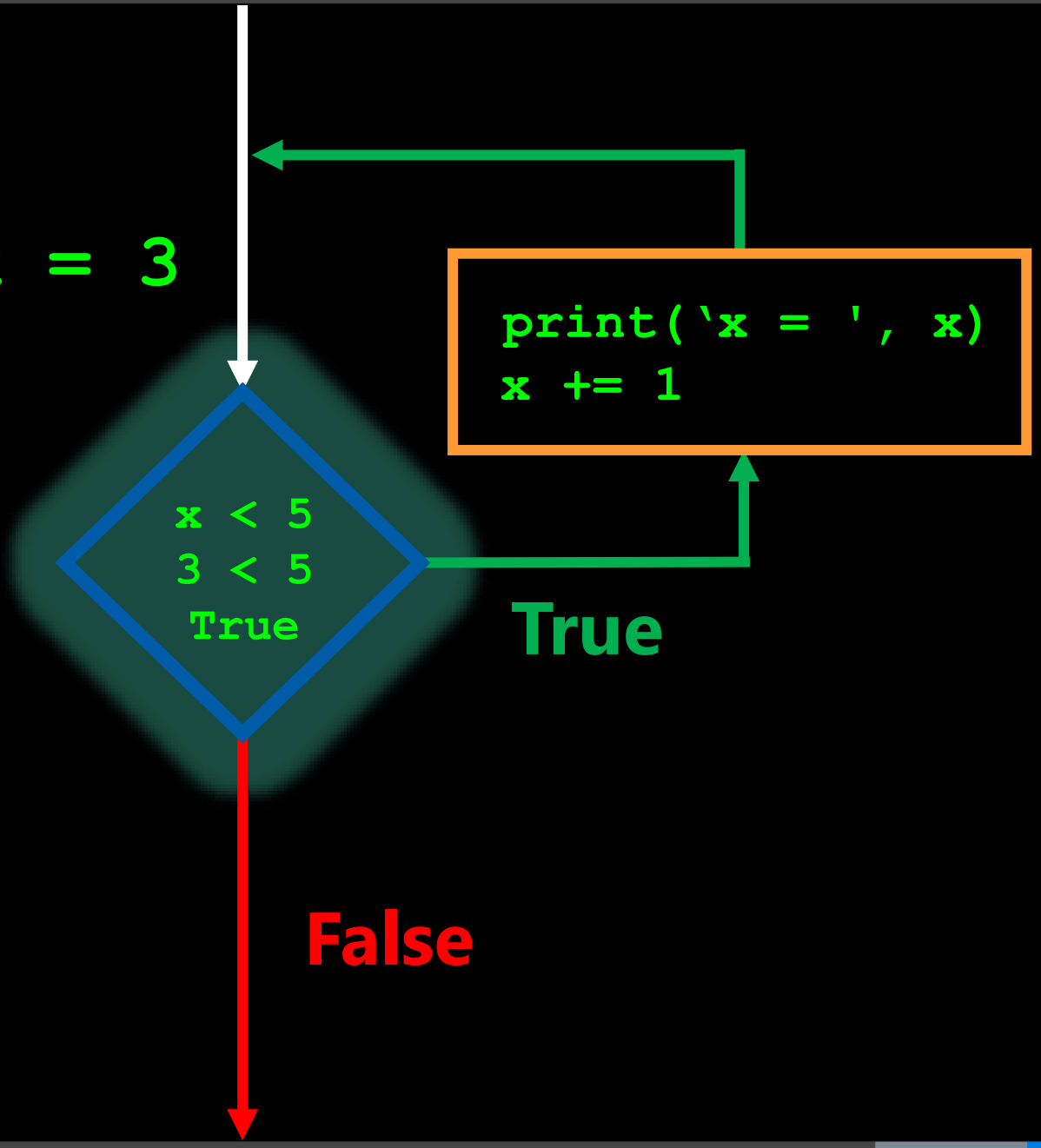


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
x = 2
```

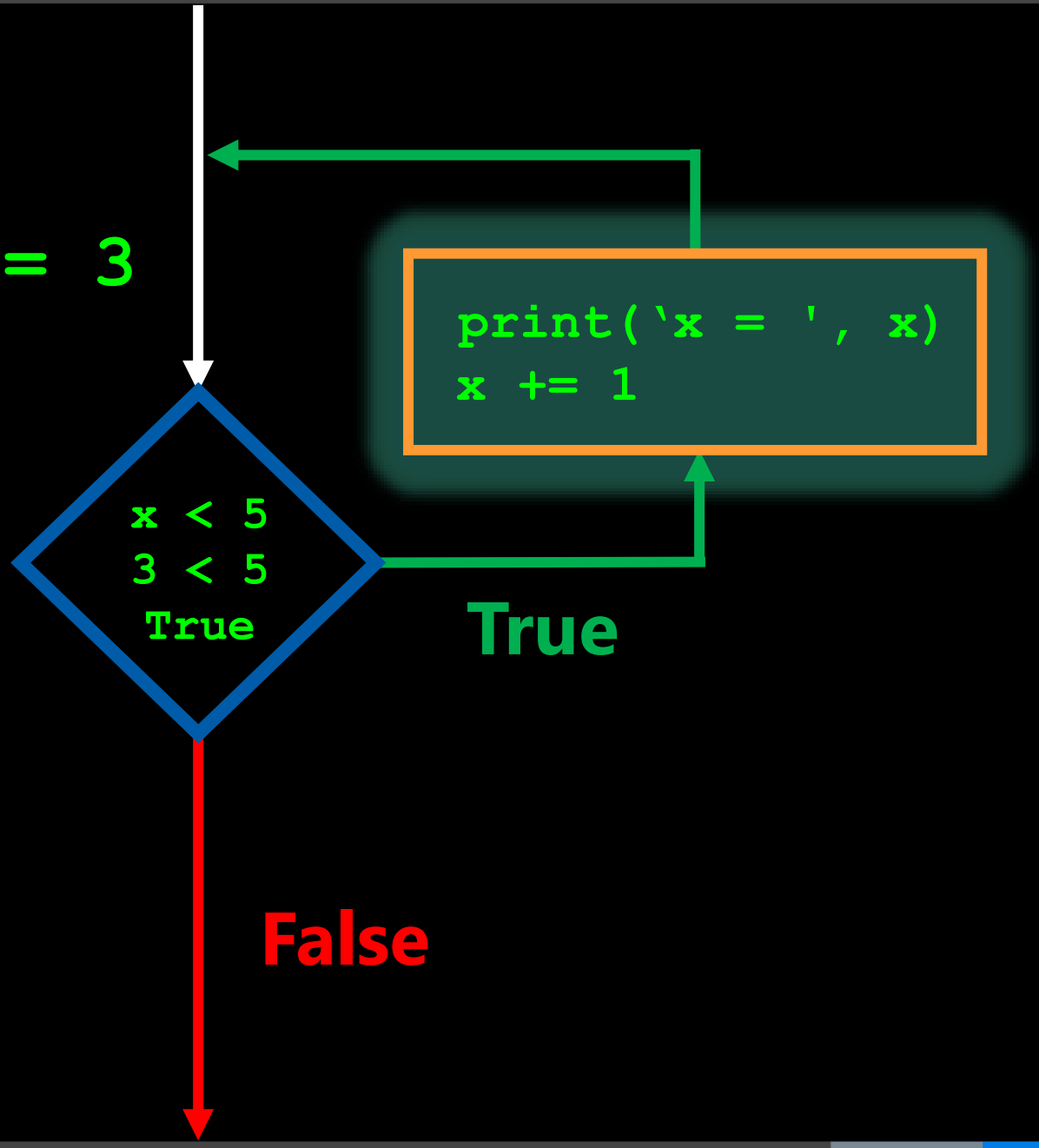


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
x = 2
x = 3
```

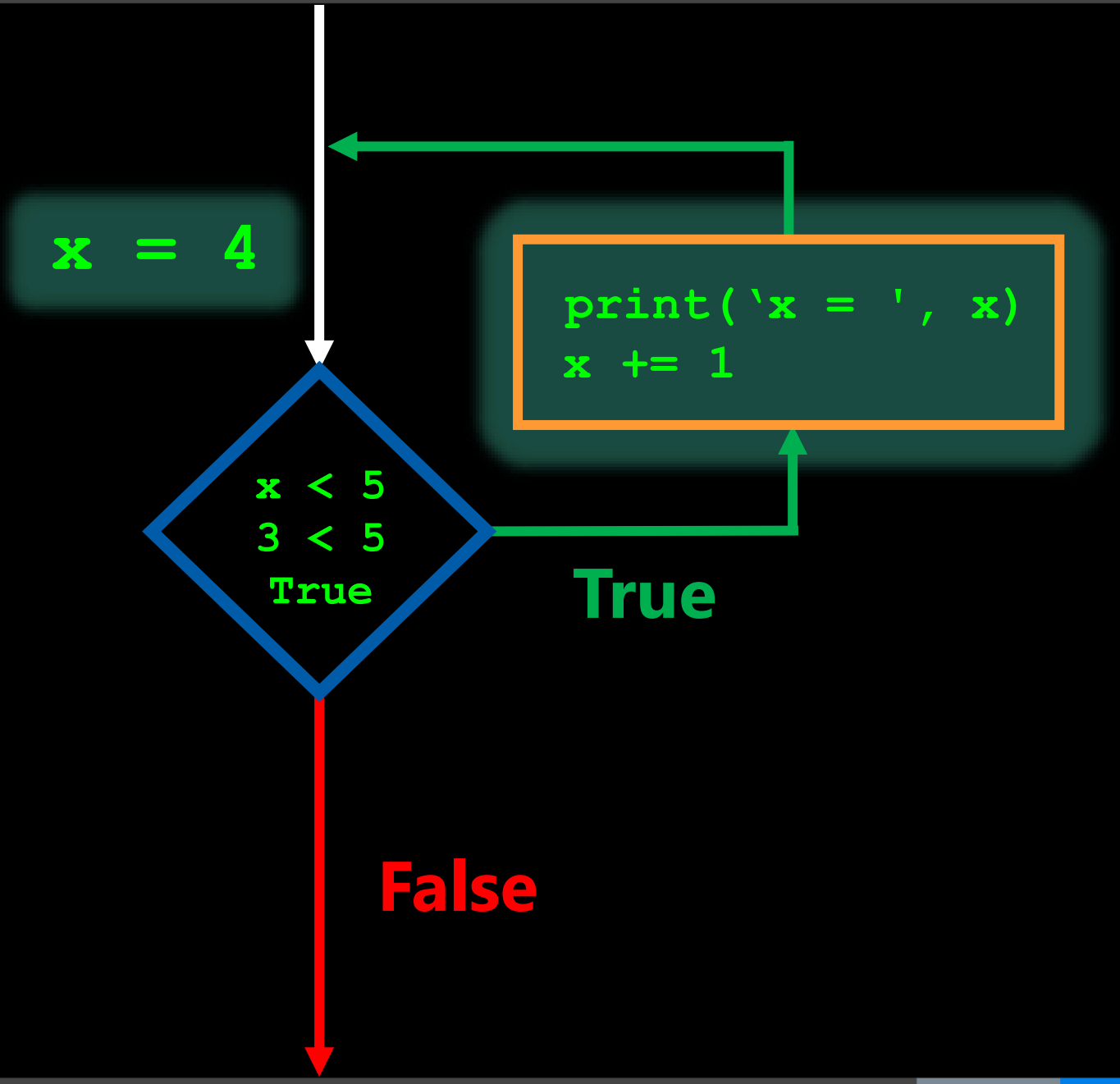


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

Standard Out.

```
x = 0
x = 1
x = 2
x = 3
```

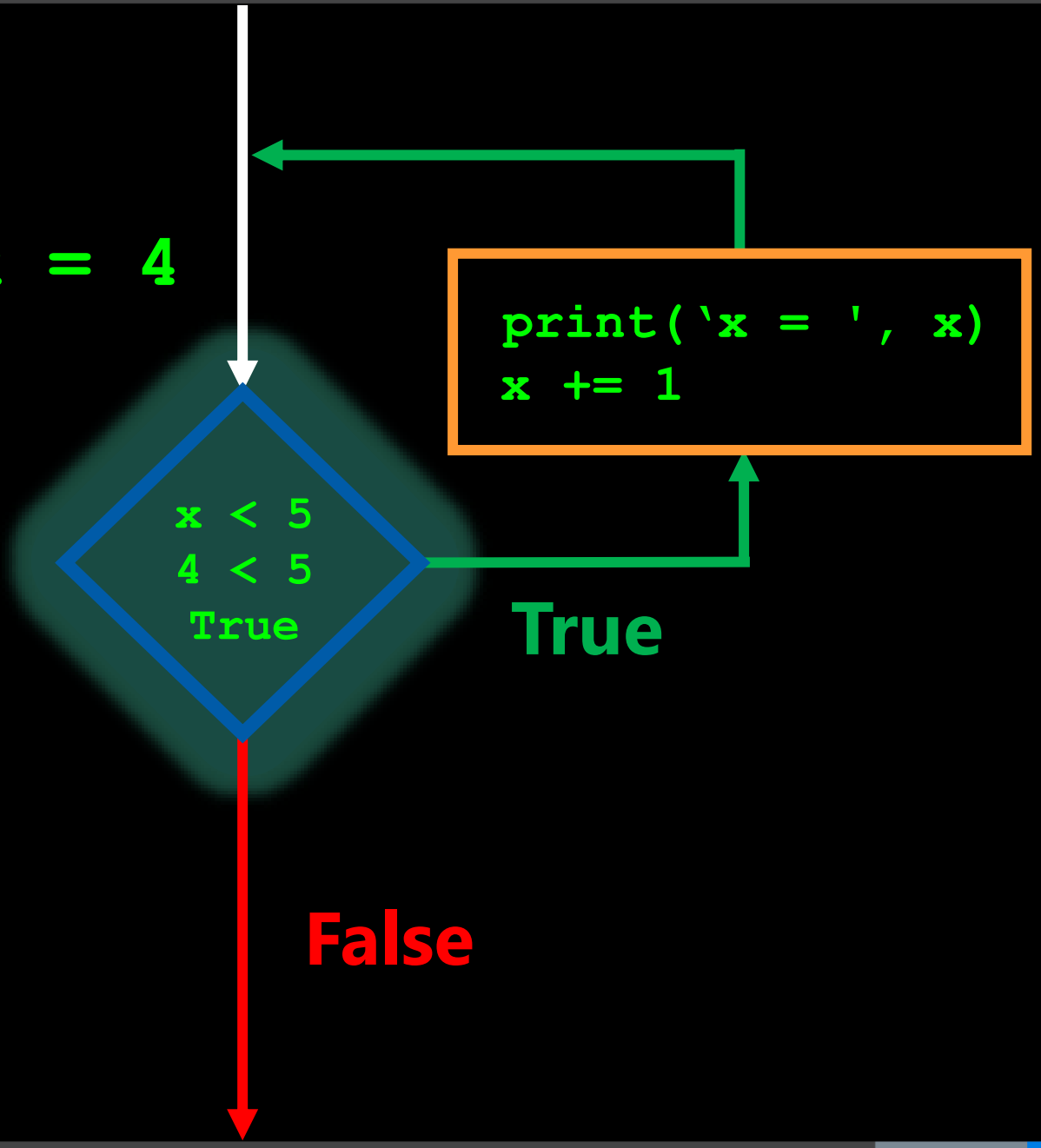


While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

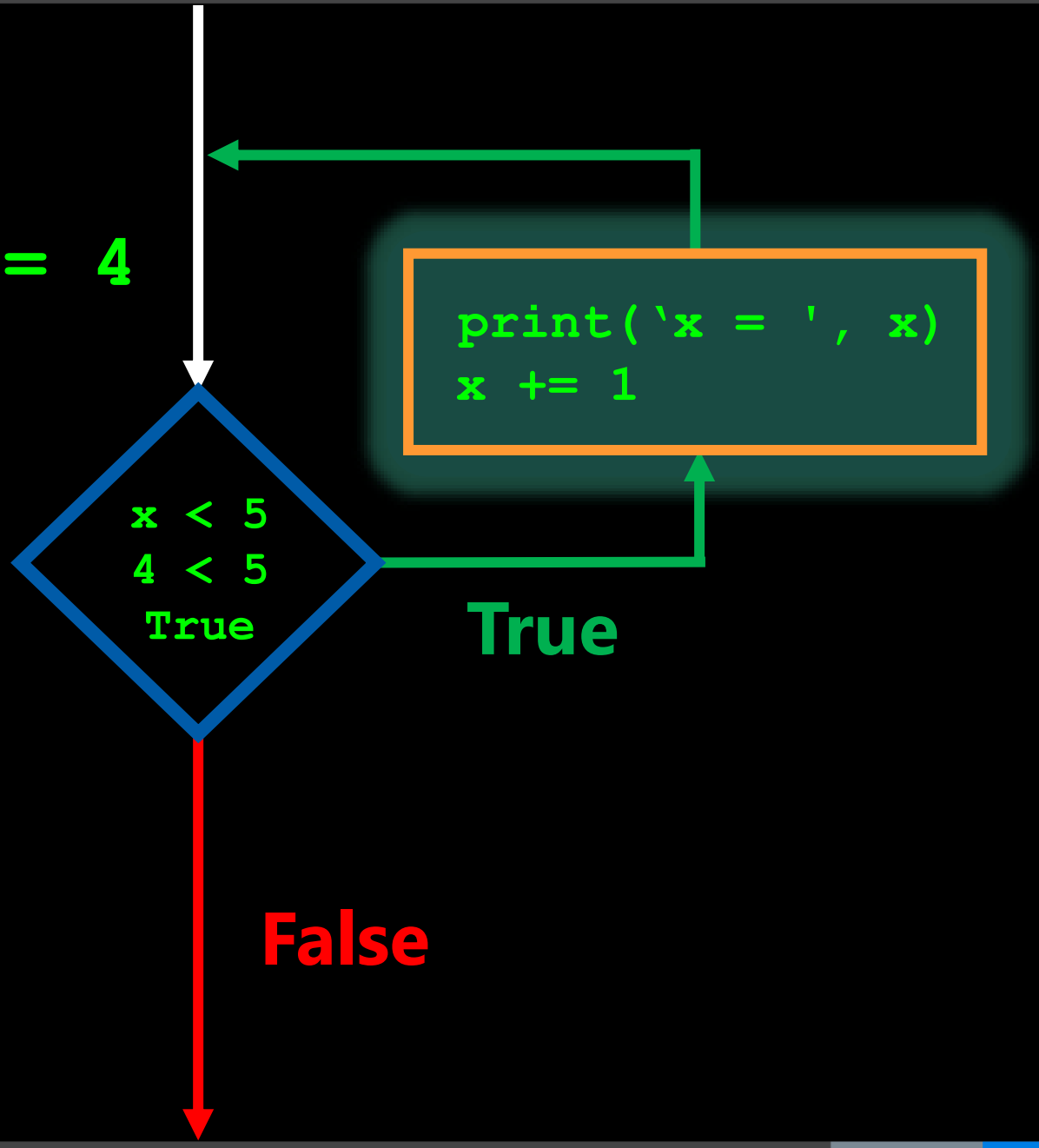
Standard Out.

```
x = 0
x = 1
x = 2
x = 3
```



While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

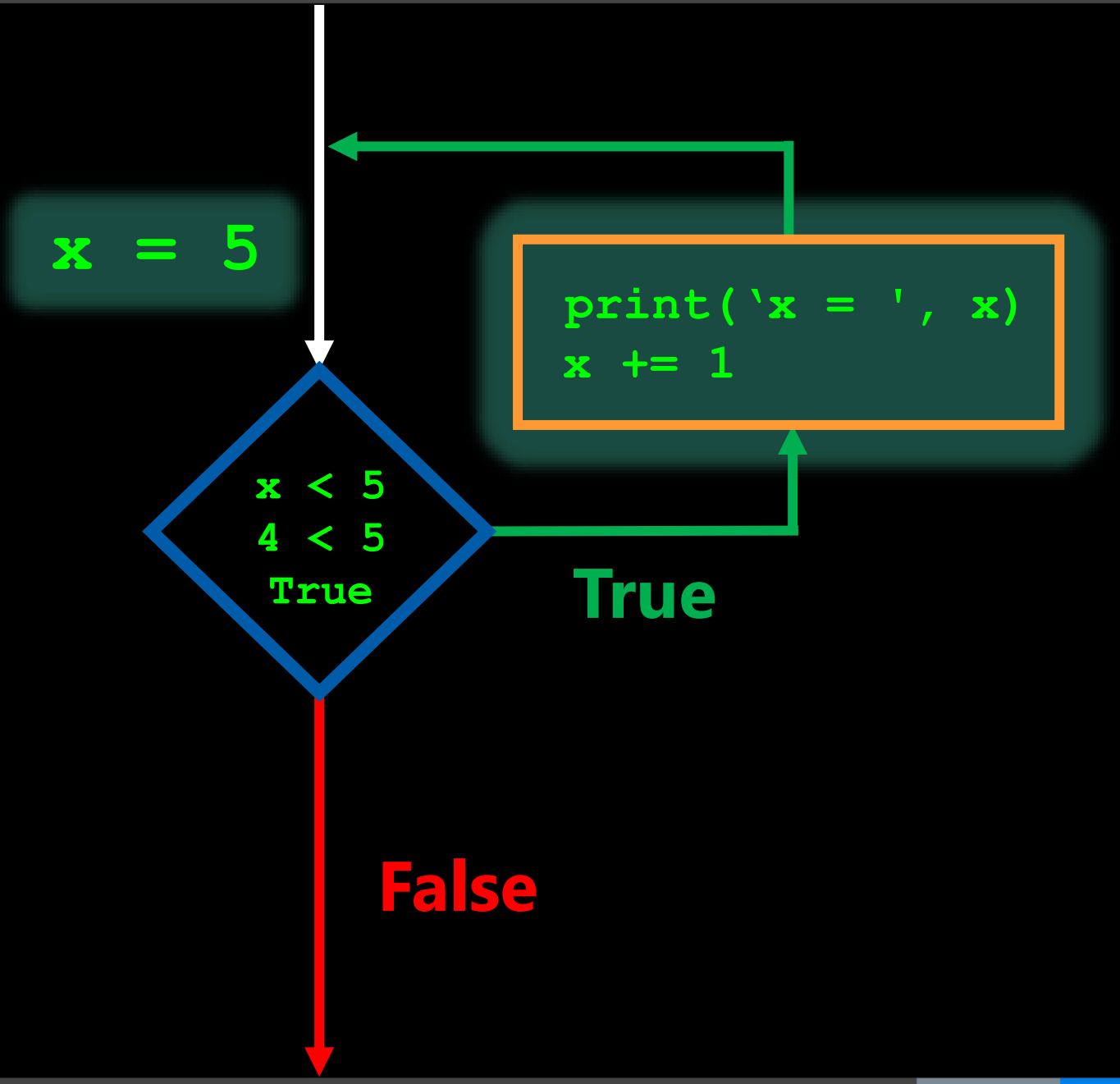


Standard Out.

```
x = 0
x = 1
x = 2
x = 3
x = 4
```

While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

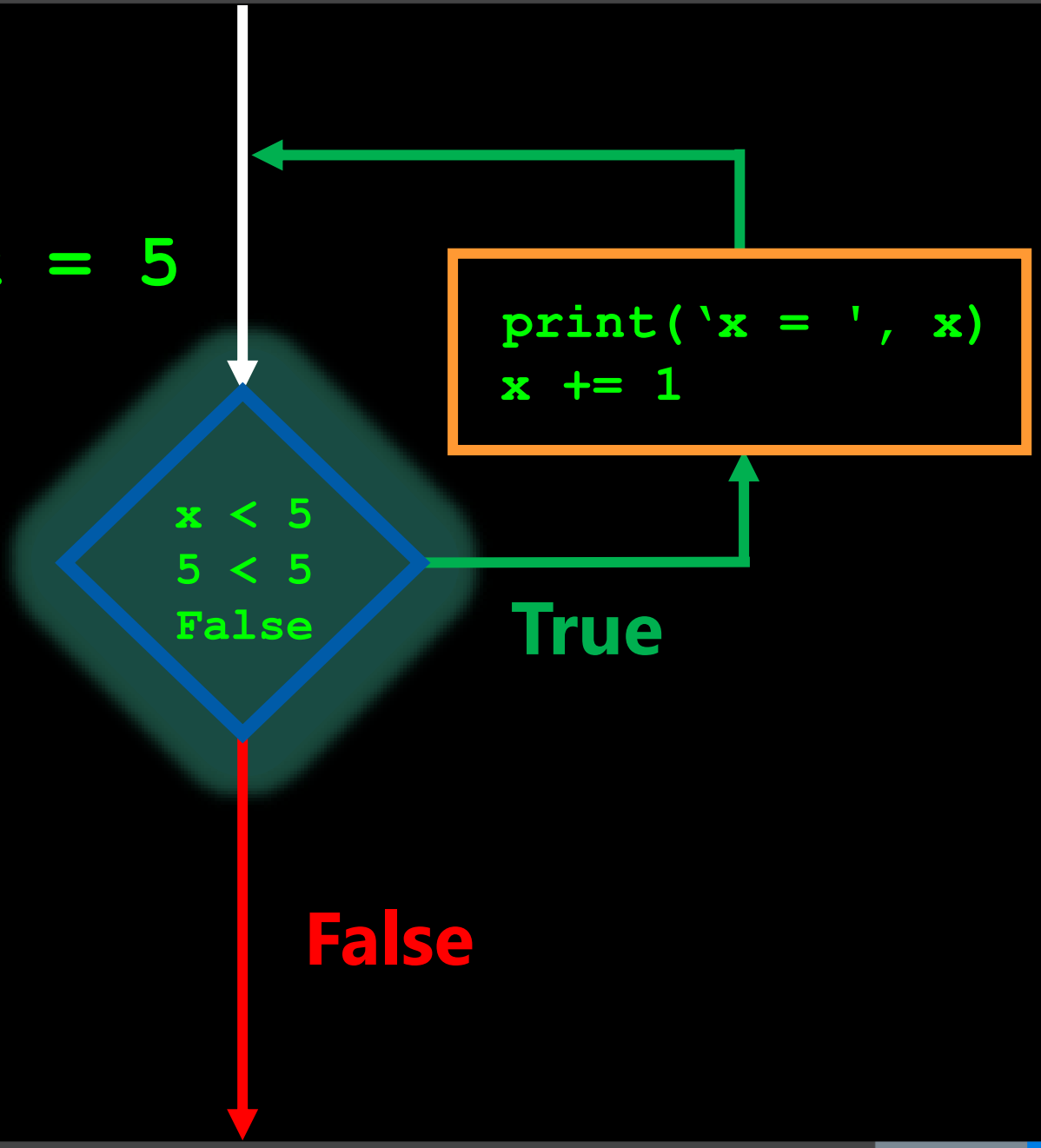


Standard Out.

```
x = 0
x = 1
x = 2
x = 3
x = 4
```

While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```

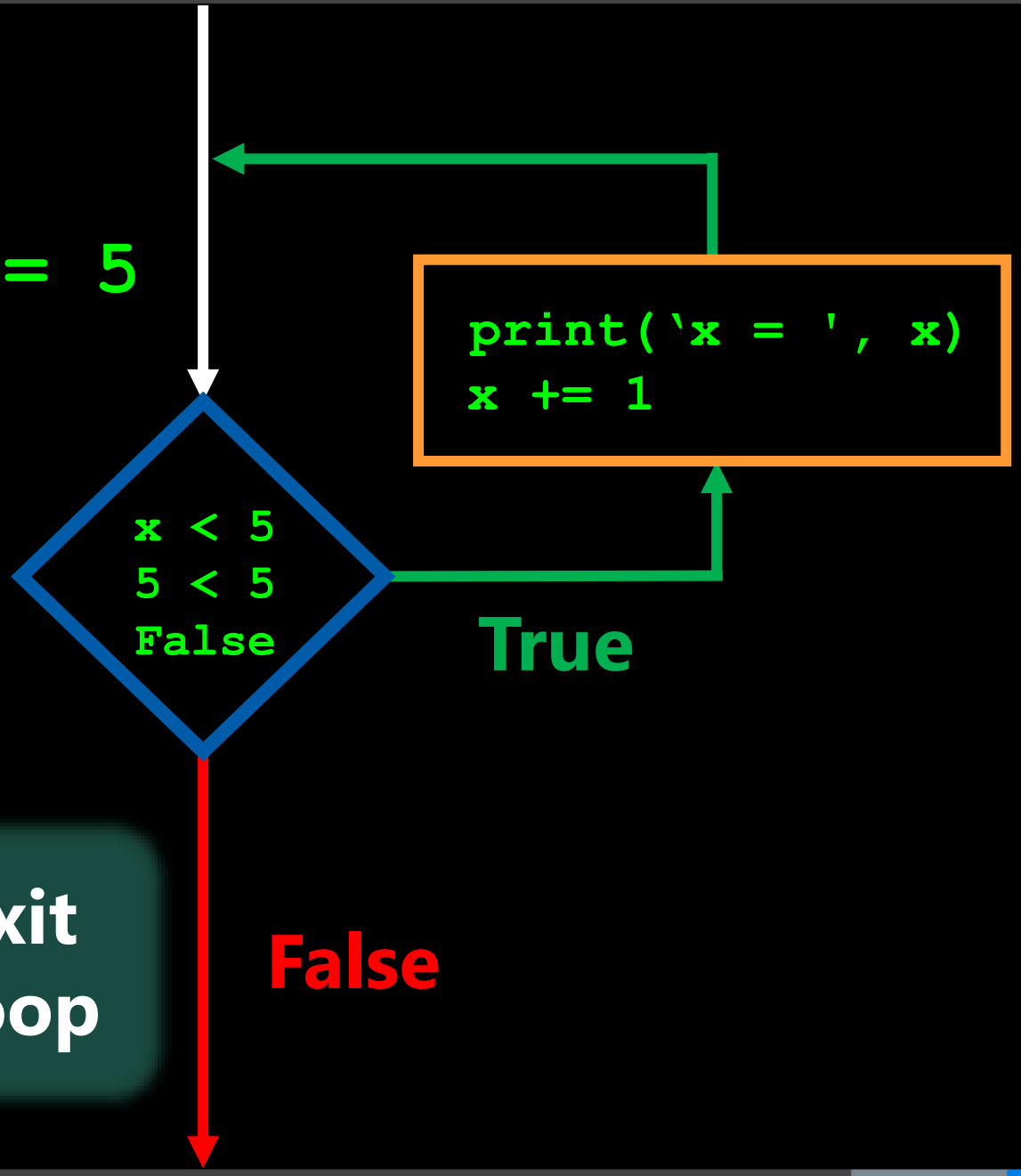


Standard Out.

```
x = 0
x = 1
x = 2
x = 3
x = 4
```

While Loops

```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```



Standard Out.

```
x = 0
x = 1
x = 2
x = 3
x = 4
```

While Loops

Must evaluate to True or False

Colon

```
while expression:
    do something.
```

Indent

Open your notebook

Click Link:
6. While Loops

Infinite Loops

- Remember that a `while` loop ends when the condition is `False`.
- A common error when working with while loops is for the condition to never be satisfied and therefore, the loop to continue forever (till infinity).
- **We need some way inside the loop for the condition to become false.**

```
x = 0
while x < 10:
    print(x)
    x += 1
```

`True`

```
x = 0, 1, 2,
3, 4, 5, 6,
7, 8, 9
```

`False`

```
x = 10
```

Infinite Loops

- Remember that a `while` loop ends when the condition is `False`.
- A common error when working with while loops is for the condition to never be satisfied and therefore, the loop to continue forever (till infinity).
- **We need some way inside the loop for the condition to become false.**

Open your notebook

Click Link:
7. Infinite Loops

While Loops

- Let's revisit our User Input code and see if the While Loop will solve out problem.

Open your notebook

Click Link:
8. Back to User Input

Breakout Session 1

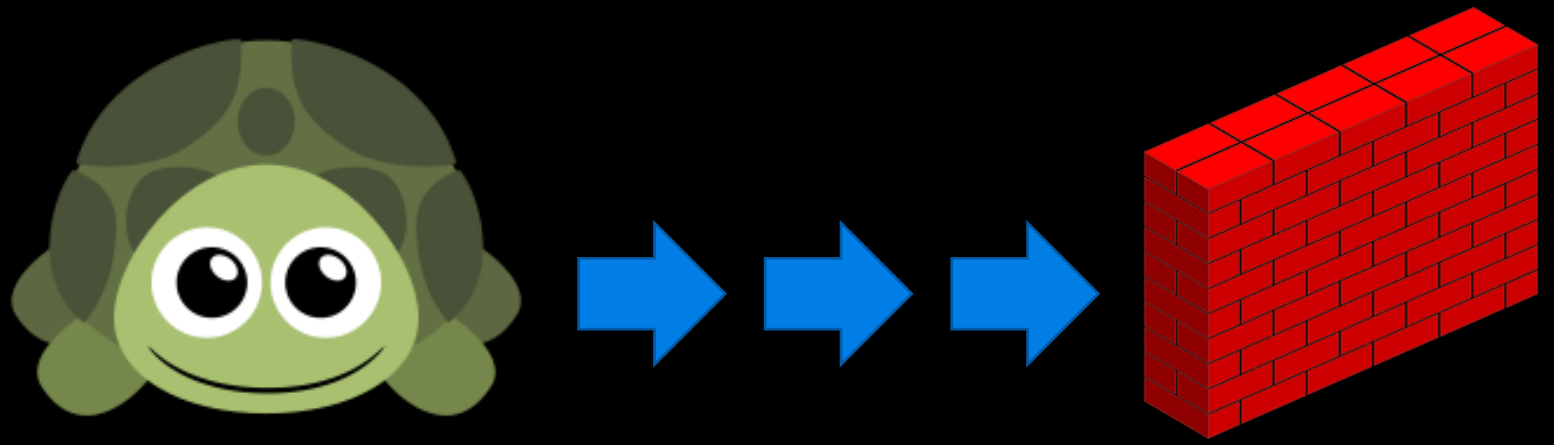
- Write code to print all the numbers from 0 to 20 that aren't evenly divisible by either 3 or 5.
- Zero is divisible by everything and should not appear in the output.

Open your notebook

Click Link:
9. Breakout Session 1

Turtles and `while` loops

- I'm a little turtle and I want to take steps to the right until I get to the brick wall.
- However, I don't know how far away the brick wall I.



Open your notebook

Click Link:
10. Turtles and while loops

Random Module

- This module implements pseudo-random number generators for various distributions.

```
import random

random.uniform()
random.random()
random.randint()
...
```

Open your notebook

Click Link:
11. Random Module

Lecture Recap

Practice!

- Looping (aka iteration) is the second key control structure in programming (if-statements/branching was the first).
- The basic idea of loops is to repeated execute the same block code.
- Looping is very powerful idea.
- While loops is one of two loop types in Python.

APS106

UNIVERSITY OF TORONTO

functions, input & output, importing modules.

Week 4 | Lecture 1 (4.1)

if nothing else, write #cleancode