# APS106

UNIVERSITY OF TORONTO

# while loops.

**Week 2** | Lecture 2 (2.2.2)

**While waiting for class to start:**

Download and open the Jupyter Notebook (.ipynb) for Lecture 2.2.2

You may also use this lecture's JupyterHub link instead (although opening it locally is encouraged).

**Upcoming:**

- Reflection 2 released Friday @ 11 AM
- Lab 3 released Friday @ 12 PM
- Lab 2 **deadline** this Friday @ 11 PM
- PRA (Lab) on Friday @ 2PM this week (ONLINE)

if nothing else, write #cleancode

# Today's Content

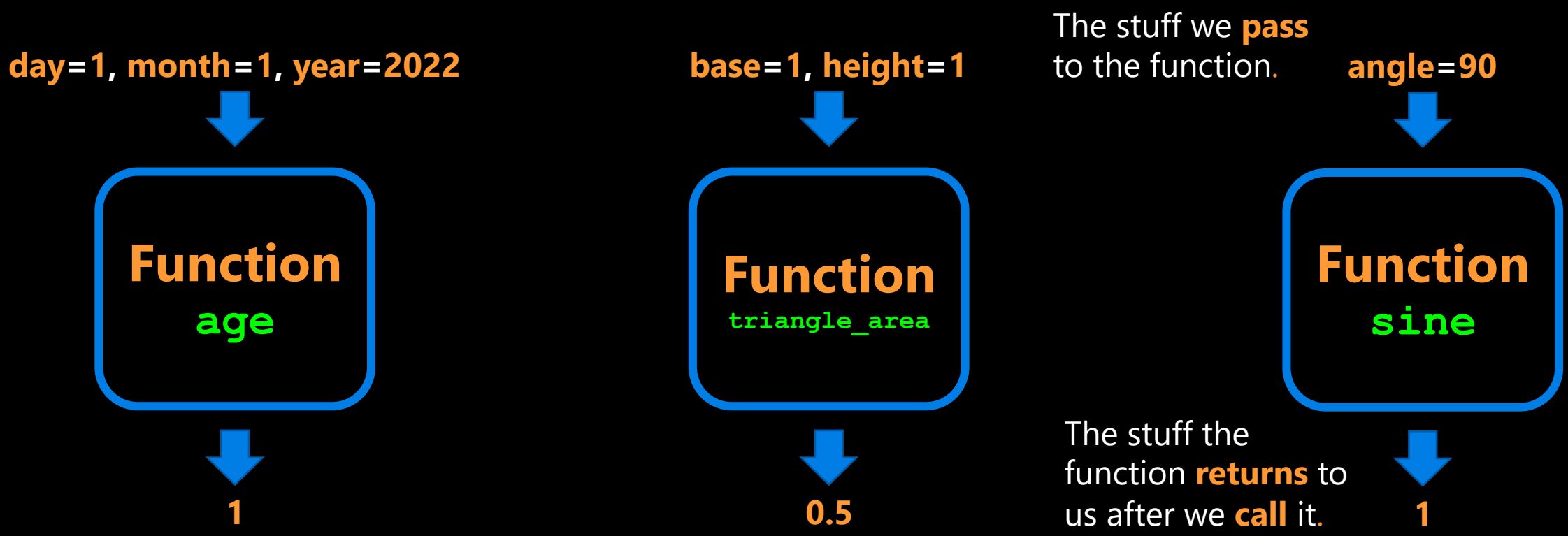
- Lecture 2.2.1
  - Rock, Paper, Scissors, Lizard, Spock
- **Lecture 2.2.2**
  - **while loops**

# `function` confusion

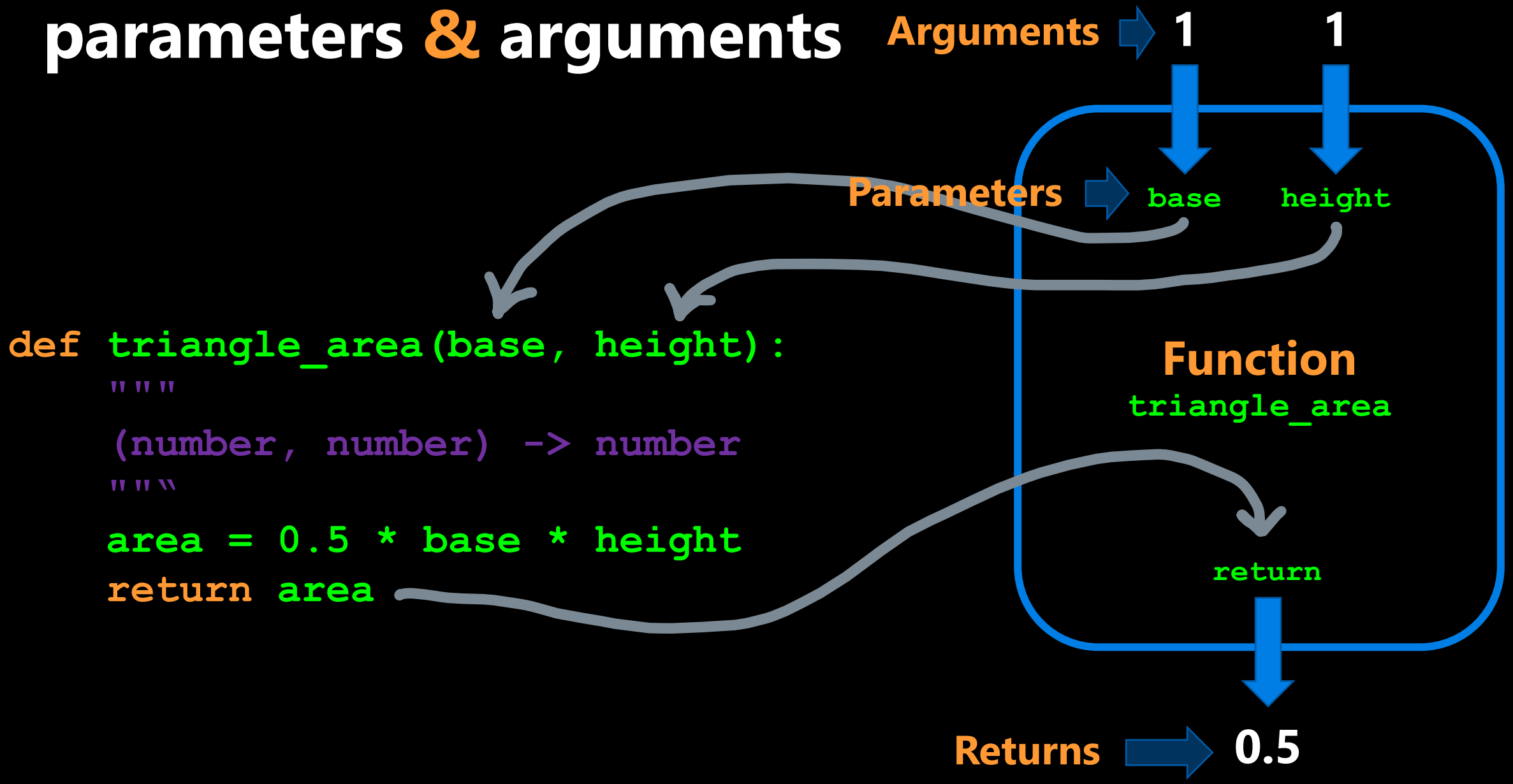
- Review.
- `parameters` and `arguments`.
- `print` and `return`.
- When is a function done?

# function, what are they?

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

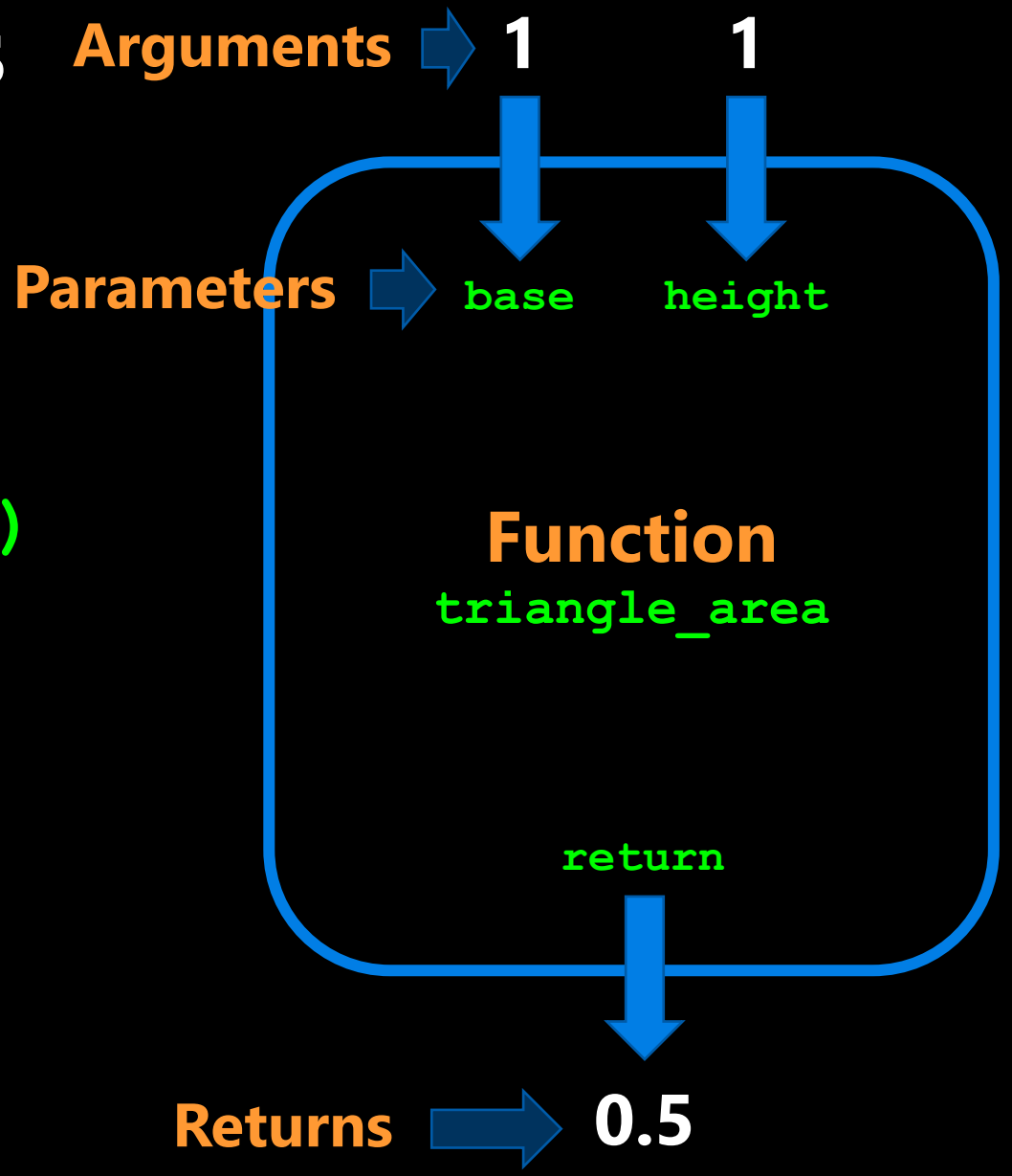


# parameters & arguments



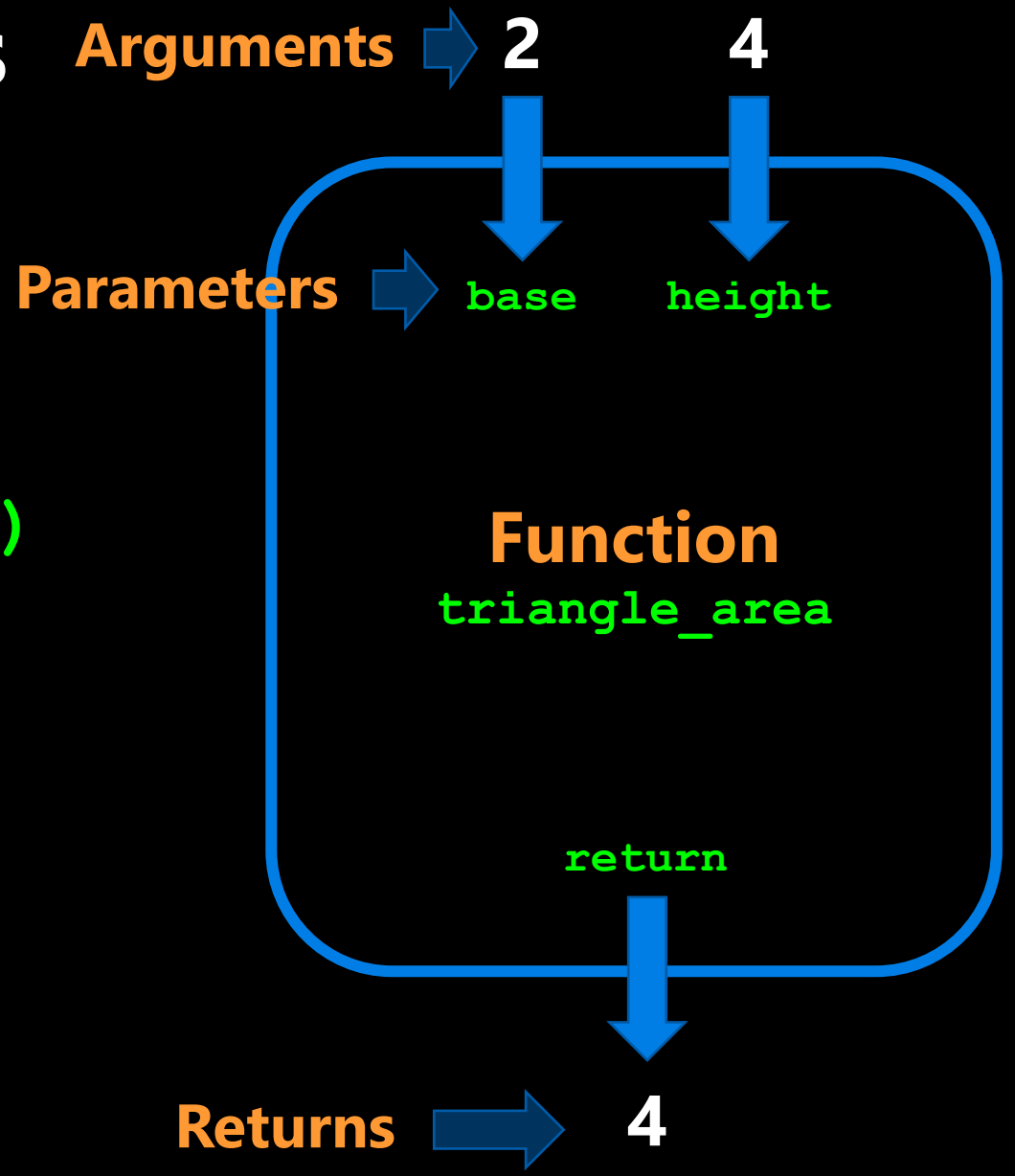
# parameters & arguments

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



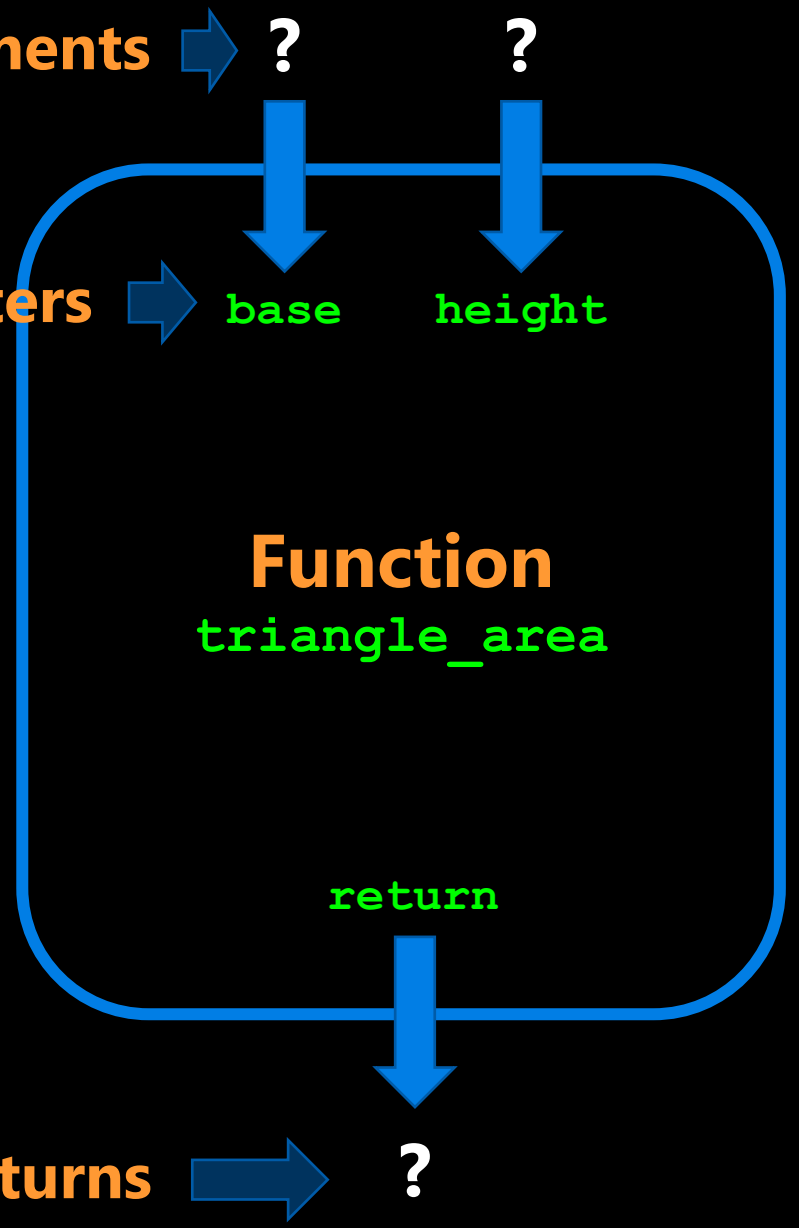
# parameters & arguments

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



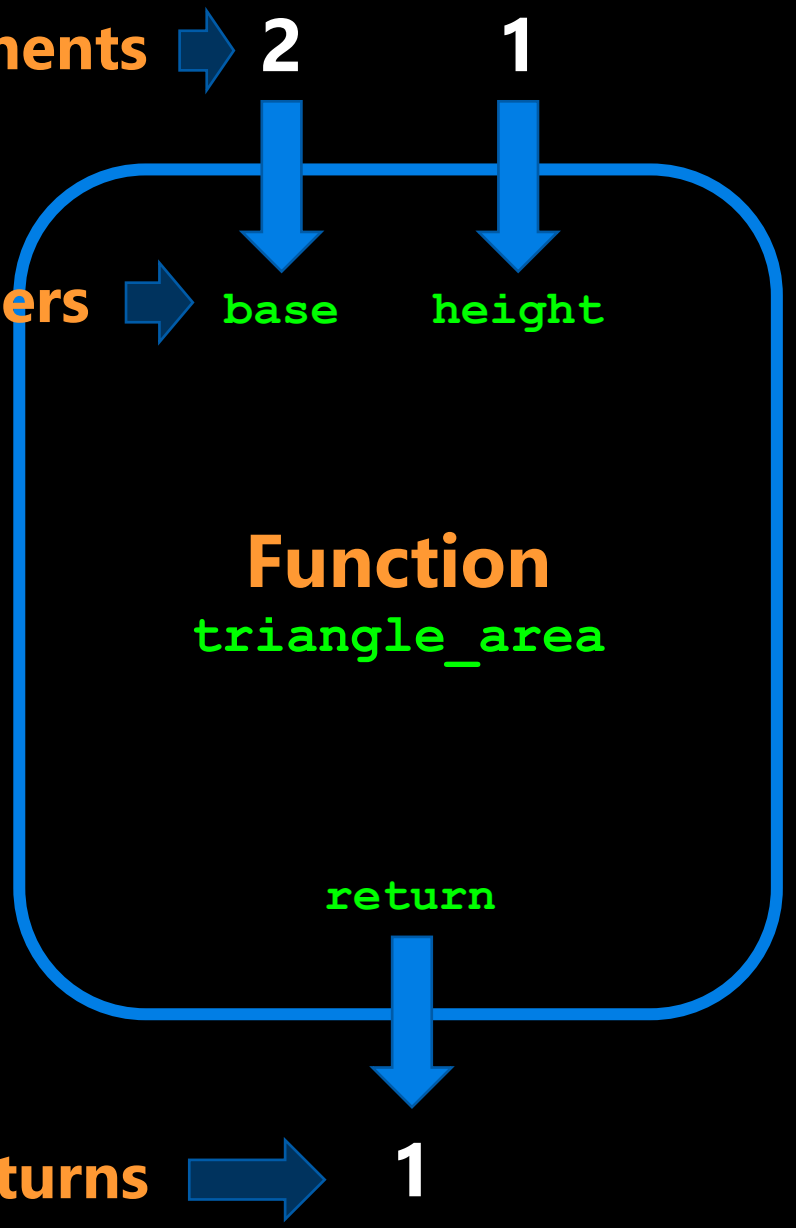
# parameters & arguments

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



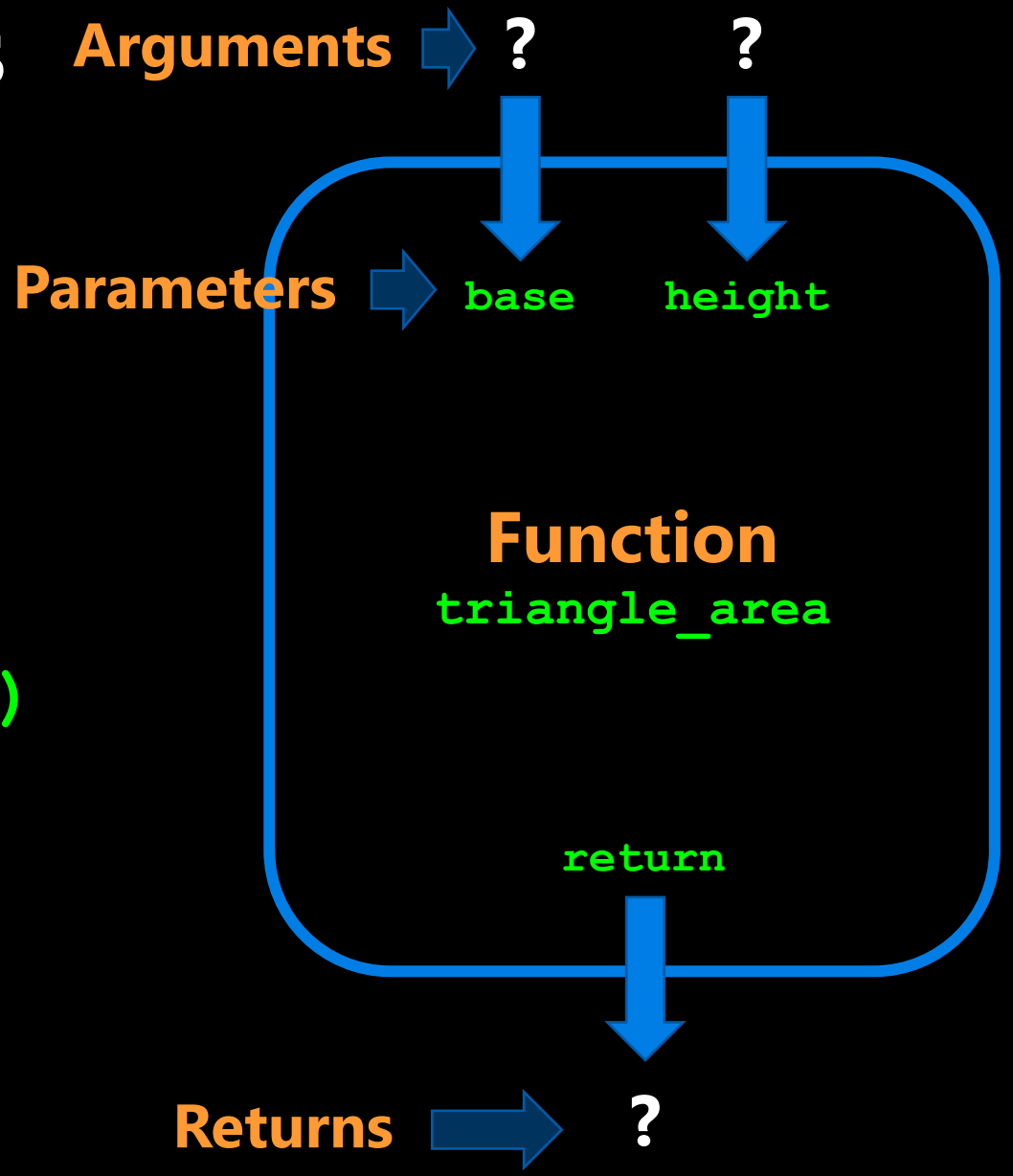
# parameters & arguments

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



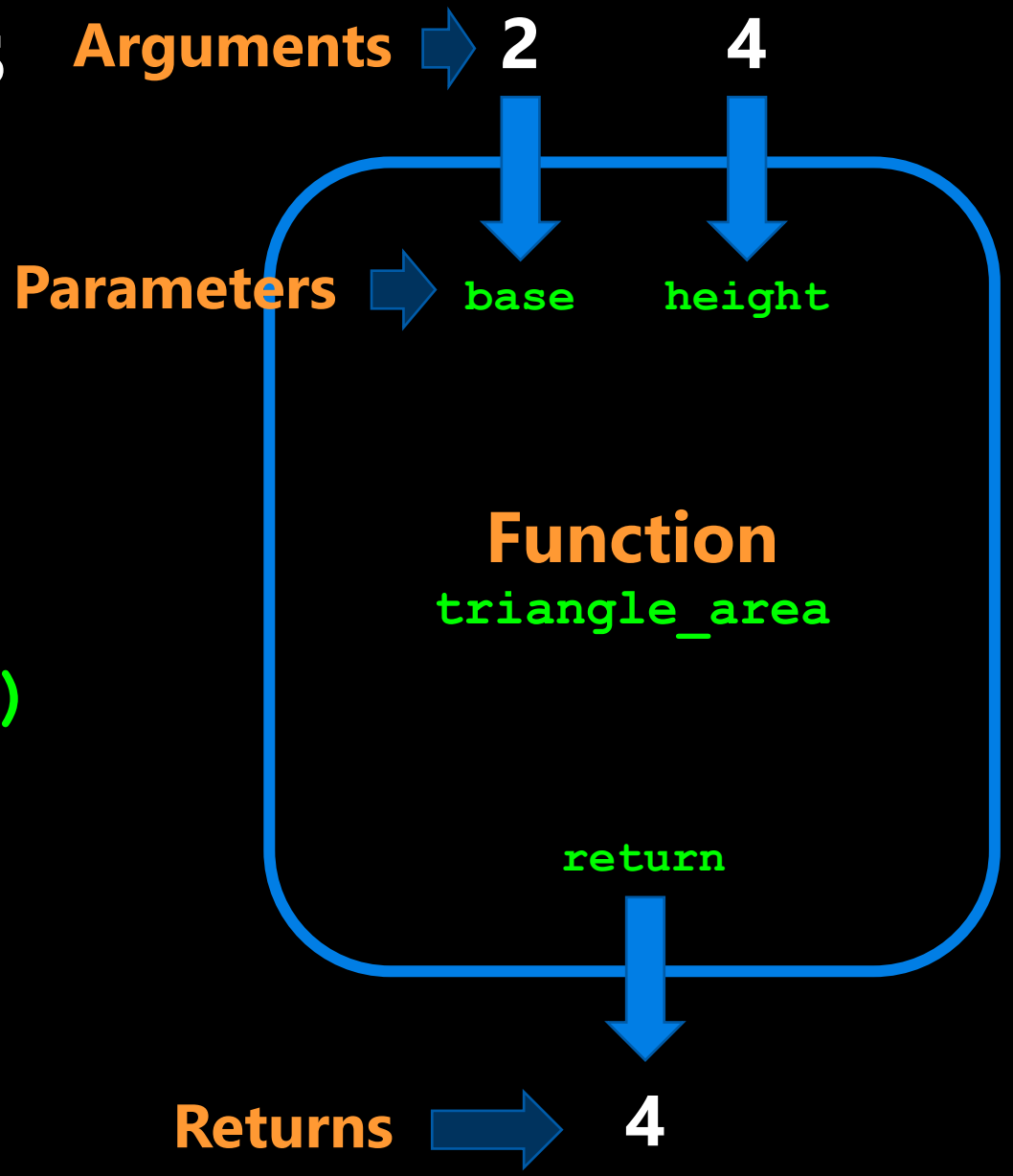
# parameters & arguments

```
>>> 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
```

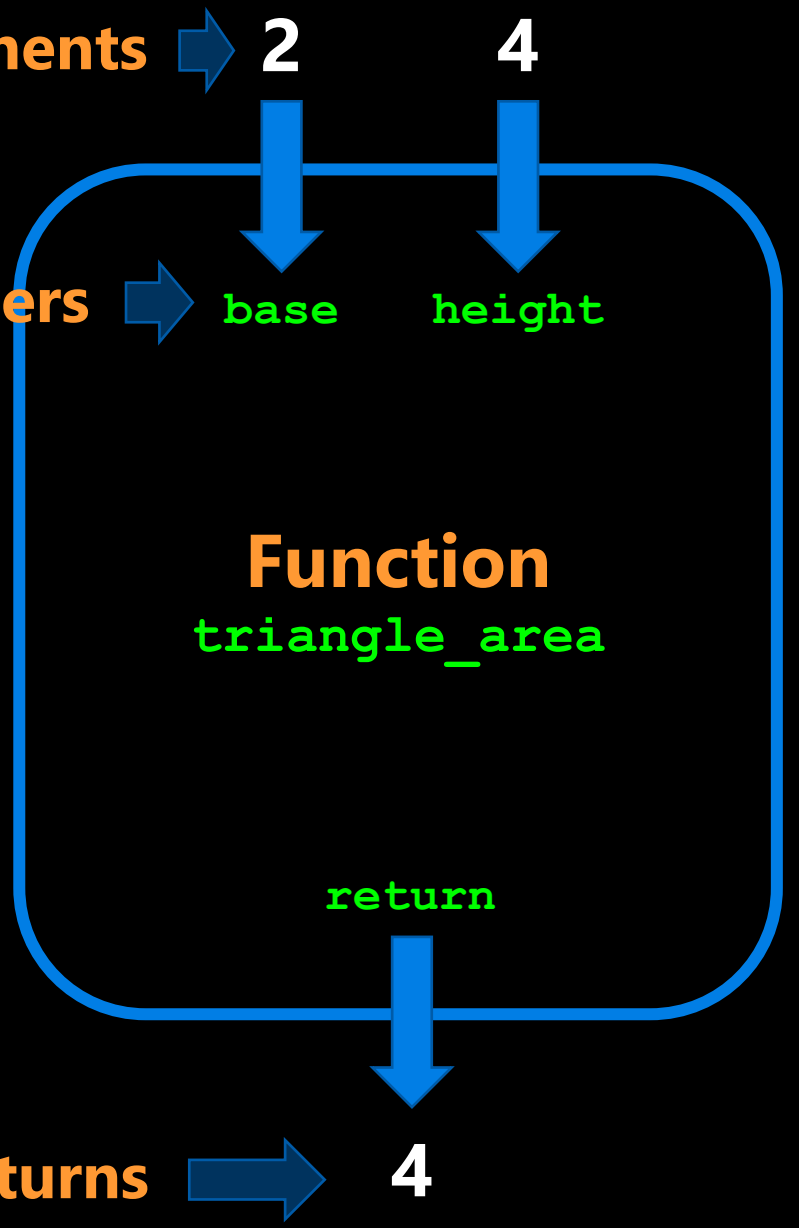


# parameters & arguments

```
>>> 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
```



# `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.


`return`


`print`

## `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

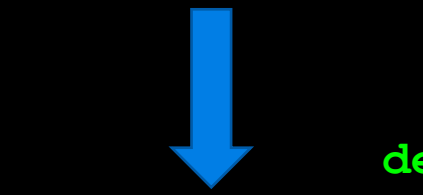
```
def square(x):
    output = x * x
    return output
```

```
>>> 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

**Standard Out** is a single area of text shared by all the code in a program.

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

**Arguments:** 2

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

**Function**
**square**

Standard Out.

4

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**

Standard Out.

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

**Returns:** 4

# 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
end. ➡ return output
```

```
>>> 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
    return None
```

end. (end of indented code)

If there is no return statement, Python adds one and returns None.

```
>>> 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.

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

```
def func(x):
    output = x * x
    output += 10
    output /= 2
    return None
```

end. (end of indented code)

If there is no return statement, Python adds one and returns None.

```
>>> 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.

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

```
def func(x):
    output = x * x
    output += 10
    output /= 2
    return None
```
end. (end of indented code)

If there is no return statement, Python adds one and returns None.

```
>>> 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?

```
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
    return None
```
end. (end of indented code)

If there is no return statement, Python adds one and returns None.

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

```
def func(x):
    output = x * x
    return output
    output += 10
    output /= 2
    return None
```
end.

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

# When is a function done?

- Let's look at some examples.

**Open your notebook**

**Click Link:**
**1. Function Review**

# Looping (Iterating)

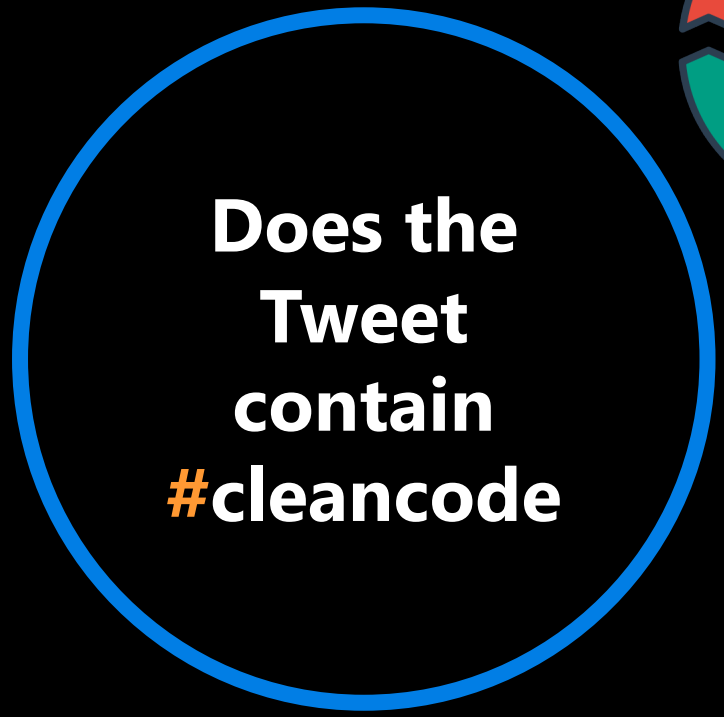
- Looping means repeating something over and over until a particular condition is satisfied.
- Looping (aka iteration) is the second key control structure in programming (if-statements/branching was the first).

# Looping (Iterating)

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

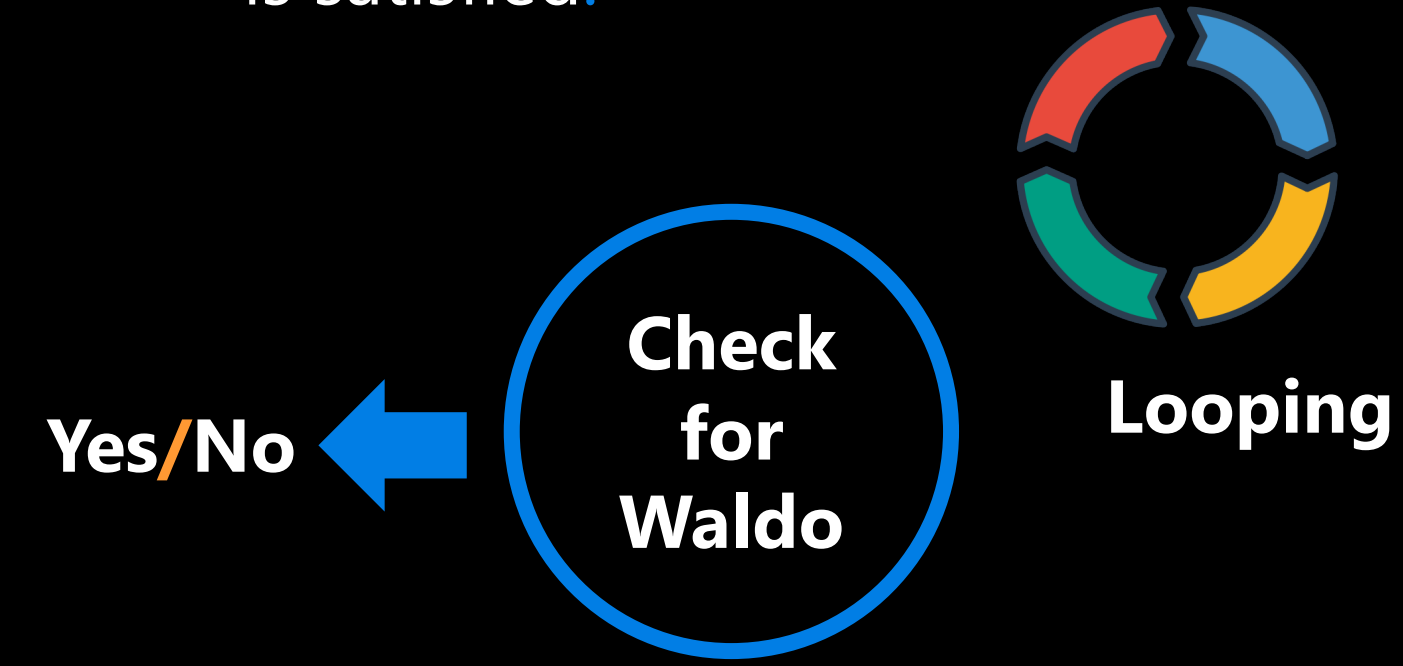
# Looping (Iterating)

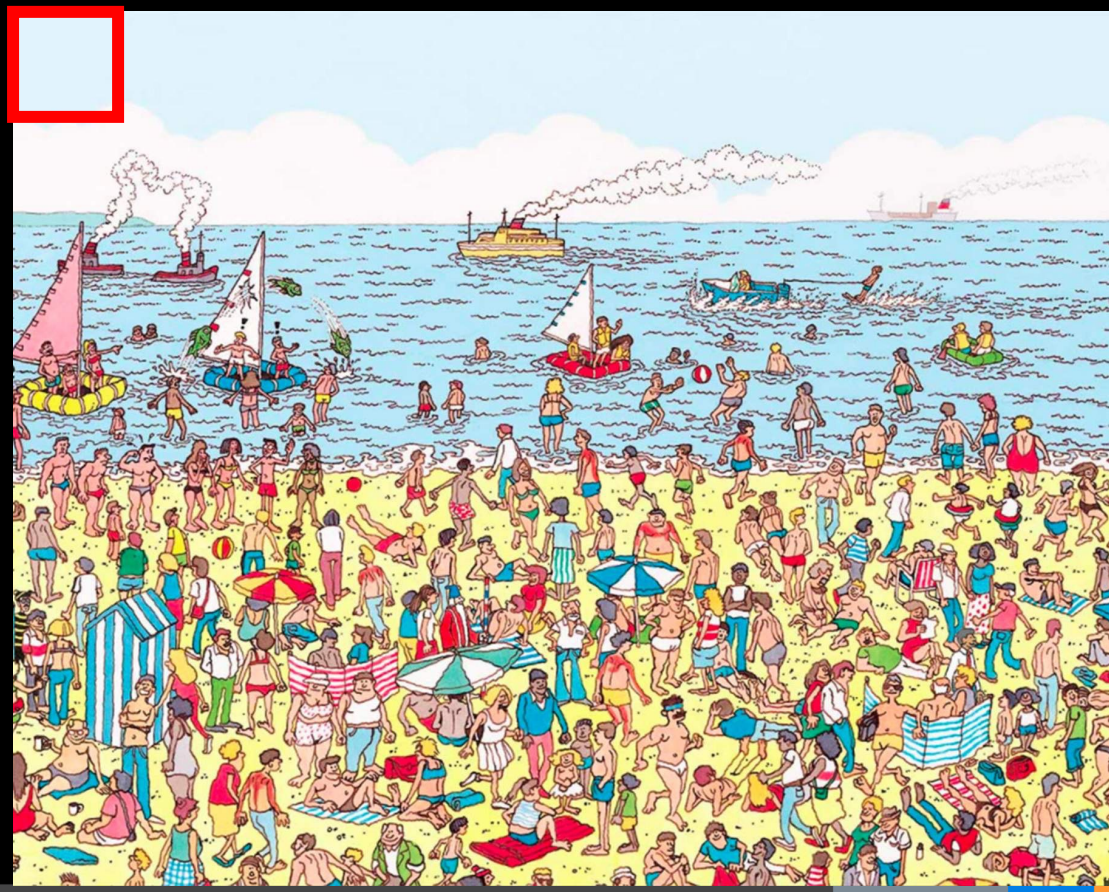
- Looping means repeating something over and over until a particular condition is satisfied.



# Looping (Iterating)

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

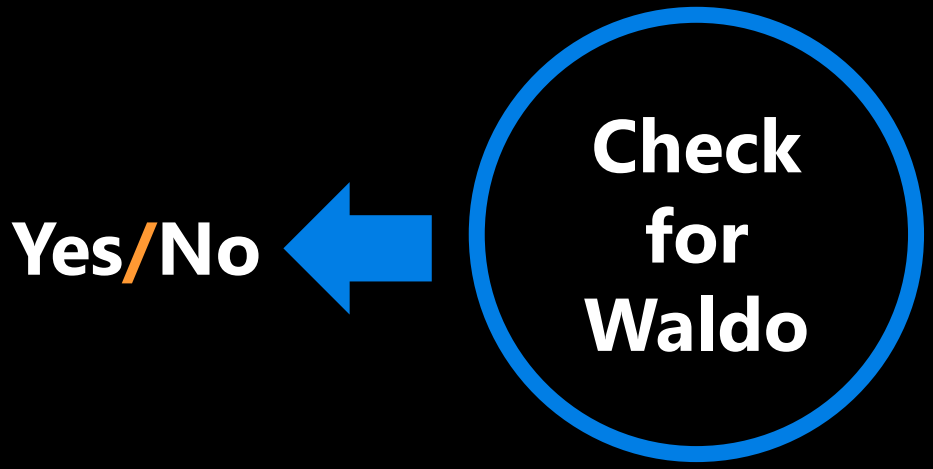


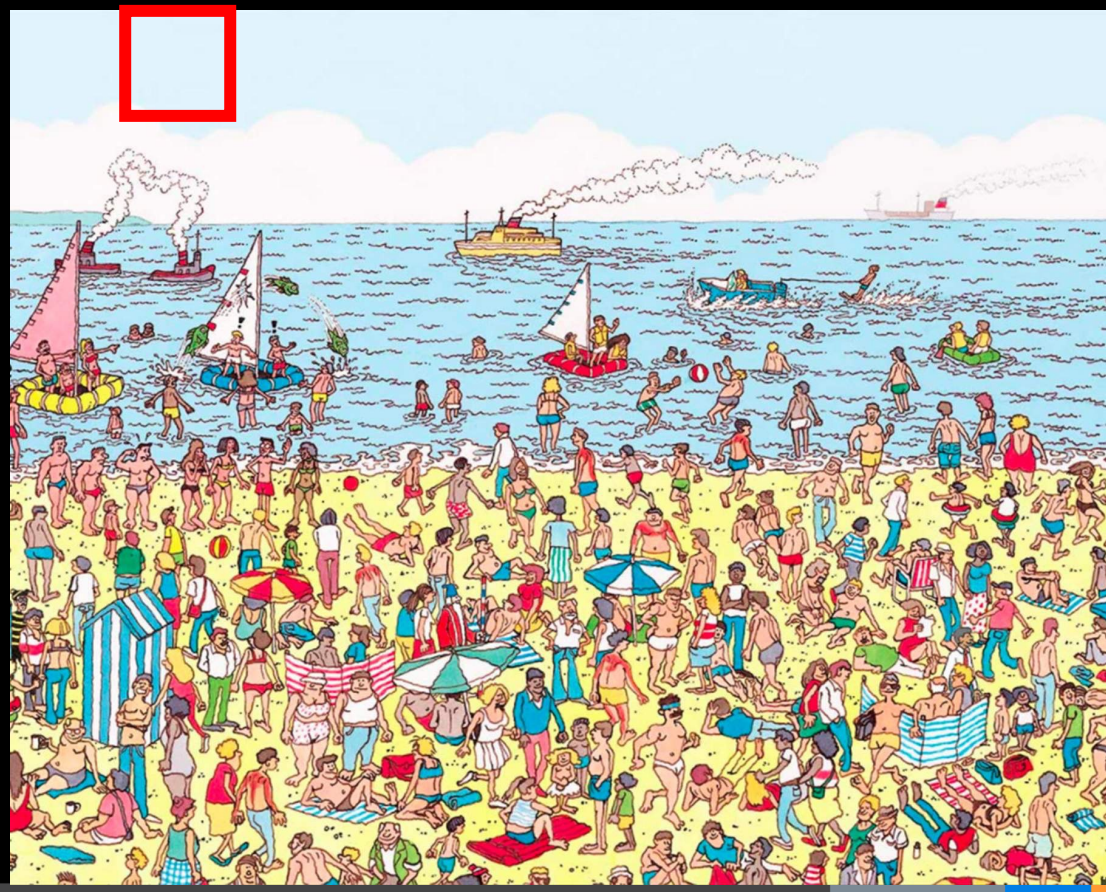



# Looping (Iterating)

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

Looping

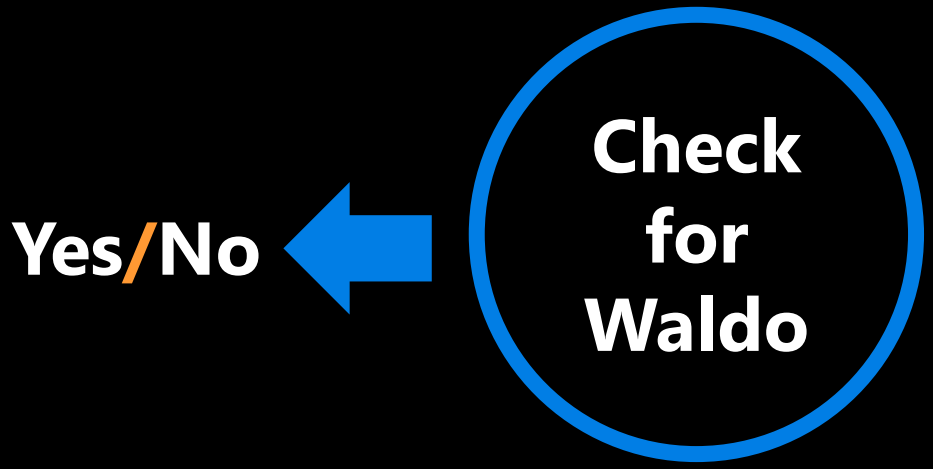


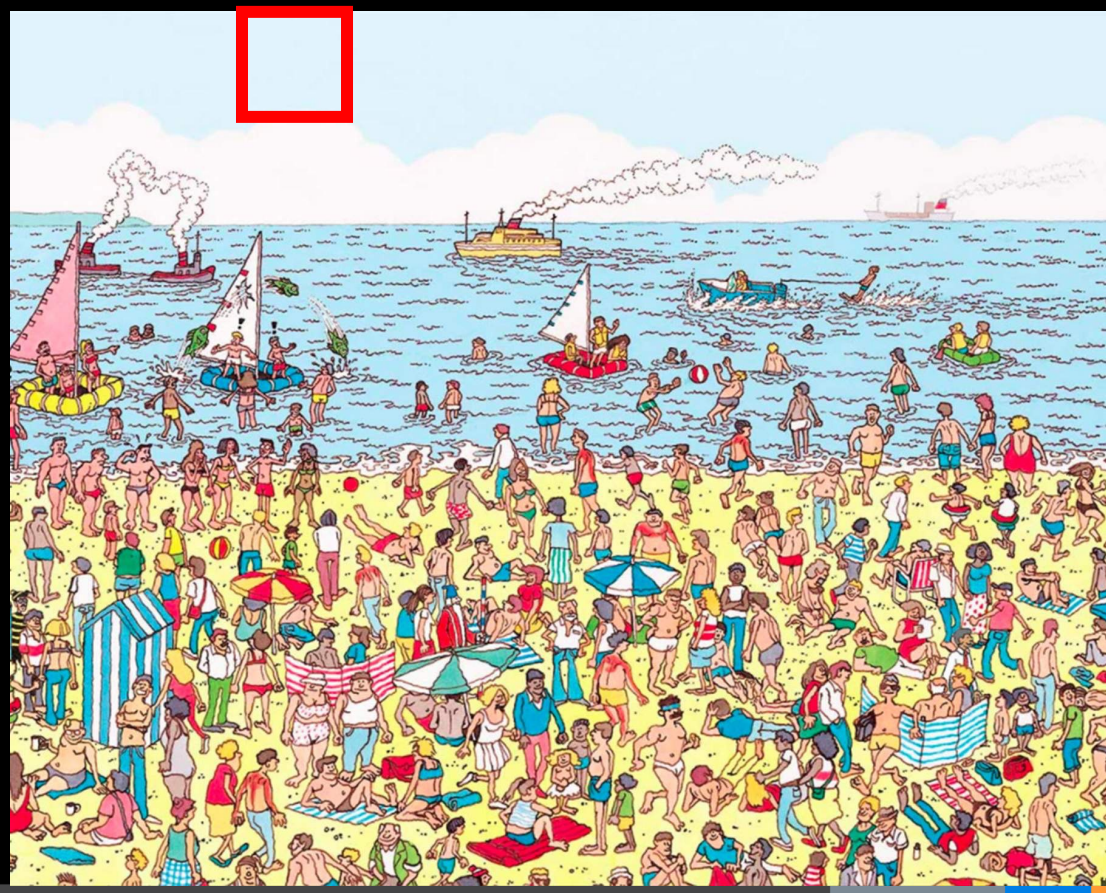



# Looping (Iterating)

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

Looping

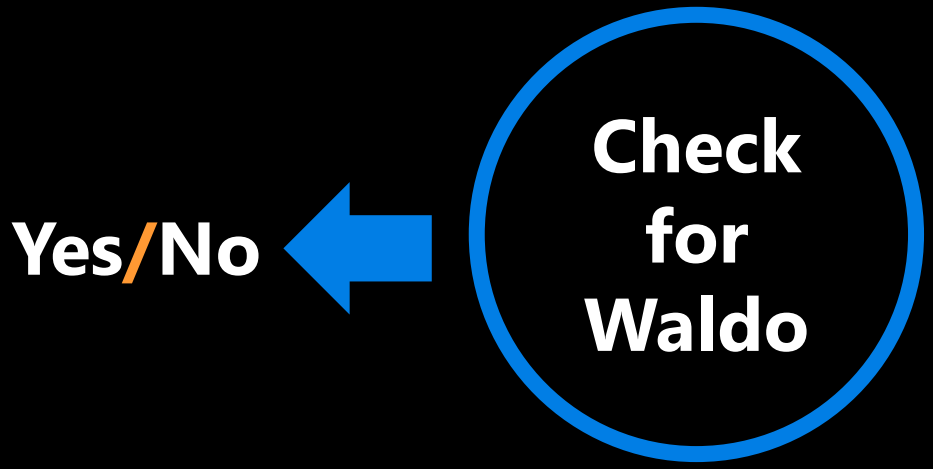


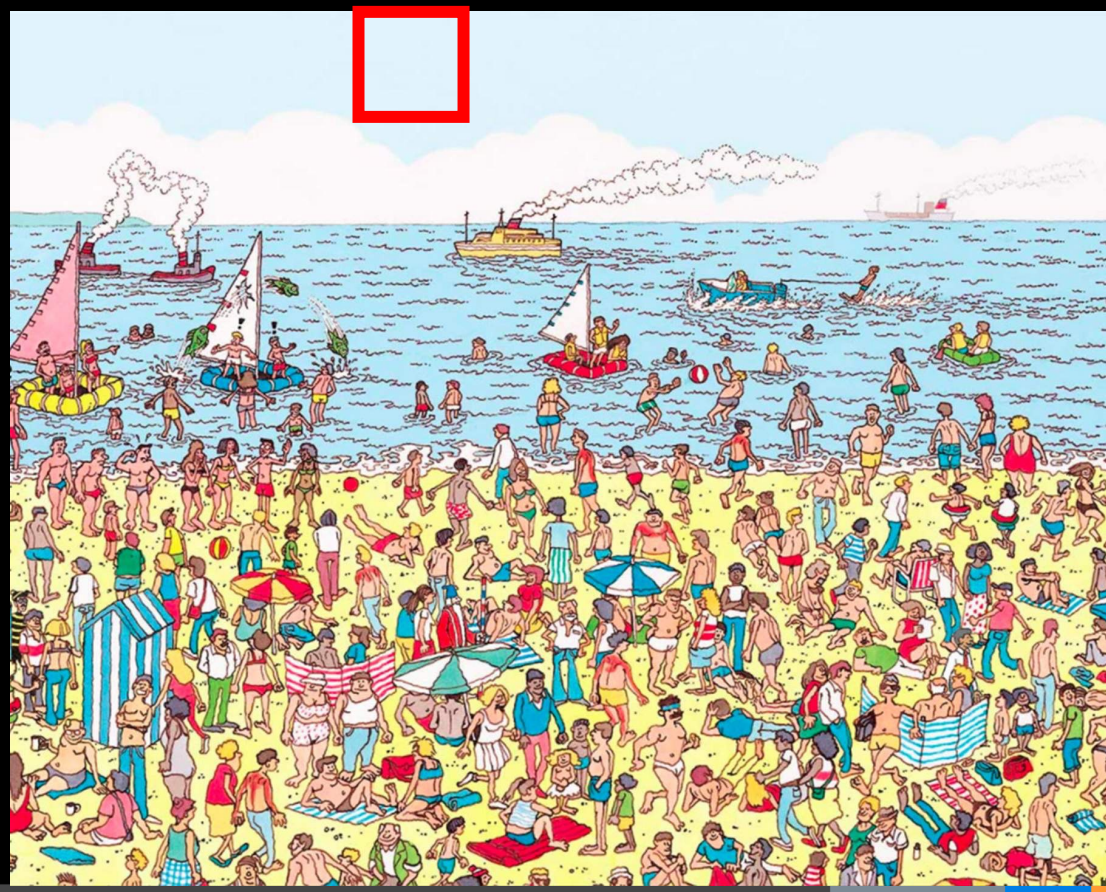



# Looping (Iterating)

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

Looping

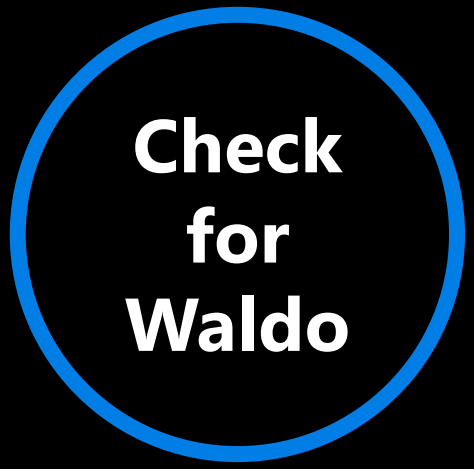


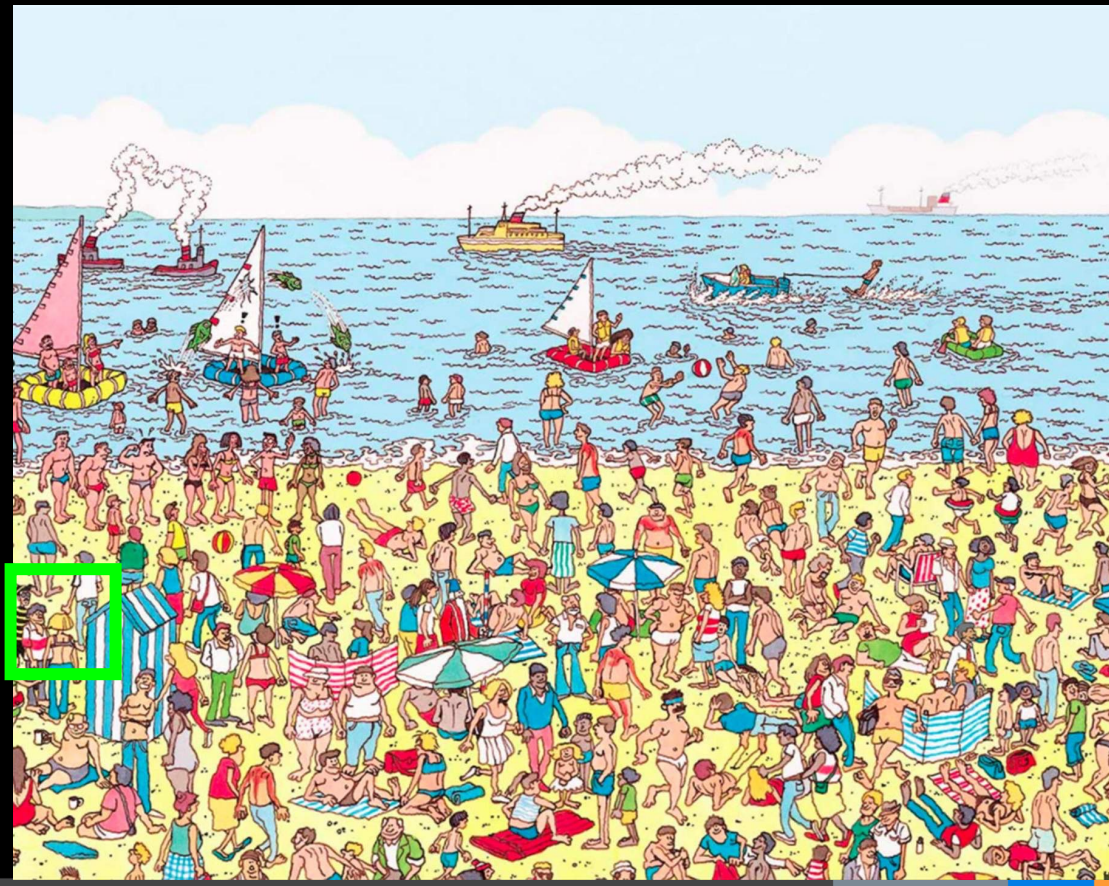



# Looping (Iterating)

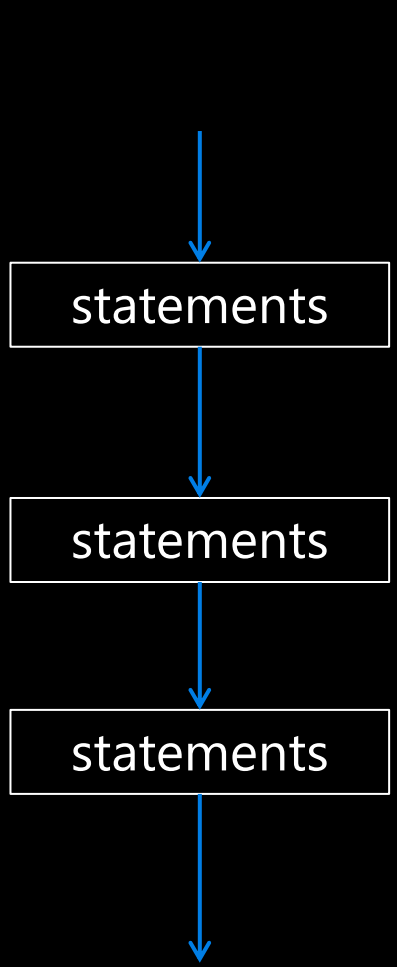
- Looping means repeating something over and over until a particular condition is satisfied.

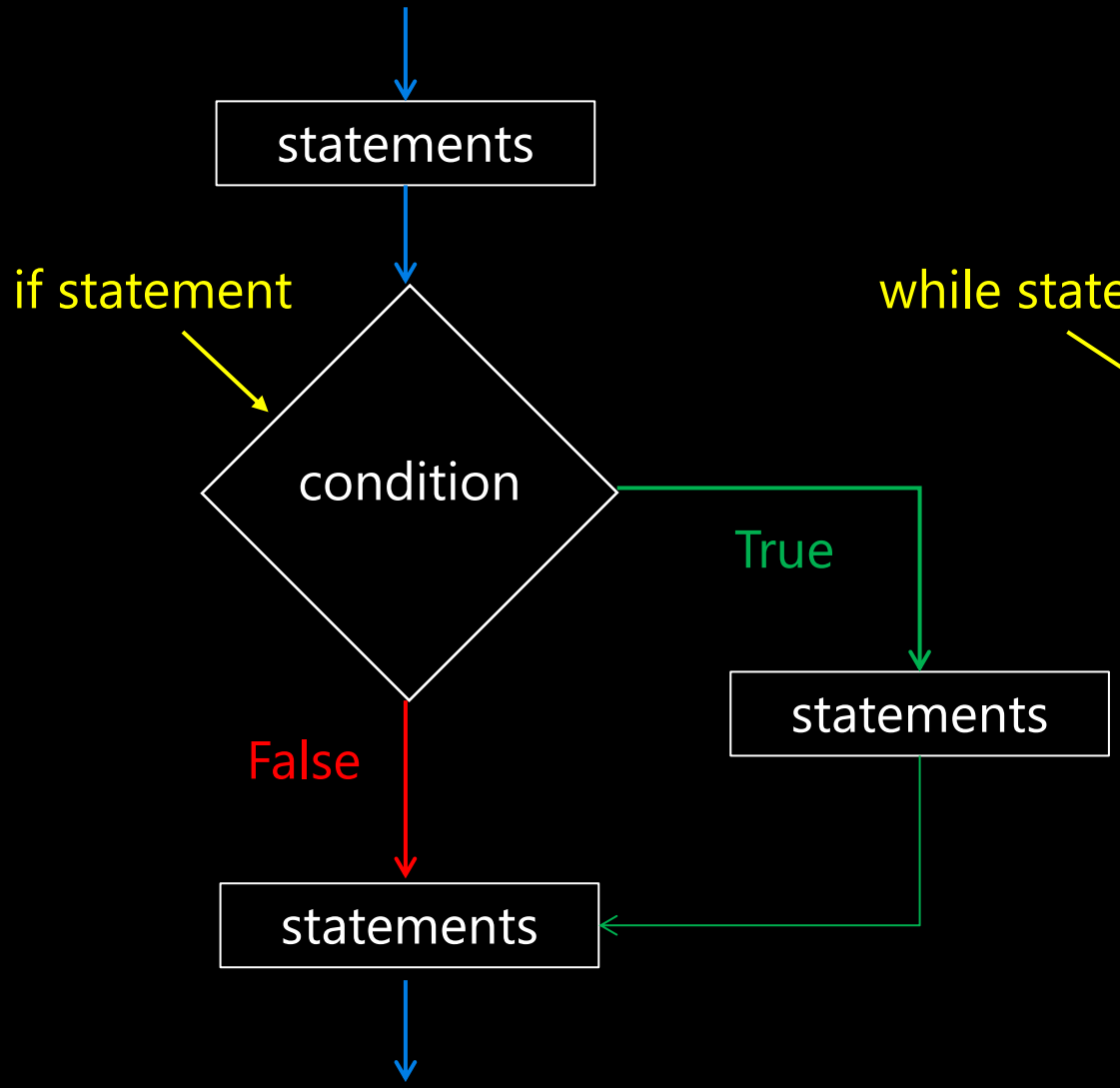
Looping



# Our programming branches so far…

# 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**

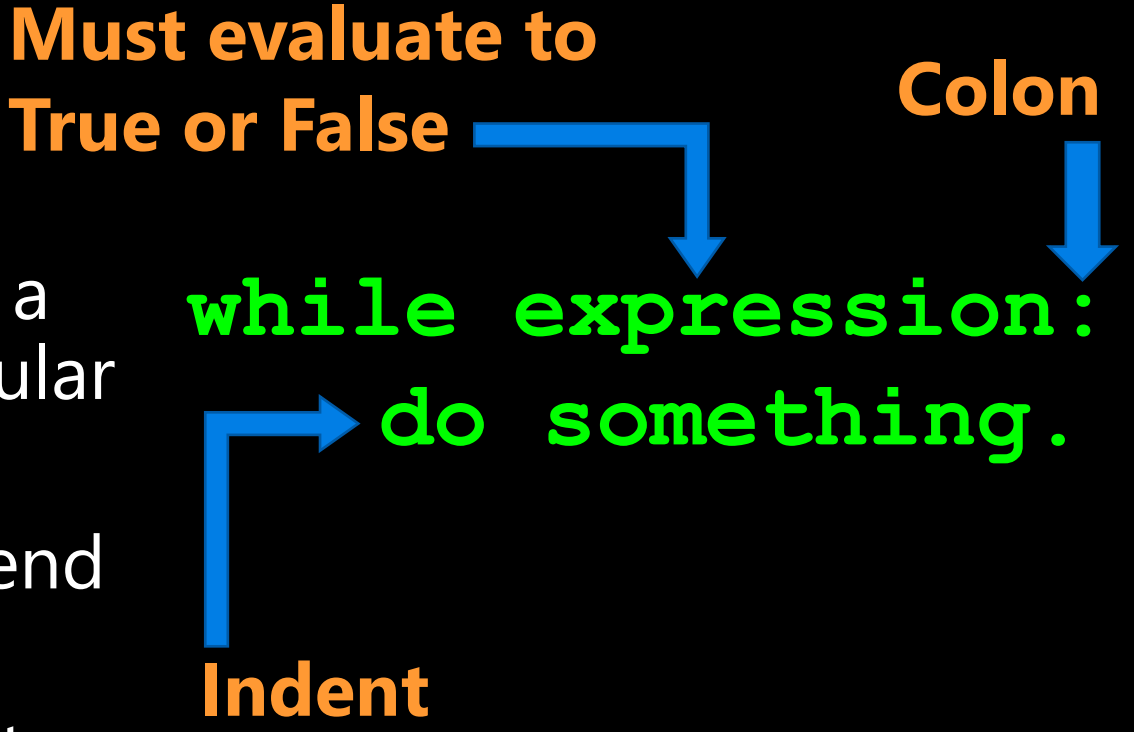
**Click Link:**
**2. Asking the User a Question**

# 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.

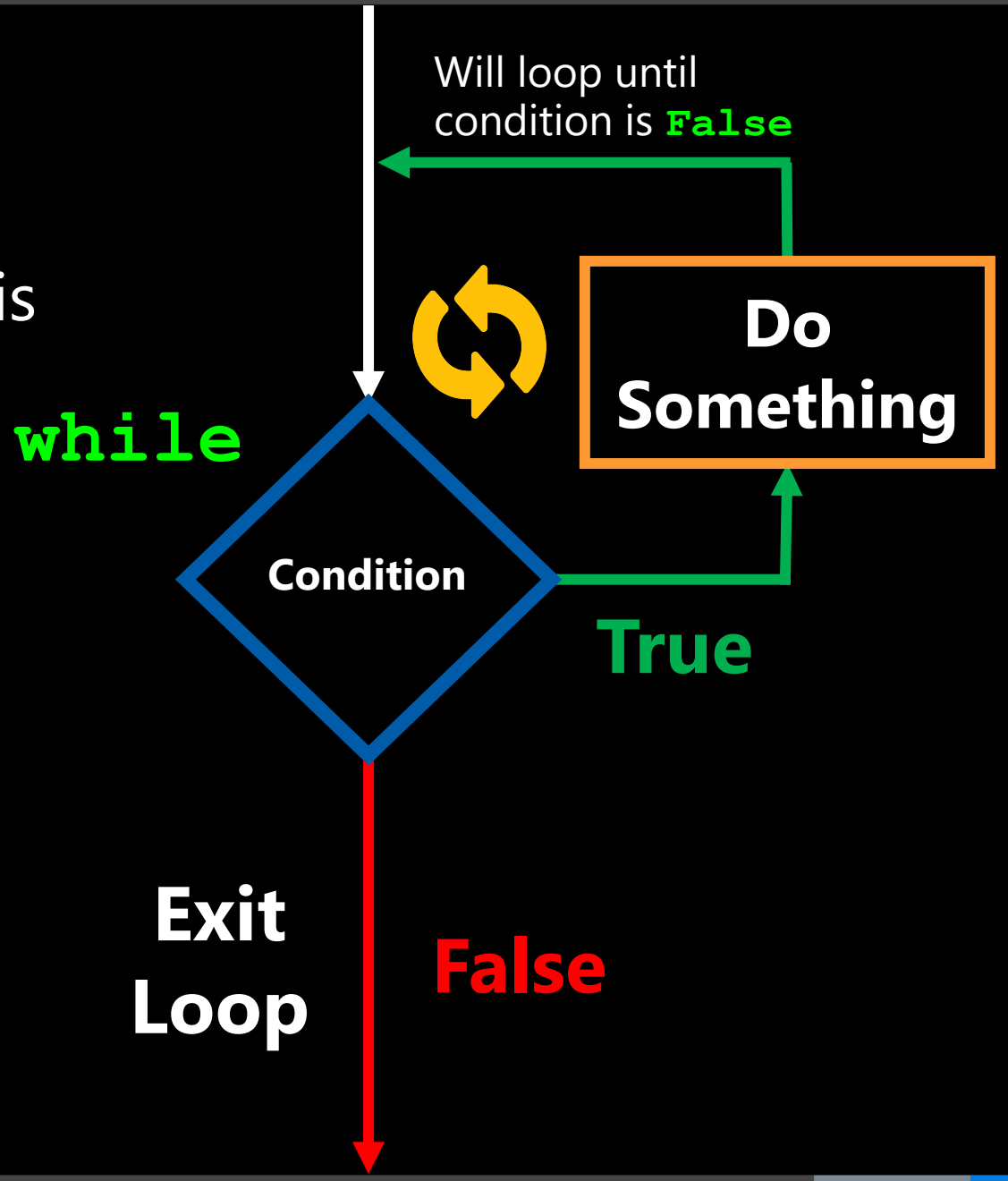
# While Loops

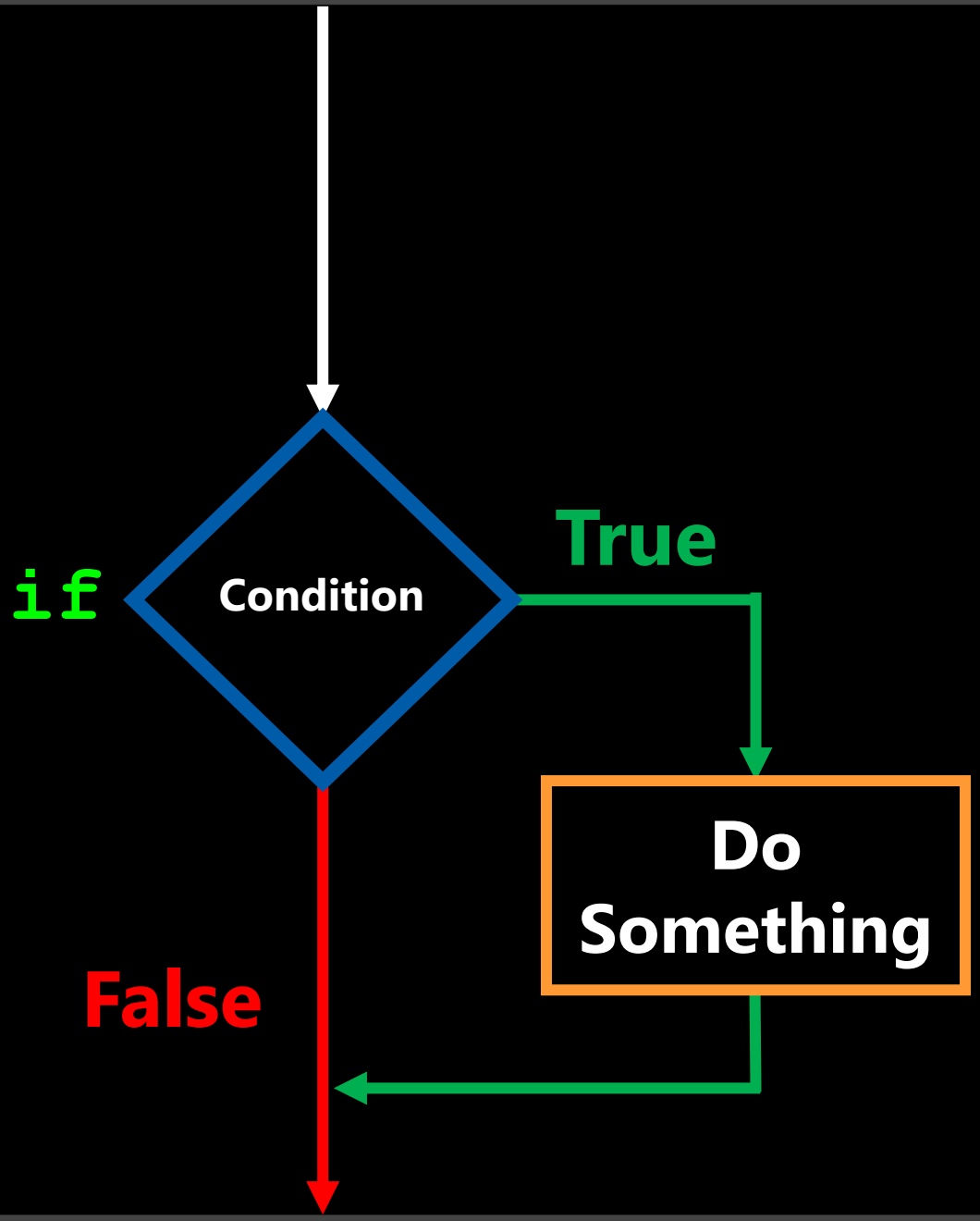
- The **while loop** keeps executing a piece of code as long as a particular condition is **True**.
- There must be a colon (:) at the end of the while statement.
- The action to be performed must be indented.

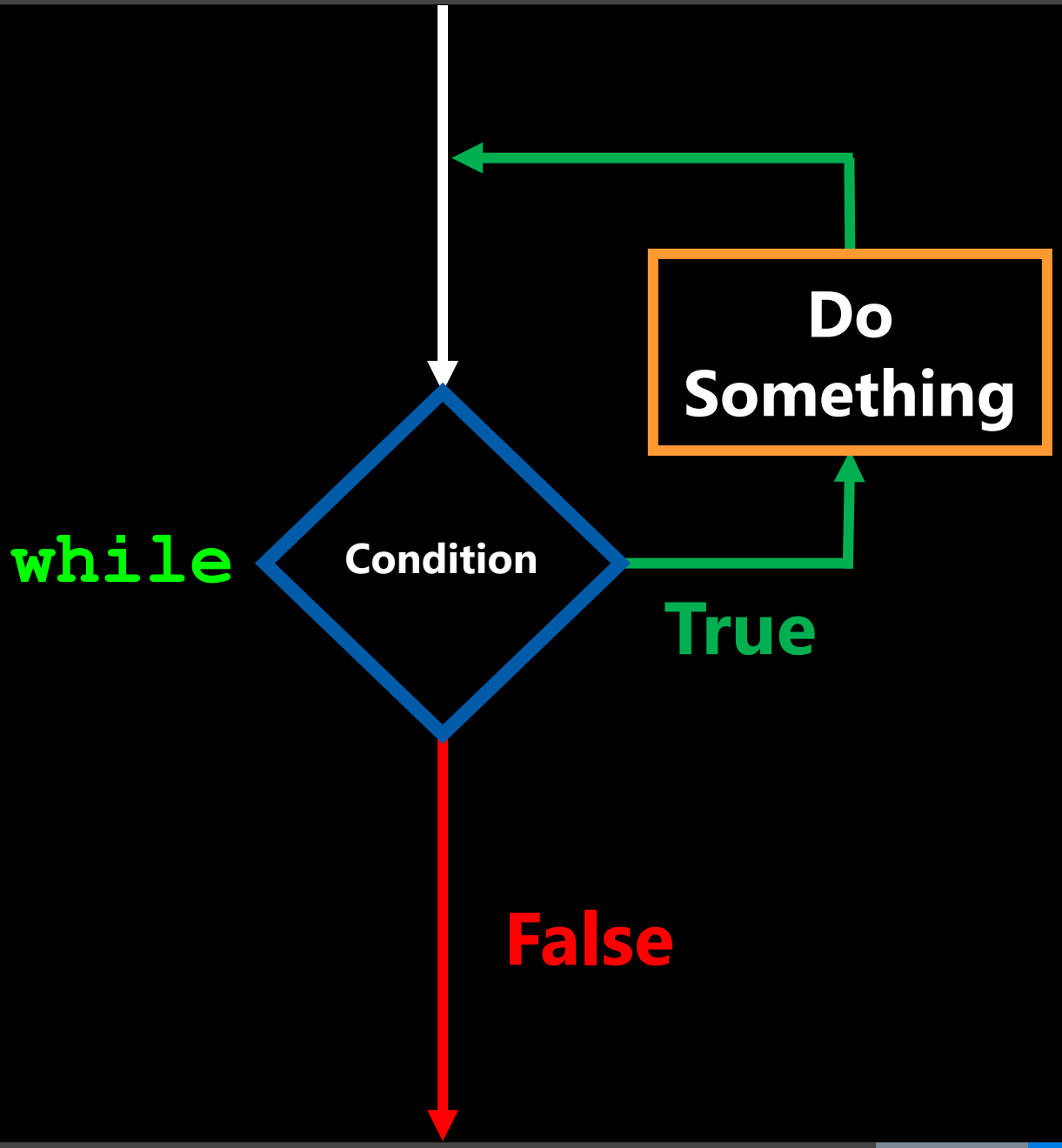


# While Loops

- 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**.

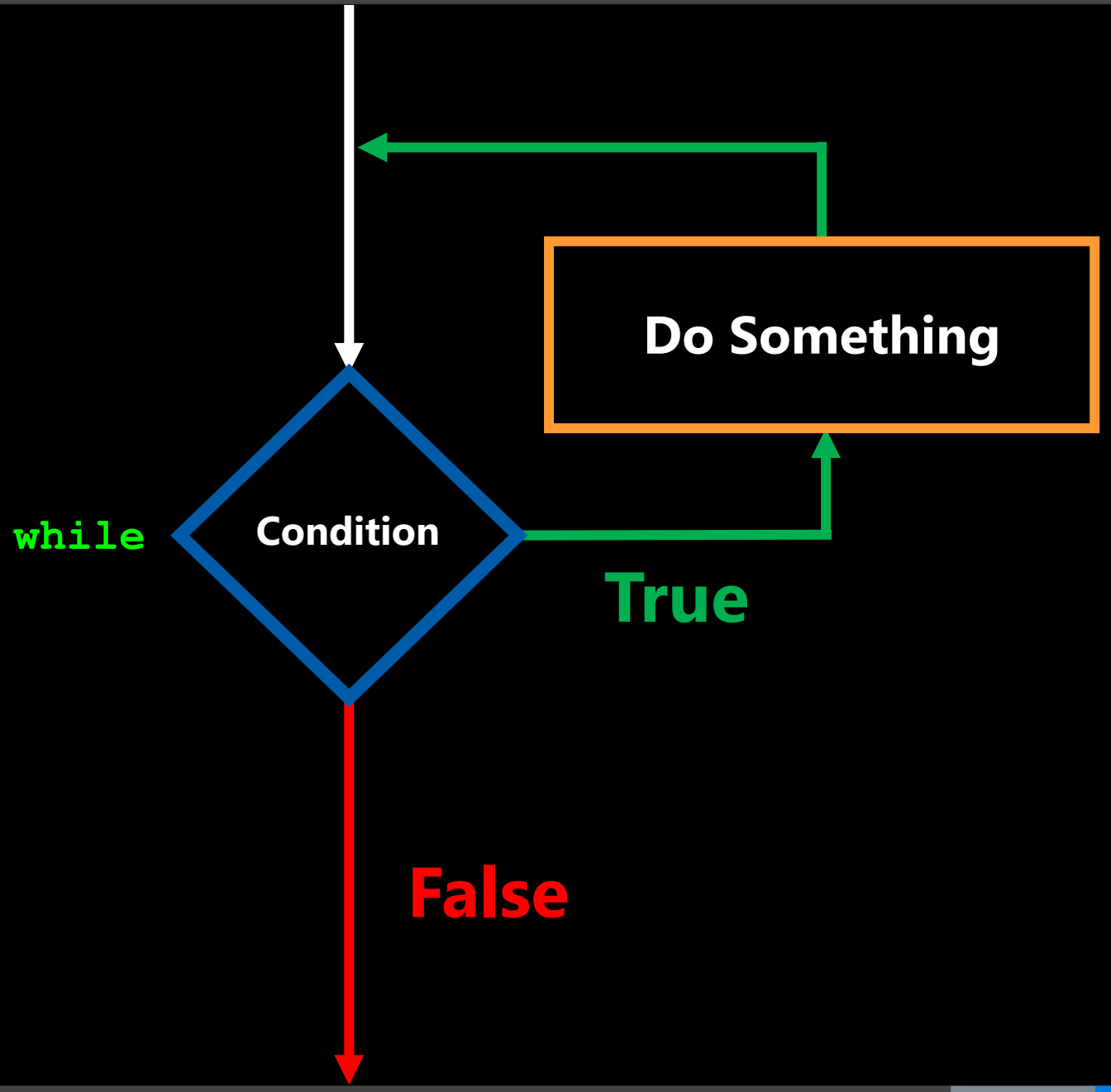



if
Condition
True
Do Something
False


while
Condition
Do Something
True
False

# While Loops

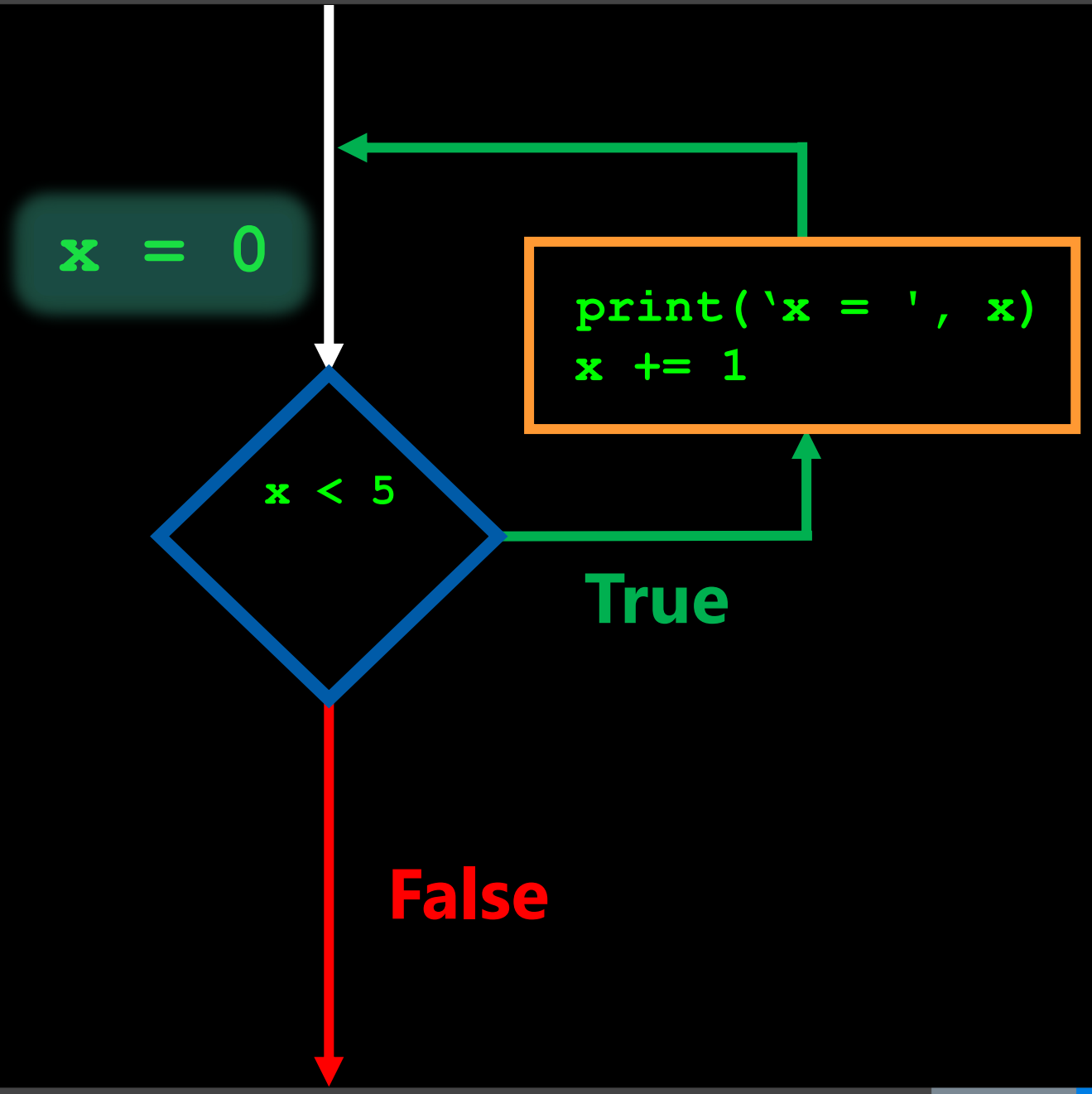
```
x = 0
while x < 5:
    print('x = ', x)
    x += 1
```



# While Loops

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

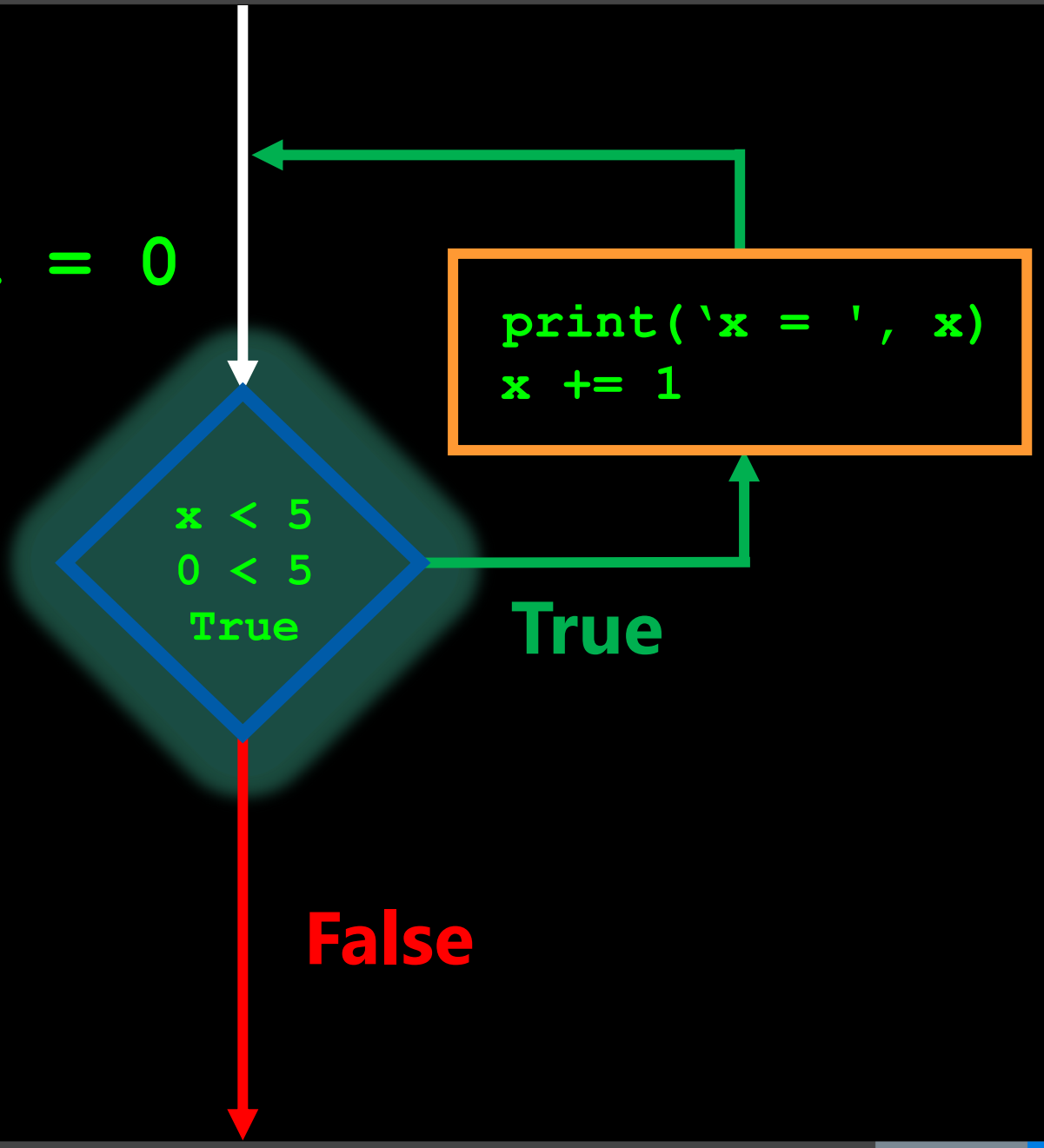
Standard Out.



# While Loops

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

Standard Out.

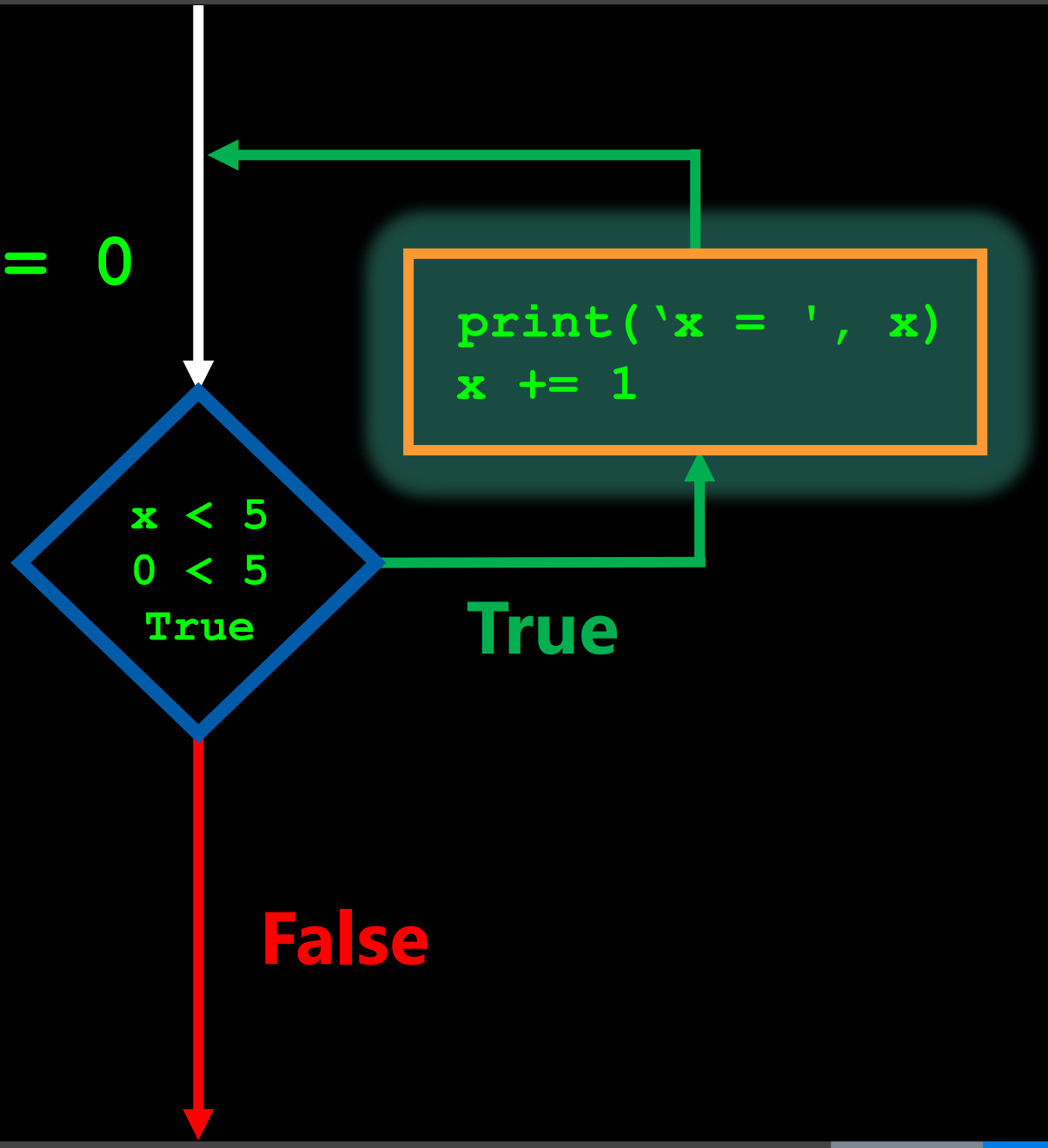


# While Loops

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

Standard Out.

```
x = 0
```

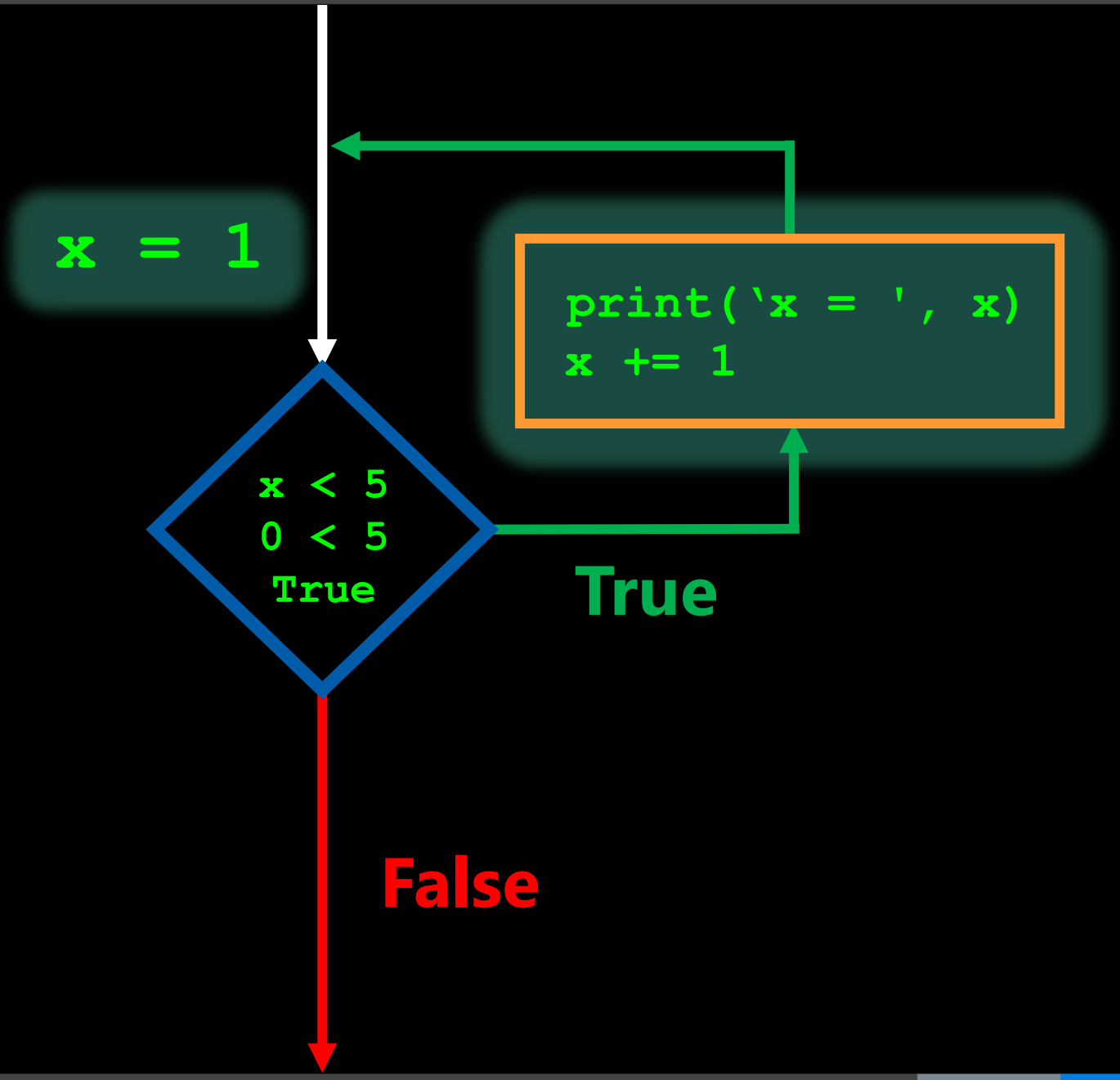


# While Loops

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

Standard Out.

```
x = 0
```

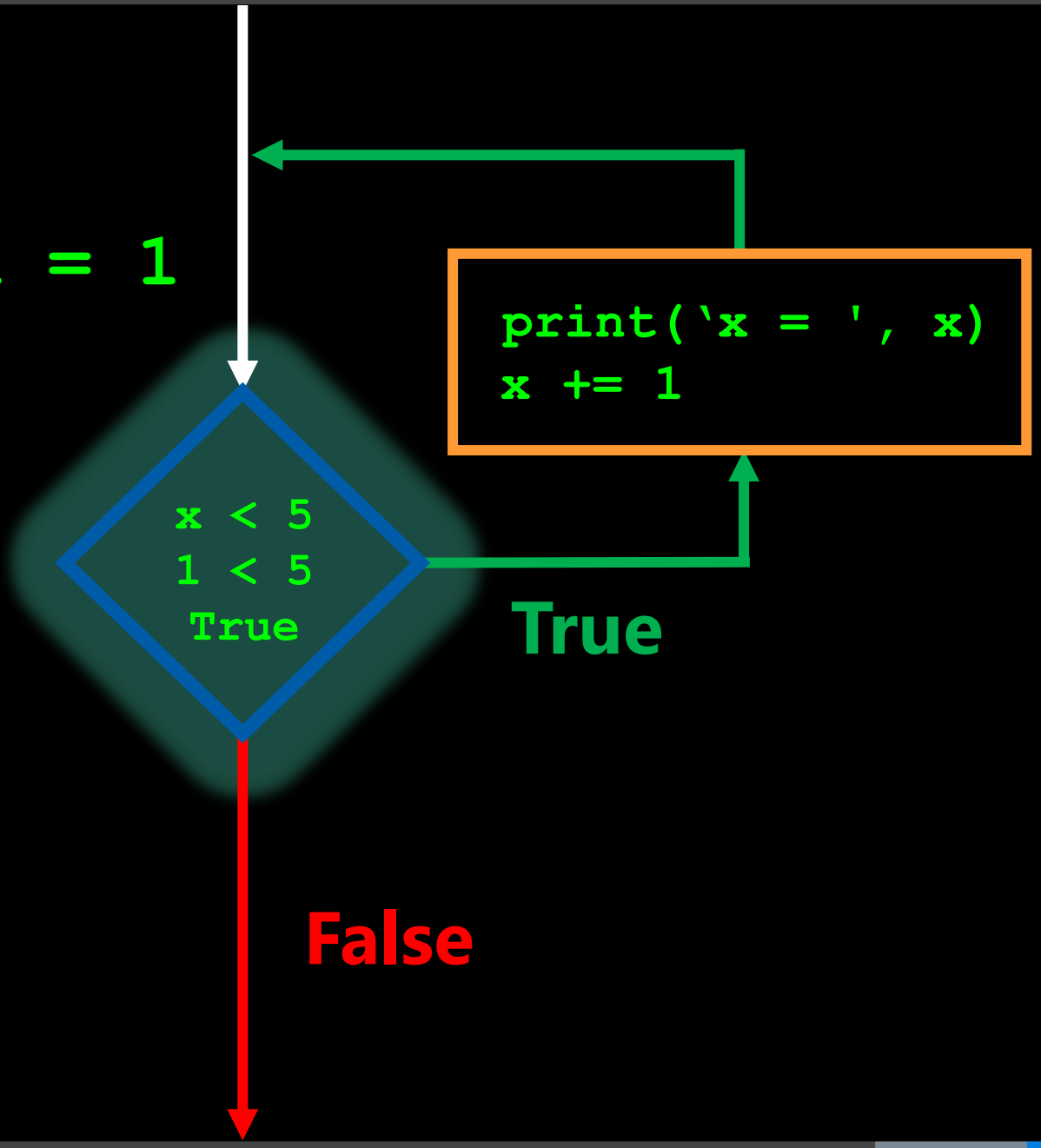


# While Loops

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

Standard Out.

```
x = 0
```

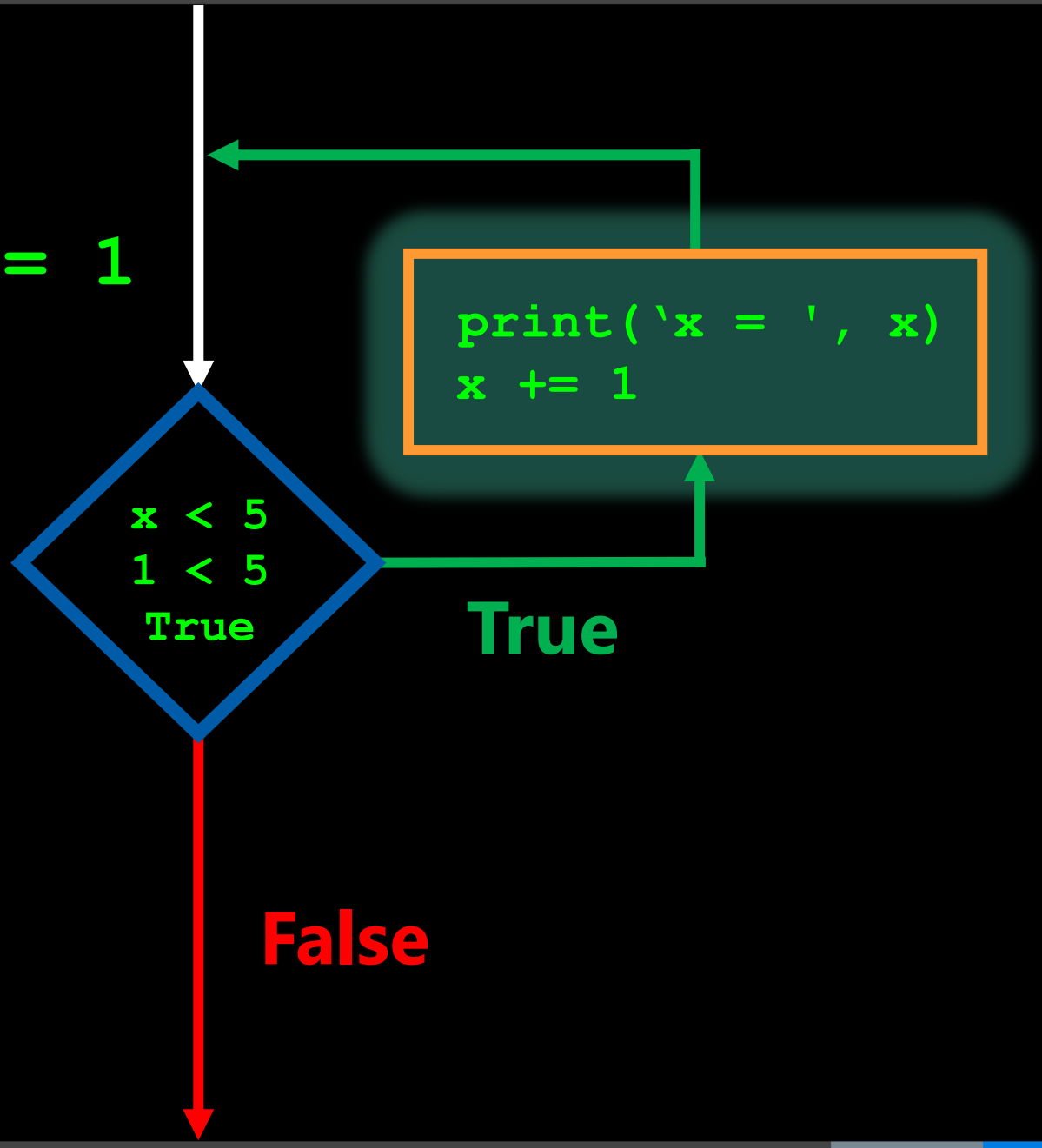


# While Loops

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

Standard Out.

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

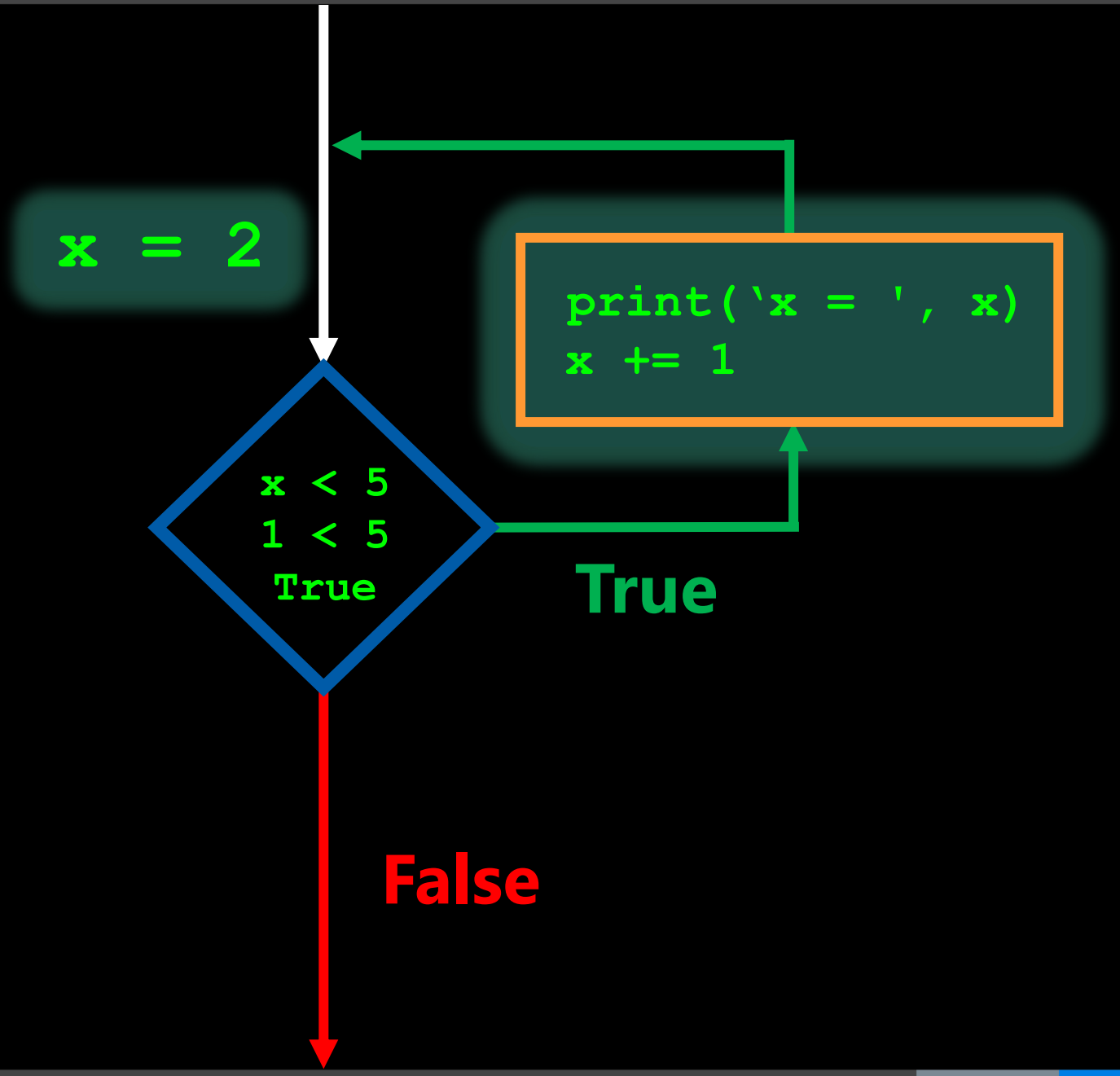


# While Loops

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

Standard Out.

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

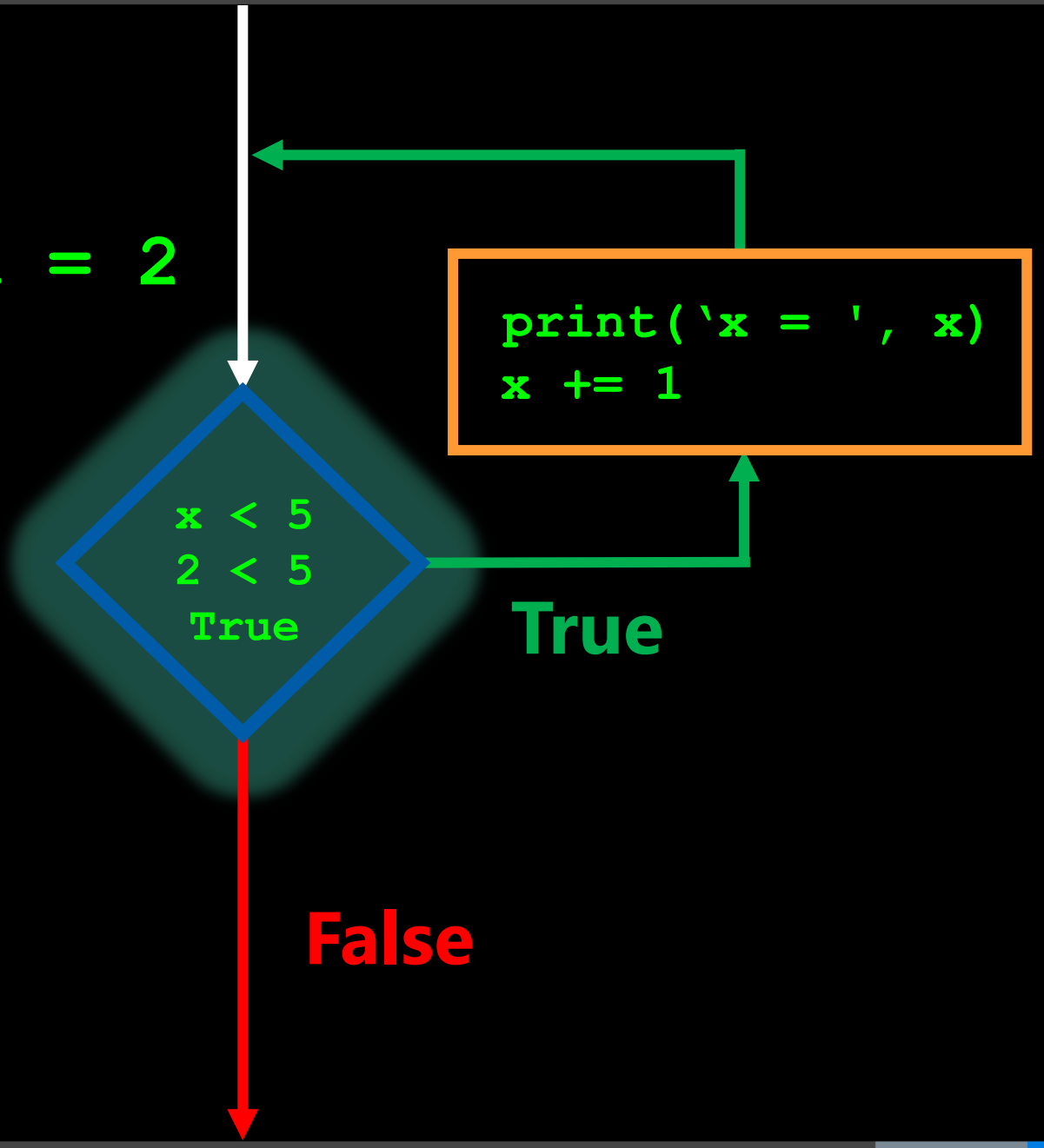


# While Loops

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

Standard Out.

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

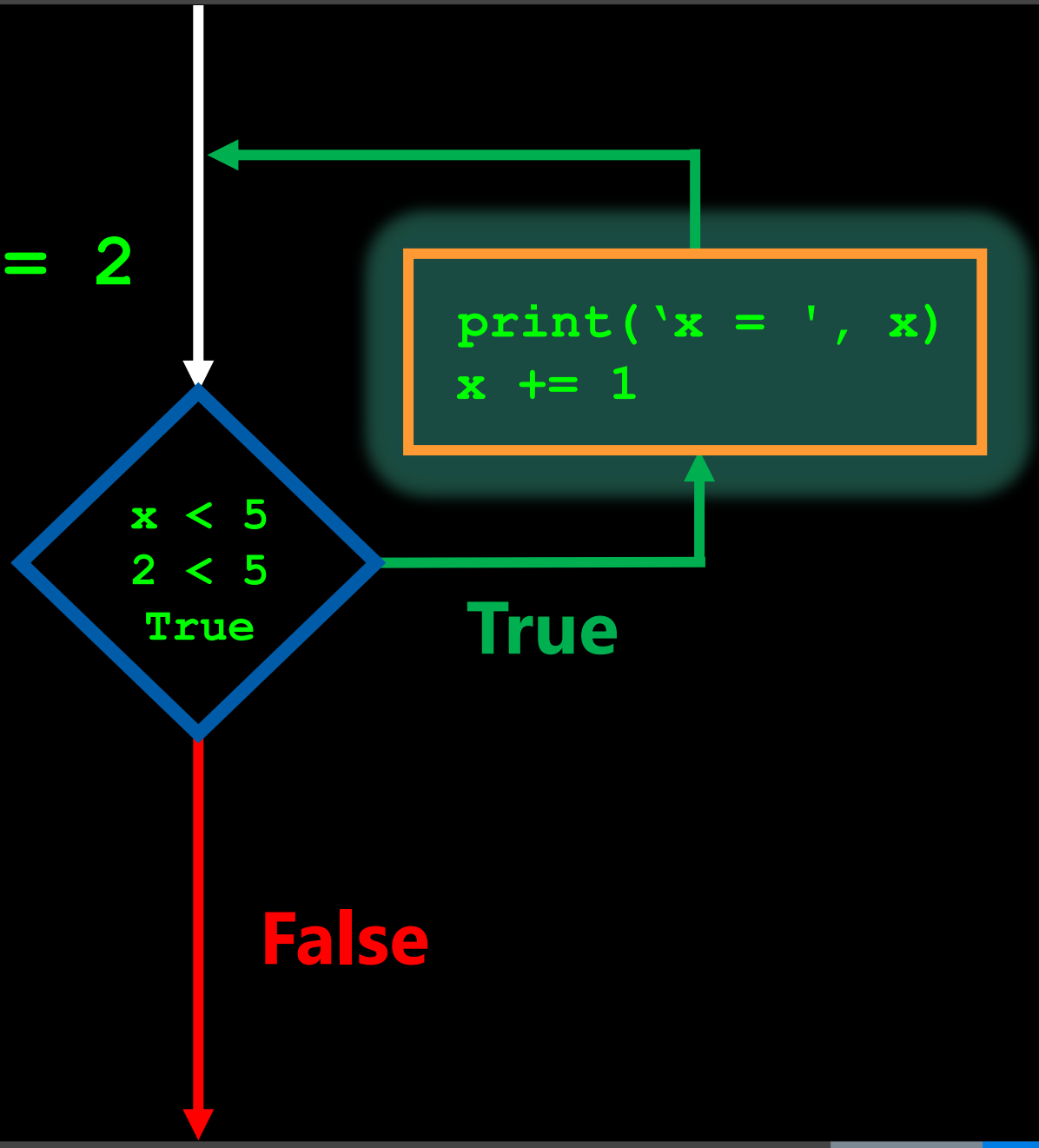


# While Loops

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

Standard Out.

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

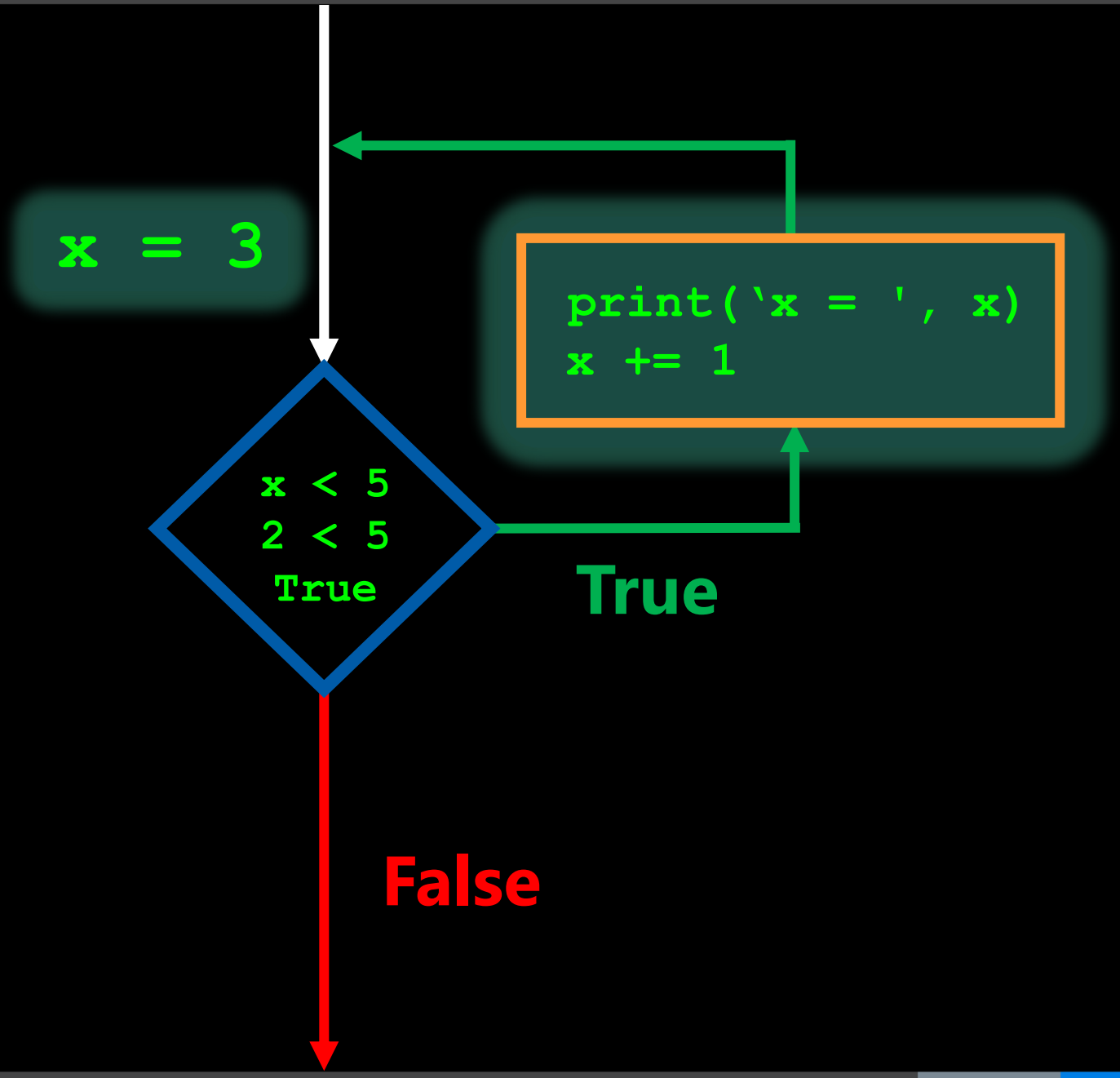


# While Loops

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

Standard Out.

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

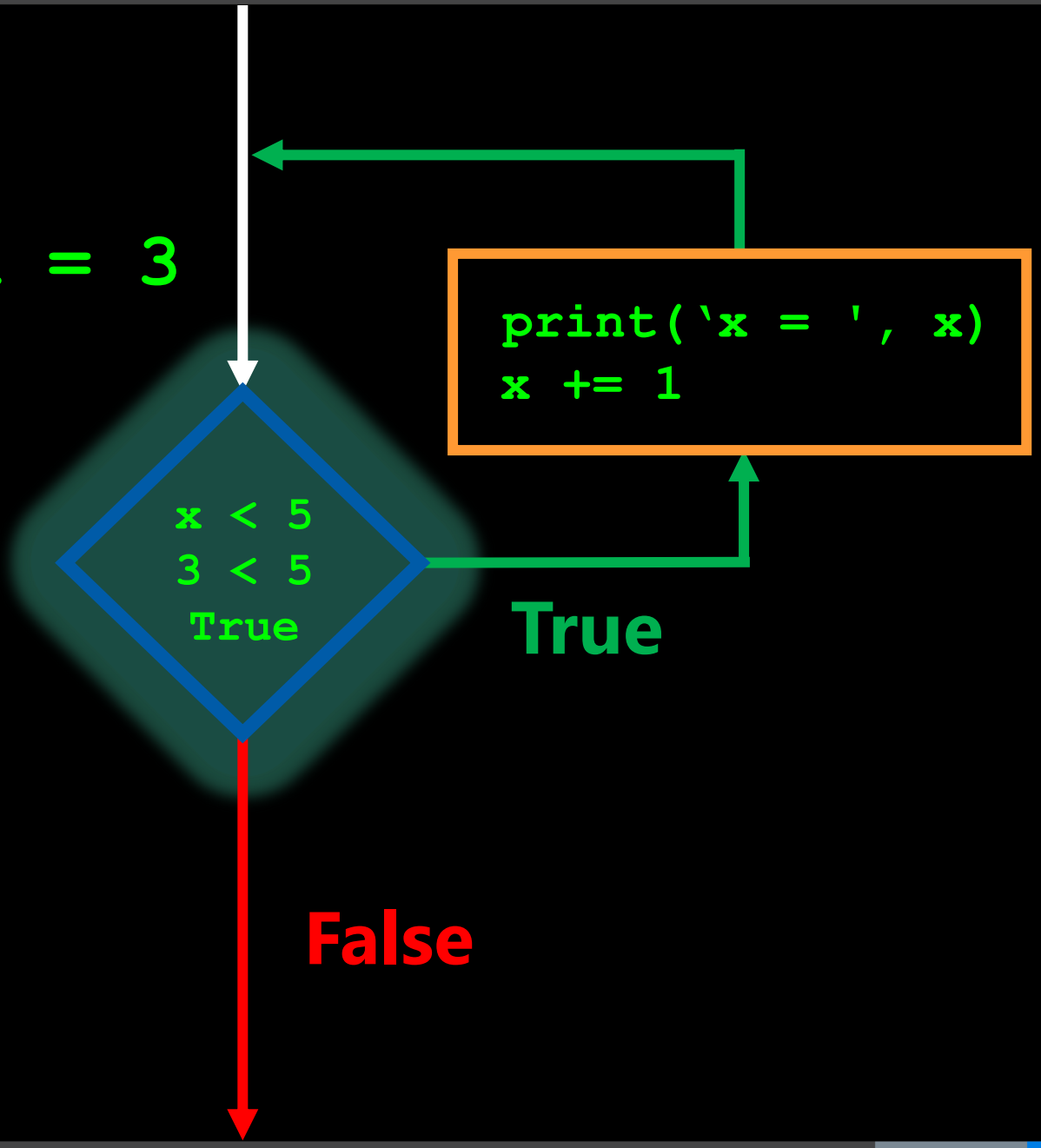


# While Loops

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

Standard Out.

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

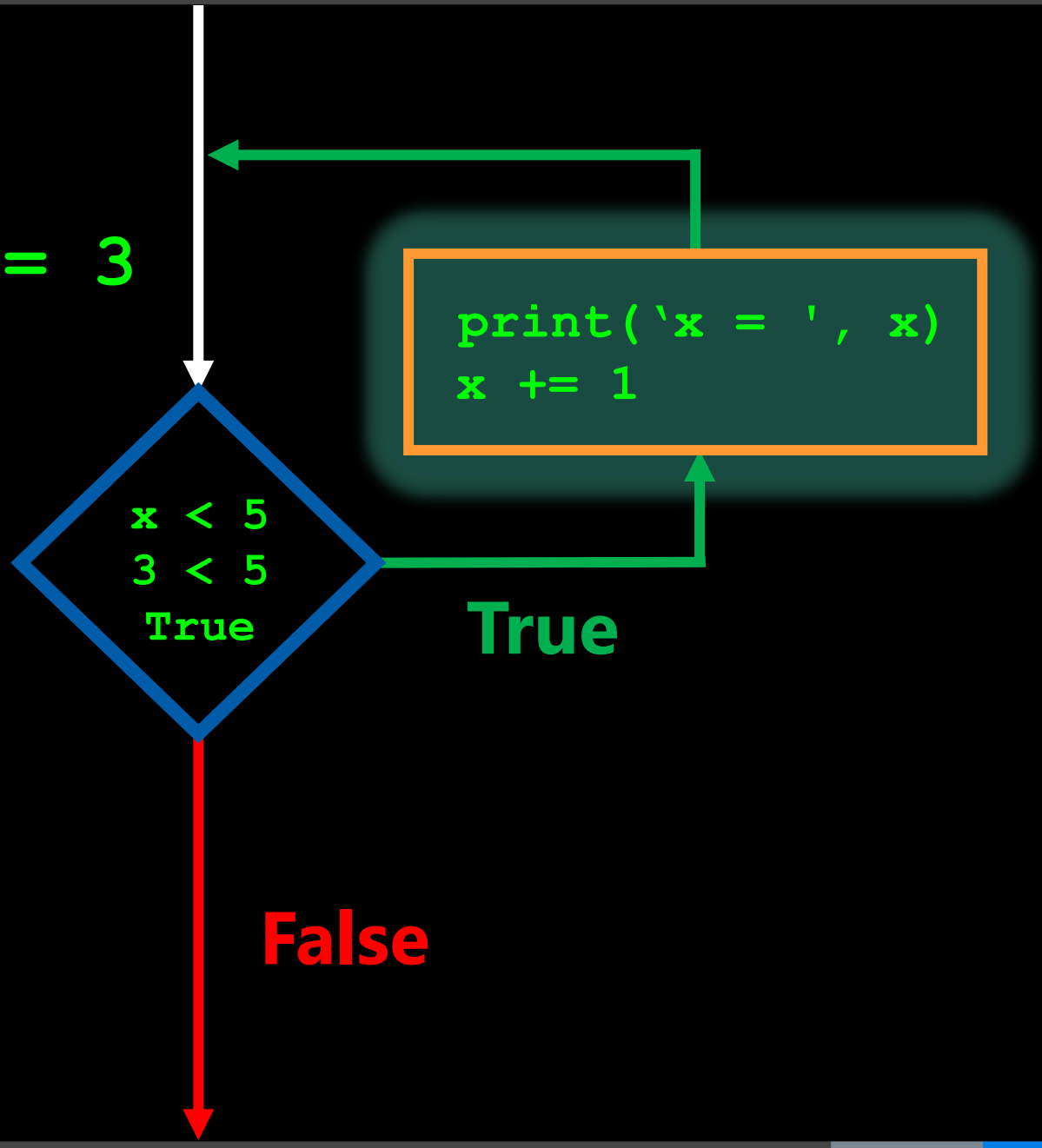


# While Loops

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

Standard Out.

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

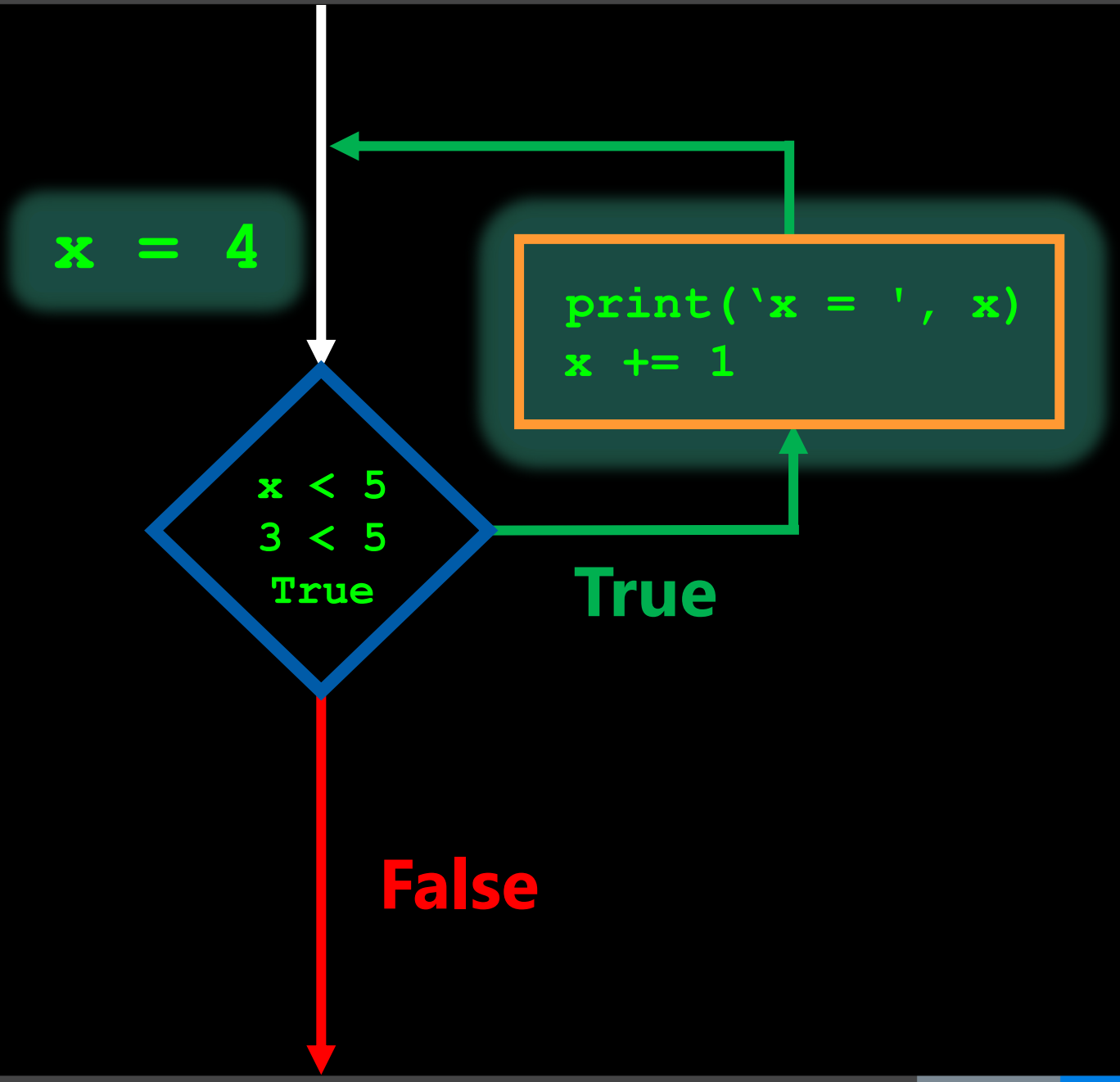


# While Loops

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

Standard Out.

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

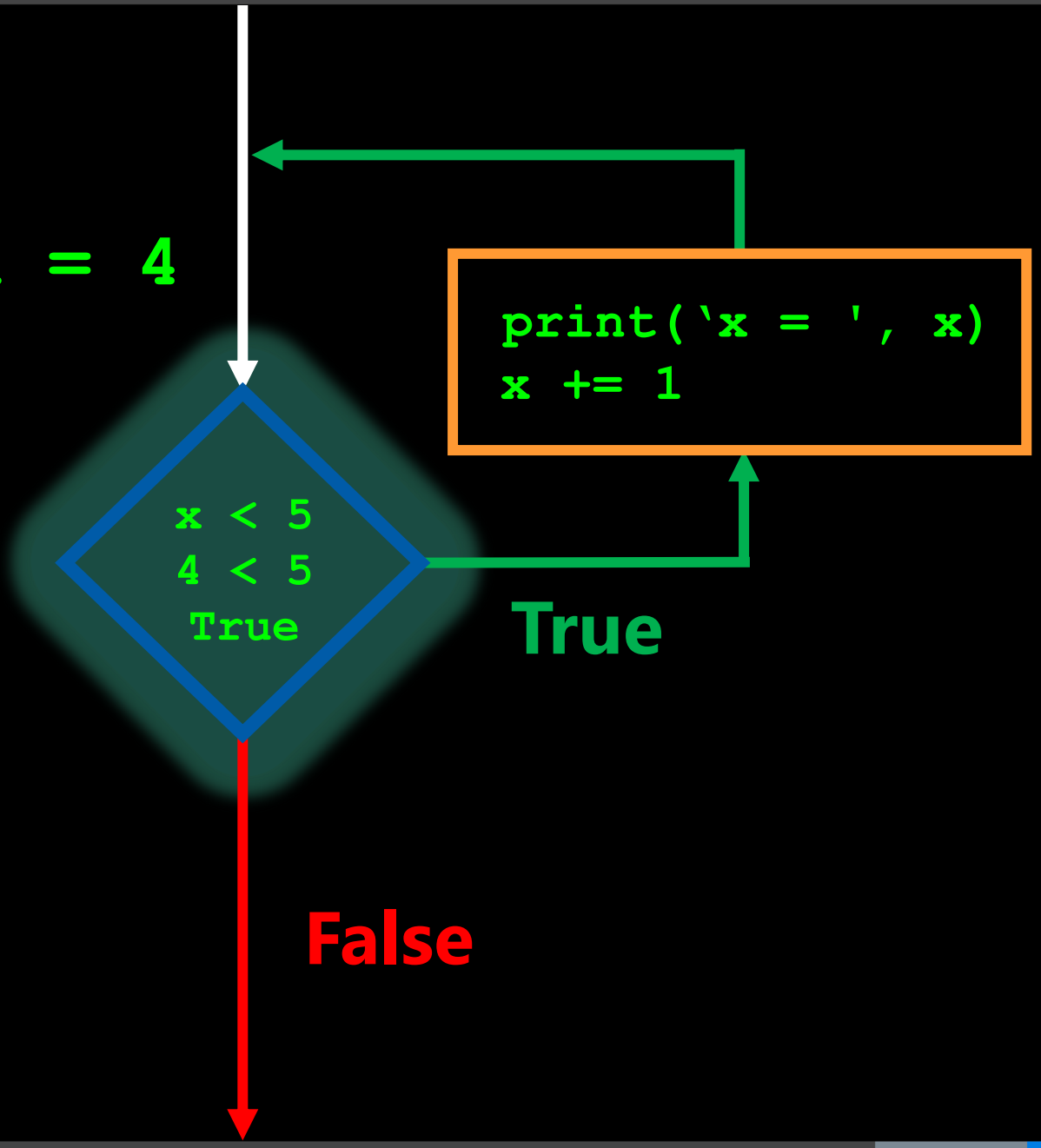


# While Loops

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

Standard Out.

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

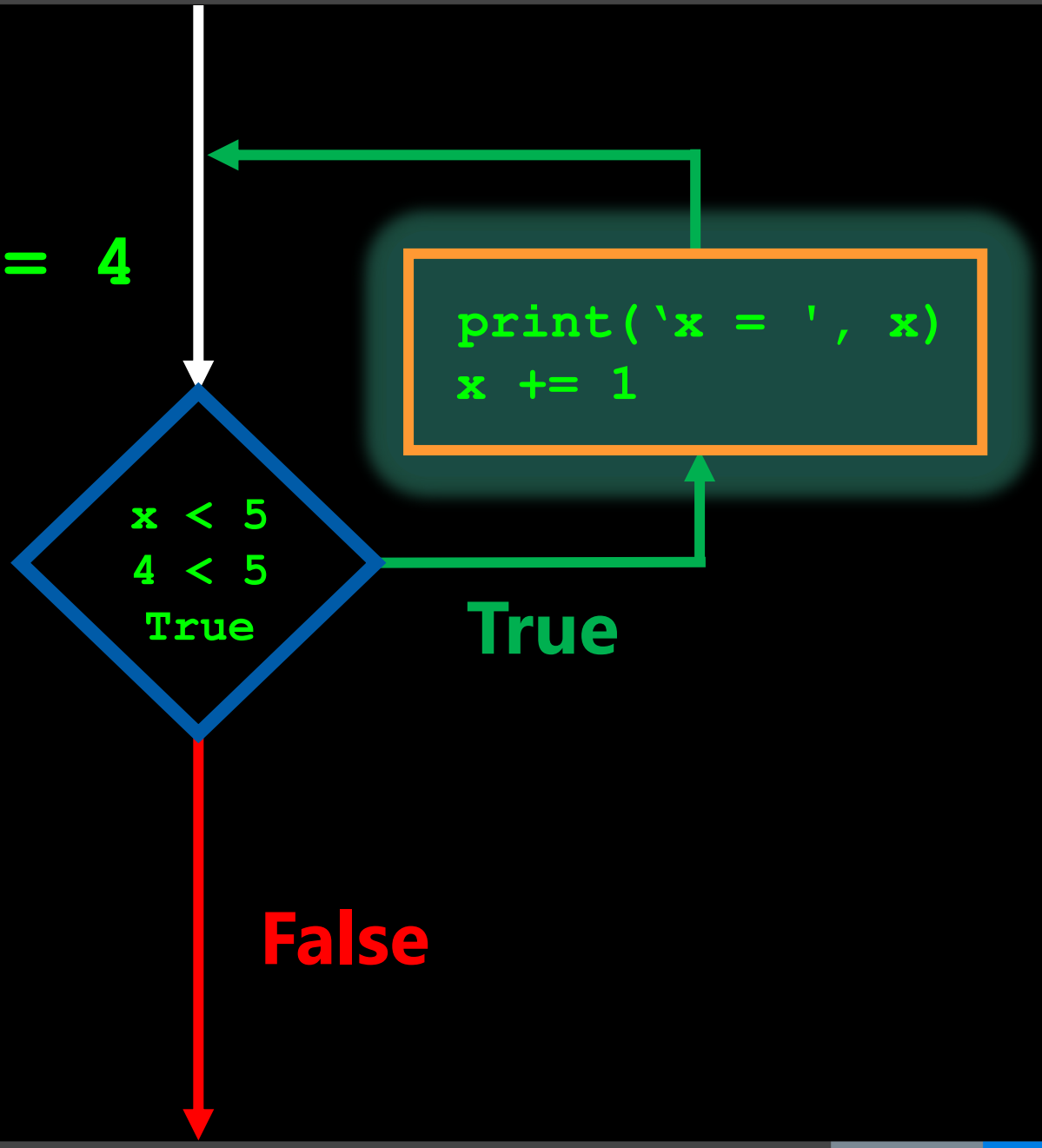


# While Loops

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

Standard Out.

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

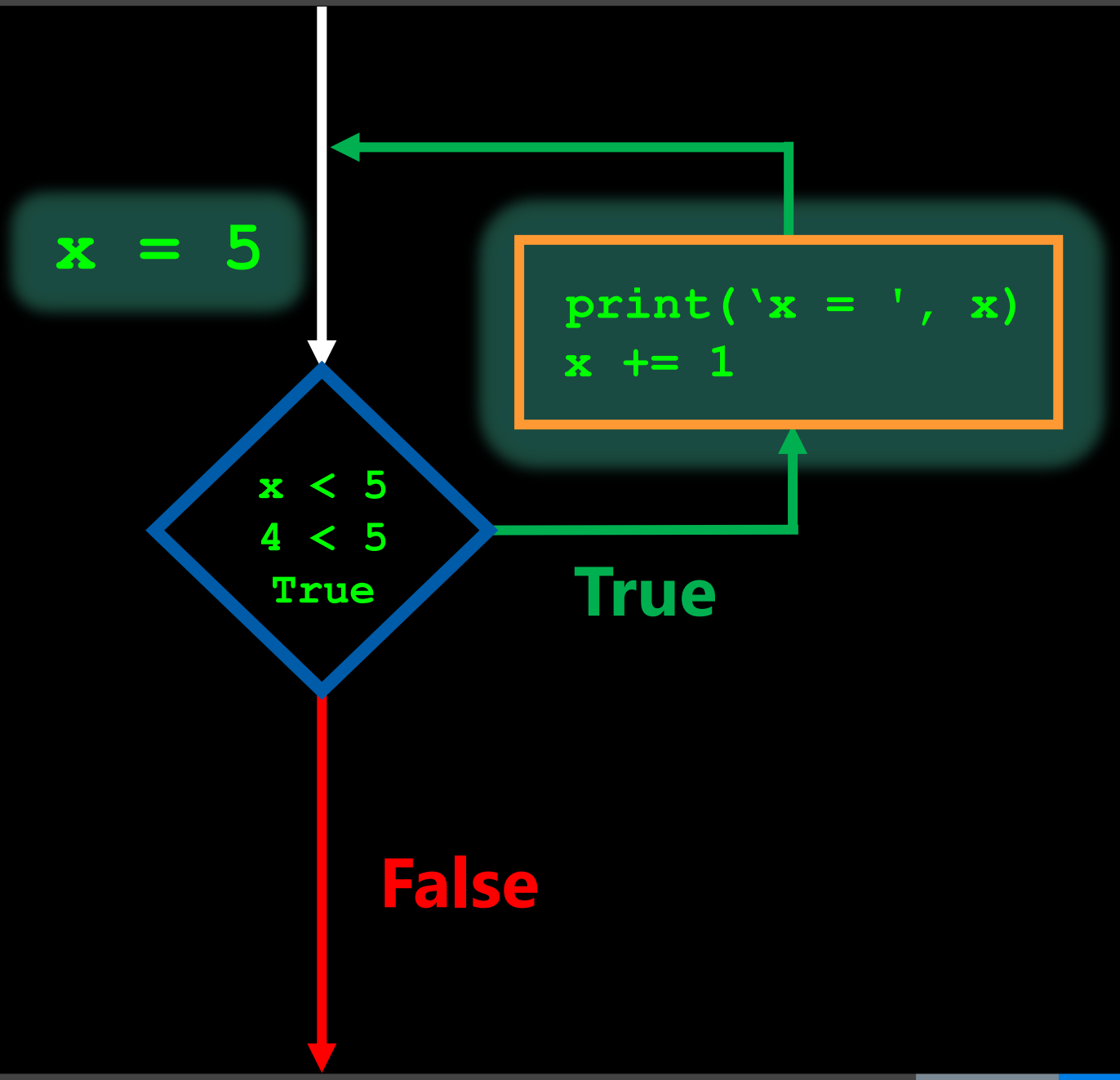


# While Loops

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

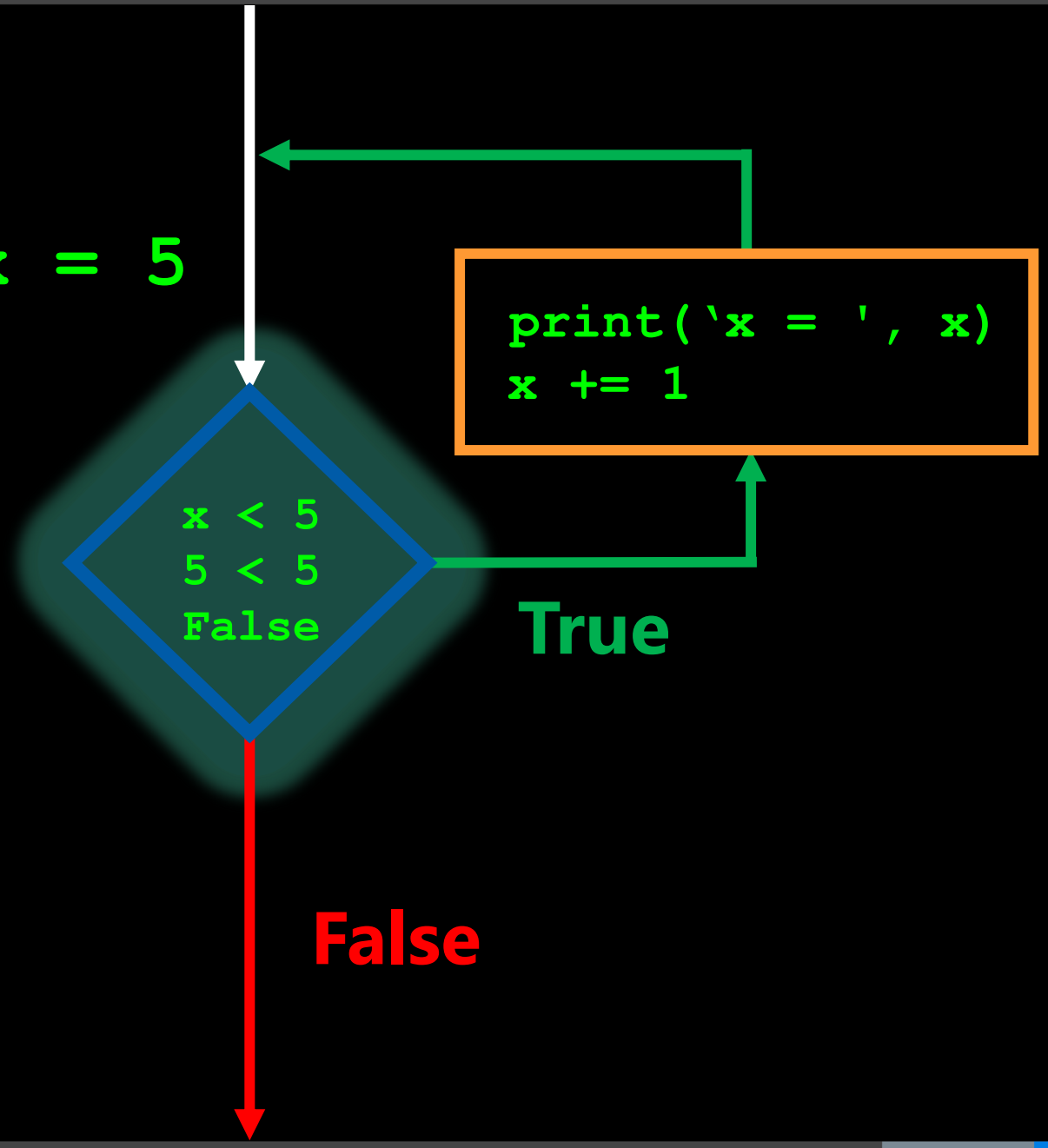
Standard Out.

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



# While Loops

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




Standard Out.

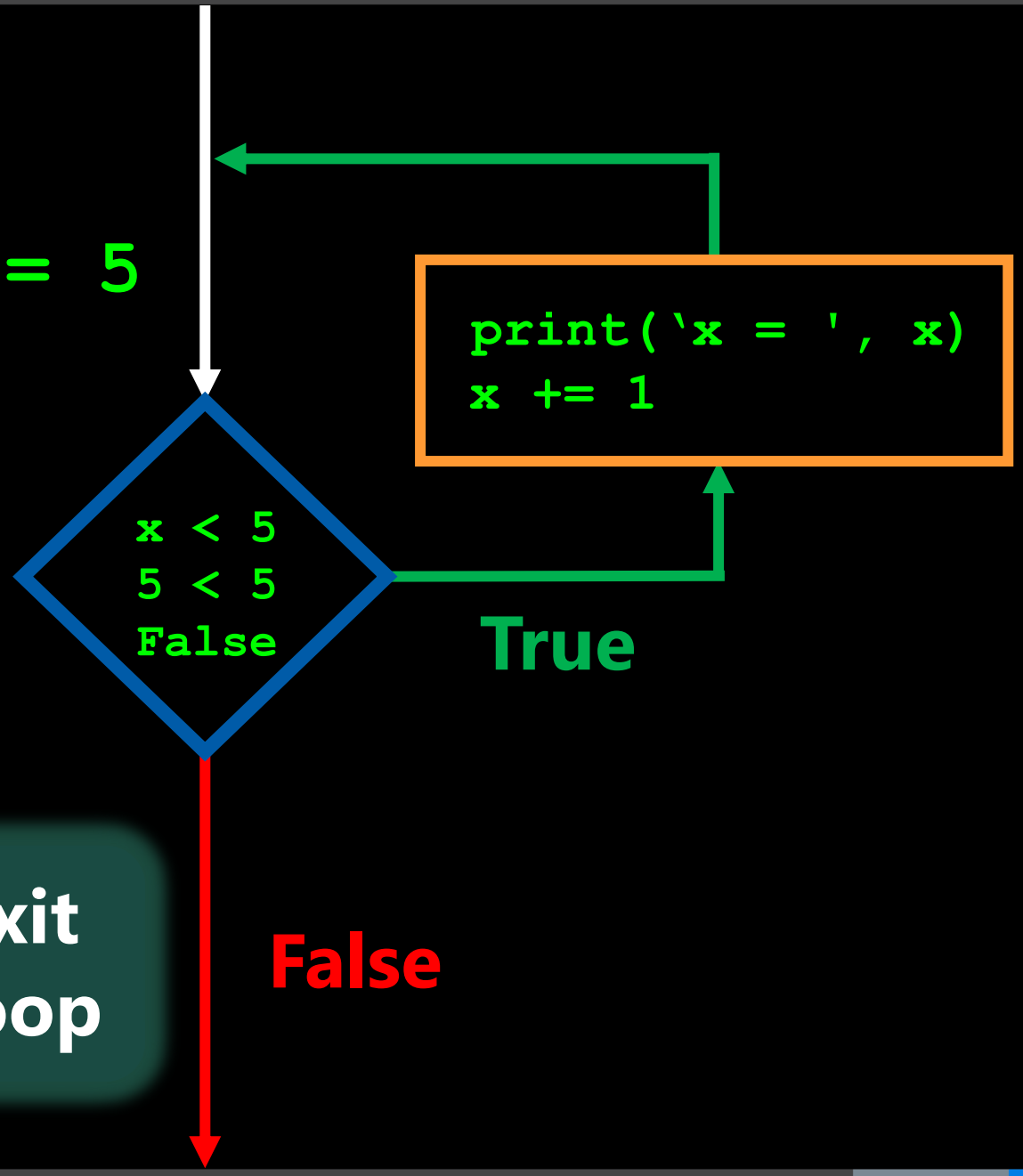
```
x = 0
x = 1
x = 2
x = 3
x = 4
```

# 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:
3. While Loops

# `while` & `for` Loops

- In Python there are two types of loops `for` and `while`.
- For loops will be introduced in Week 3.
- What is the difference between `for` loops and `while` loops and when would we use one over the other?

```
for item in iterable:
    do something.
```

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

# while & for Loops

- for loop
- The number of iterations to be done is already known.

```
for item in iterable:
    do something.
```

```
cats = ['Persian', 'Siamese', 'Tabby']

for cat in cats:
    print(cat)

>>> Persian
>>> Siamese
>>> Tabby
```

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

# while & for Loops

- **while loop**
- The number of iterations to be done is NOT known and iteration continues until a condition is met.

```
x = 0
while x*x < 200:
      print(x)
      x += 1
```

```
>>> 0
>>> …
>>> 14
```

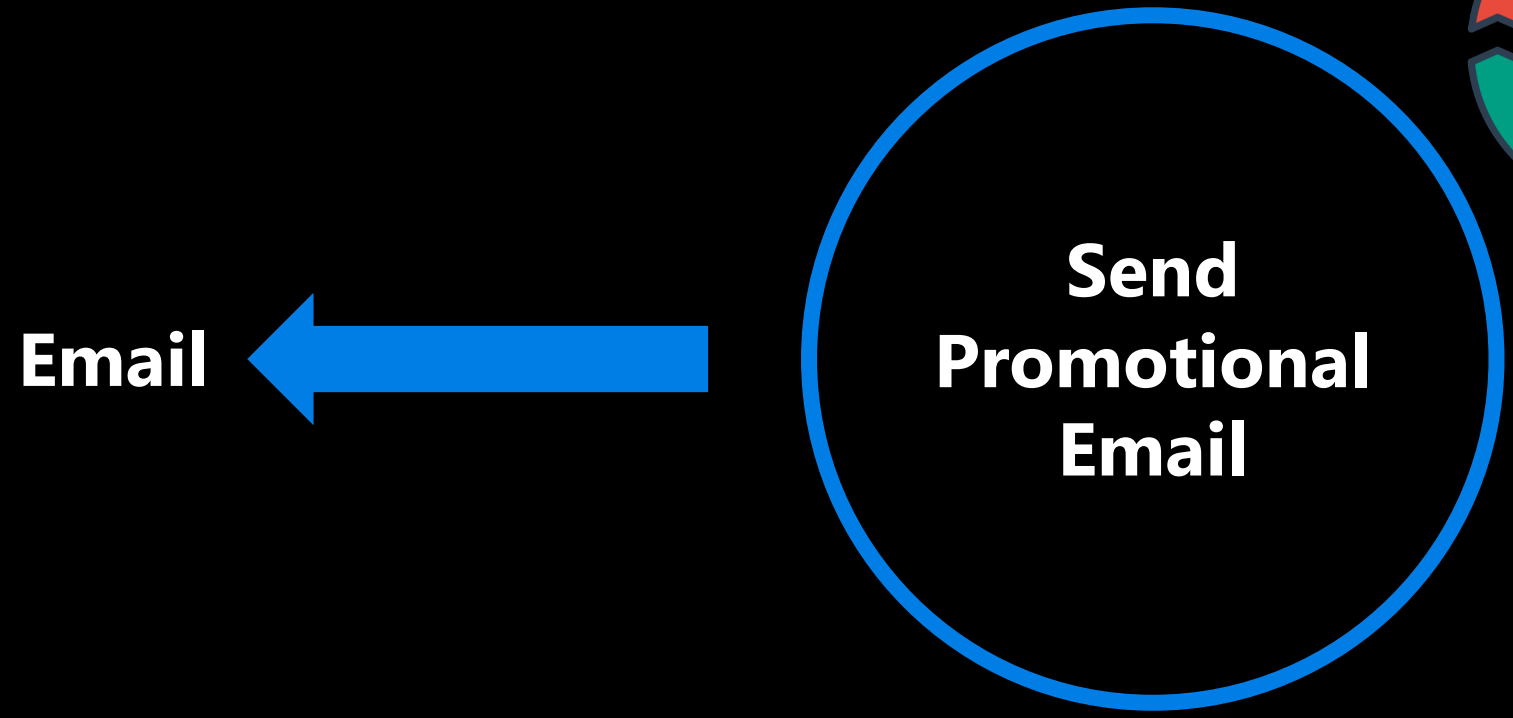
I don't know how many times I should iterate but I know when I should stop.

```
for item in iterable:
    do something.
```

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

# `while` & `for` Loops

- `for` loop or `while` loop?



# `while` & `for` Loops

- `for` loop or `while` loop?

List of Tweets

Looping

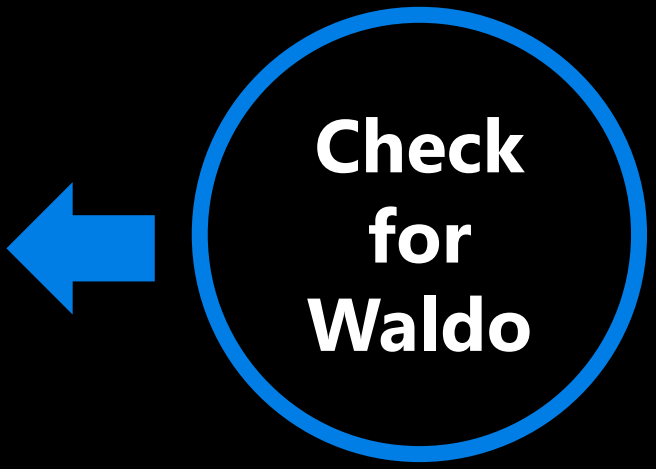
Does the Tweet contain #cleancode

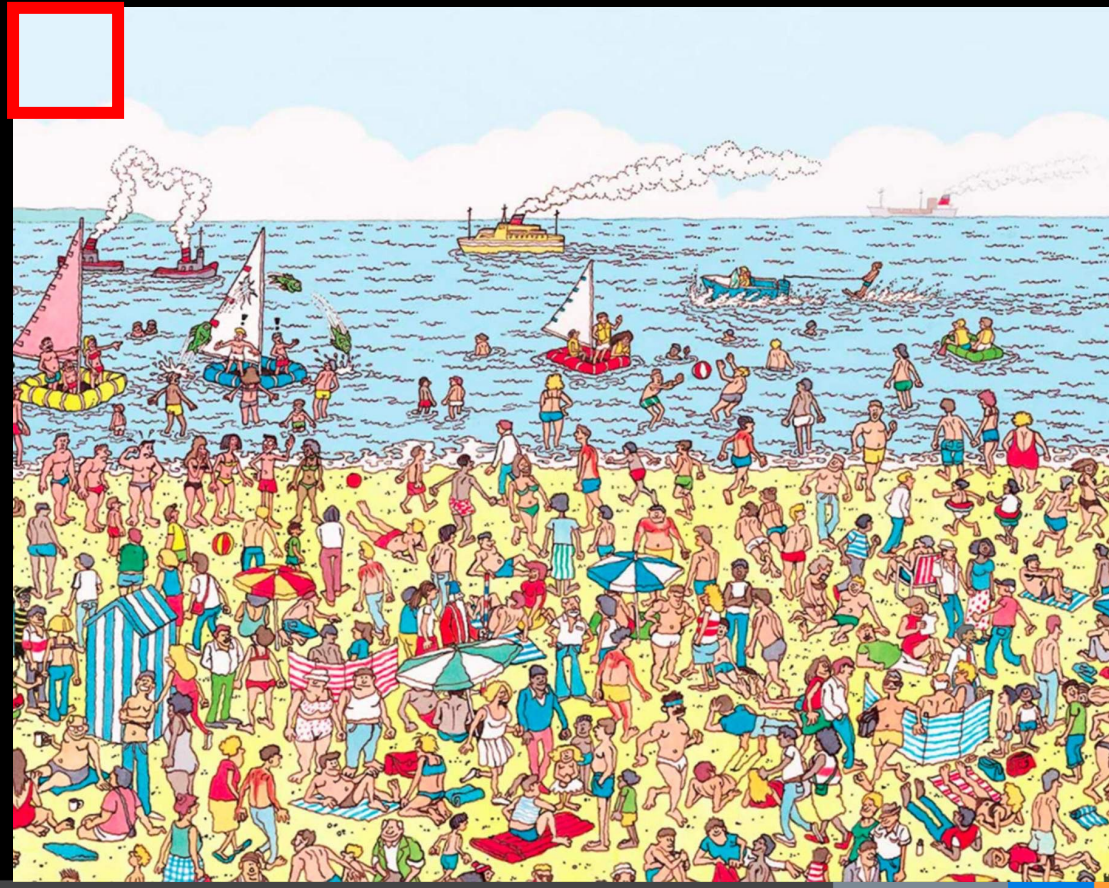
Yes/No

# `while` & `for` Loops

- `for` loop or `while` loop?

Looping



# 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 satisfied (`True`).
- 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:**
**4. Infinite Loops**

# Variable Scope does not apply to Loops

```
def func(x):
    x += 1

x = 0
func(x)
```

x (Local)

x (Global)

```
x = 0
while x < 10:
    x += 1
```

x (Global)

x (Global)

# While Loops

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

**Open your notebook**

**Click Link:**
**5. Back to User Input**

# Breakout Session 1

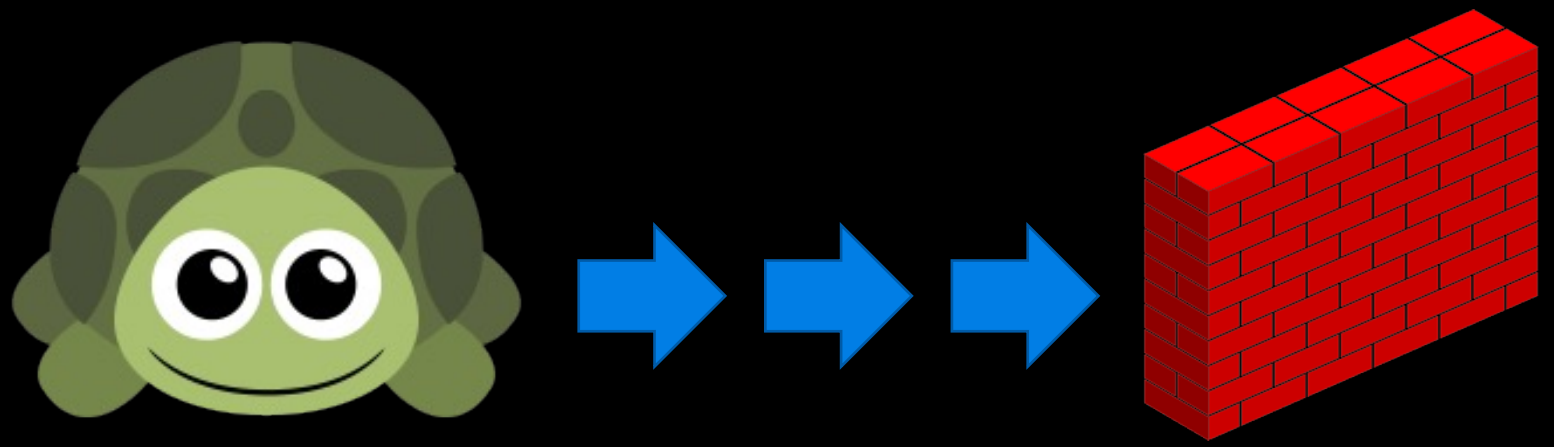
- Write code to print all the numbers from 0 to 20 that aren't evenly divisible by either 3 or 5.
- Zero is divisible by everything and should not appear in the output.

**Open your notebook**

**Click Link:**
**6. 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 am.



**You can follow along in the linked .py file**

# Random Module

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

```
import random

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

**Open your notebook**

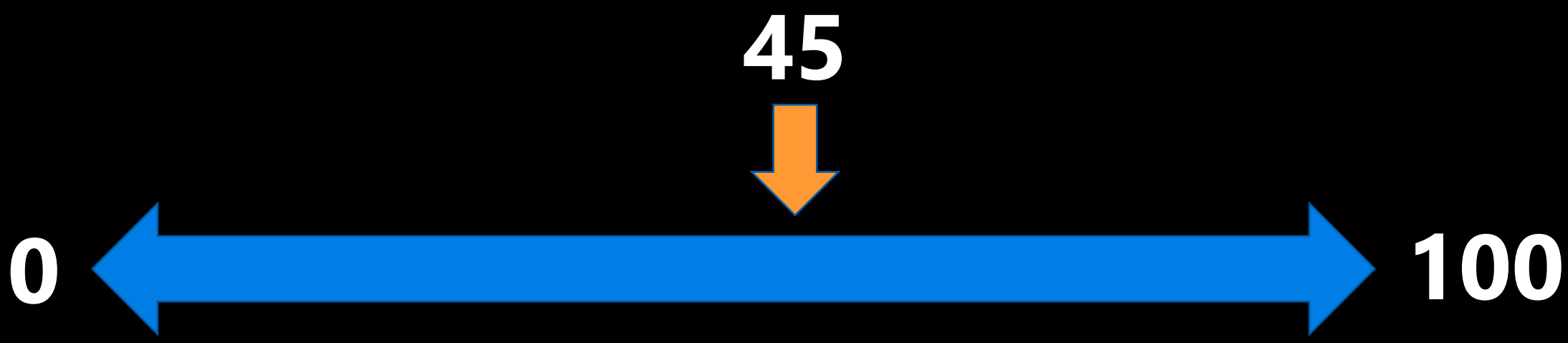
**Click Link:**
**7. Random Module**

# Guessing Game

- Let's build a simple guessing game.
  - Get the computer to choose a random integer from 0 to 100.
  - Ask the user for a guess and allow the user to input a guess or "q".
  - If the user inputs "q" print a nice message and end the program.
  - If the user enters a guess, tell them if they should guess higher, lower, or if they got it right.
  - If they got it right, print a nice message and quit.

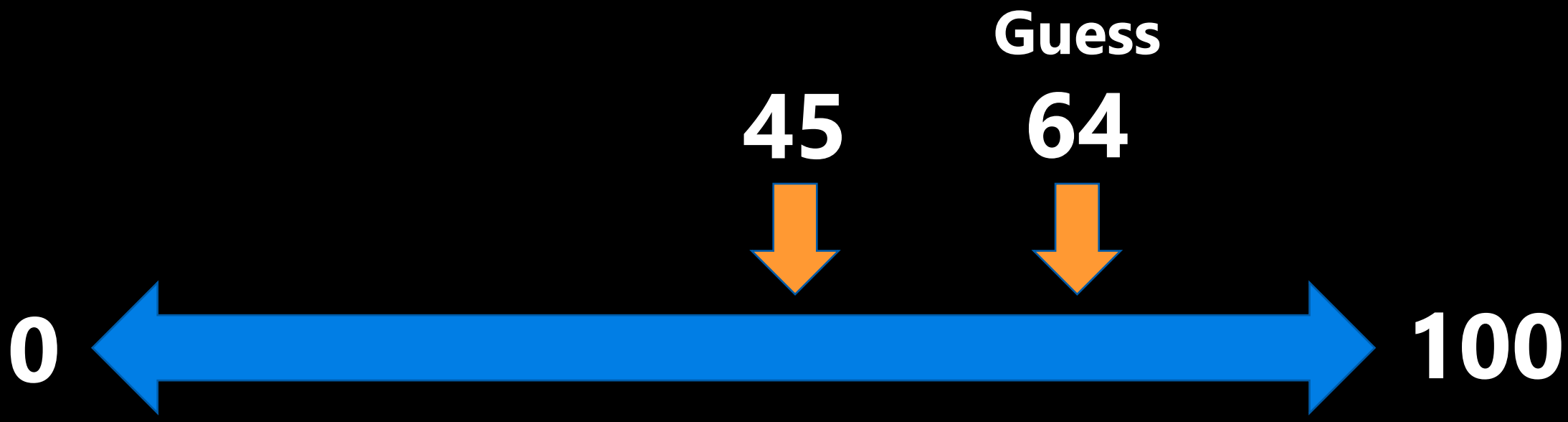
# Guessing Game

- Get the computer to choose a random integer from 0 to 100.
  - The computer selects 45.

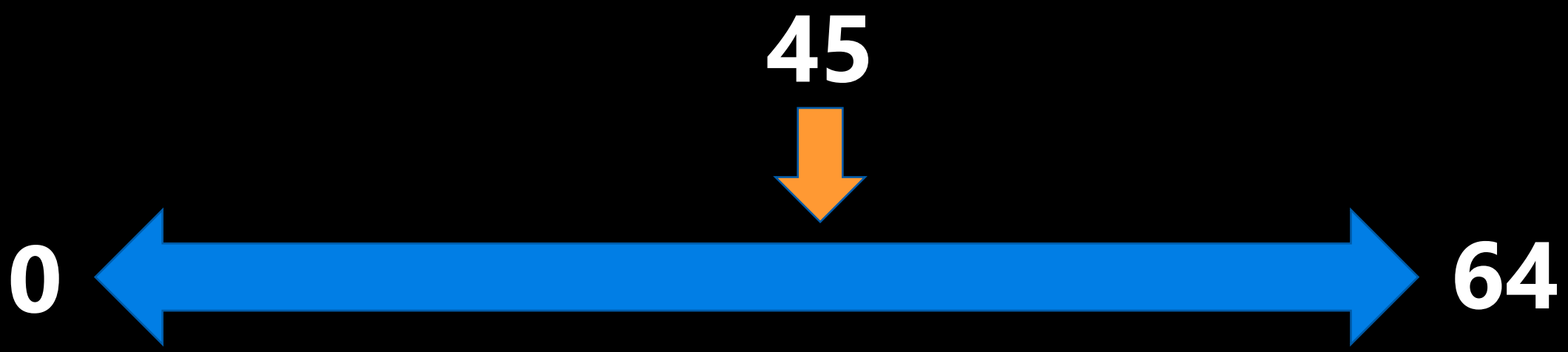


# Guessing Game

- The user guesses 64.
  - The computer says **LOWER**.

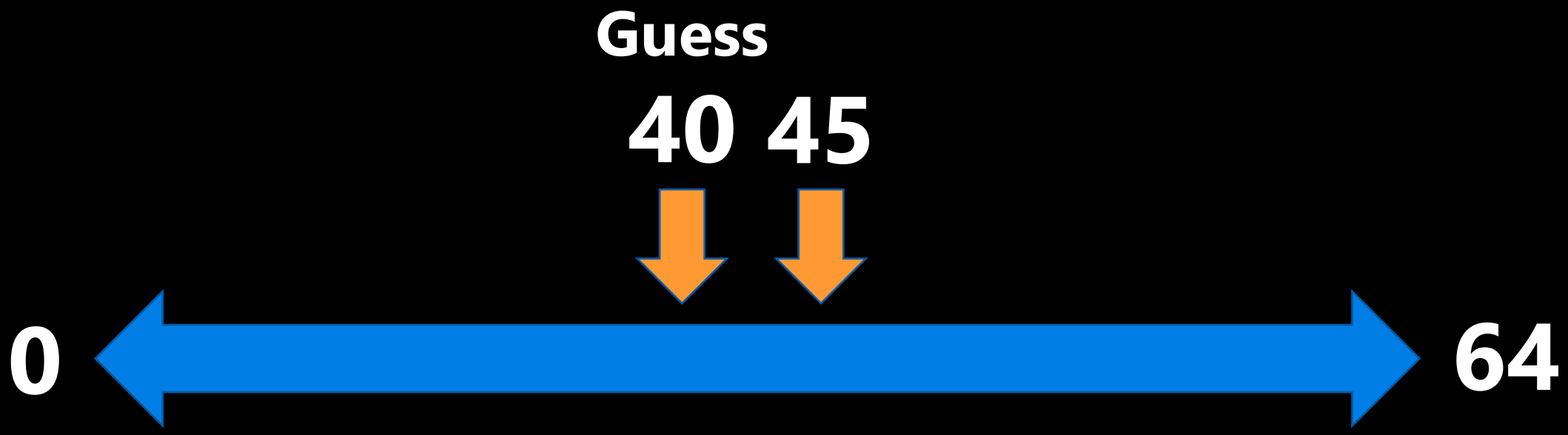


# Guessing Game

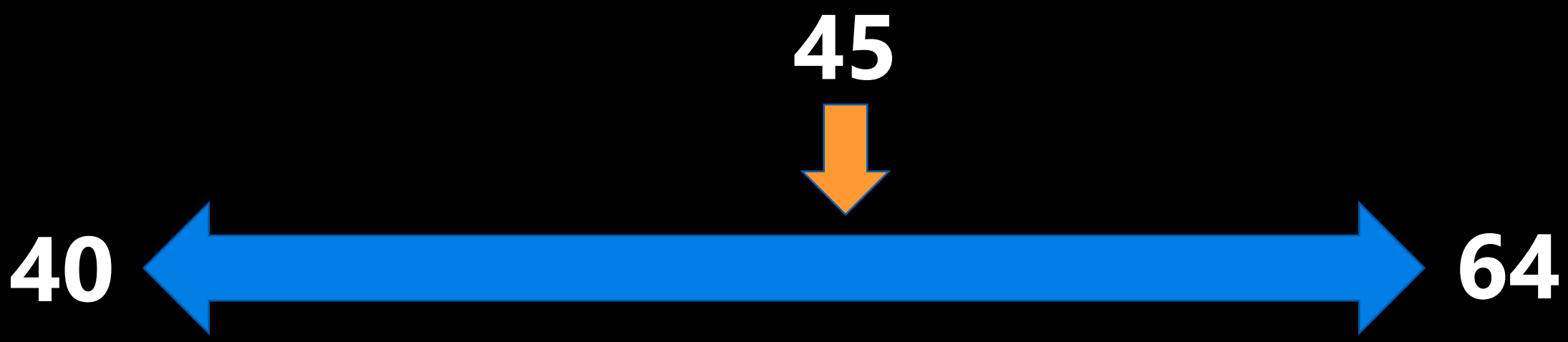


# Guessing Game

- The user guesses 40.
  - The computer says **HIGHER**.

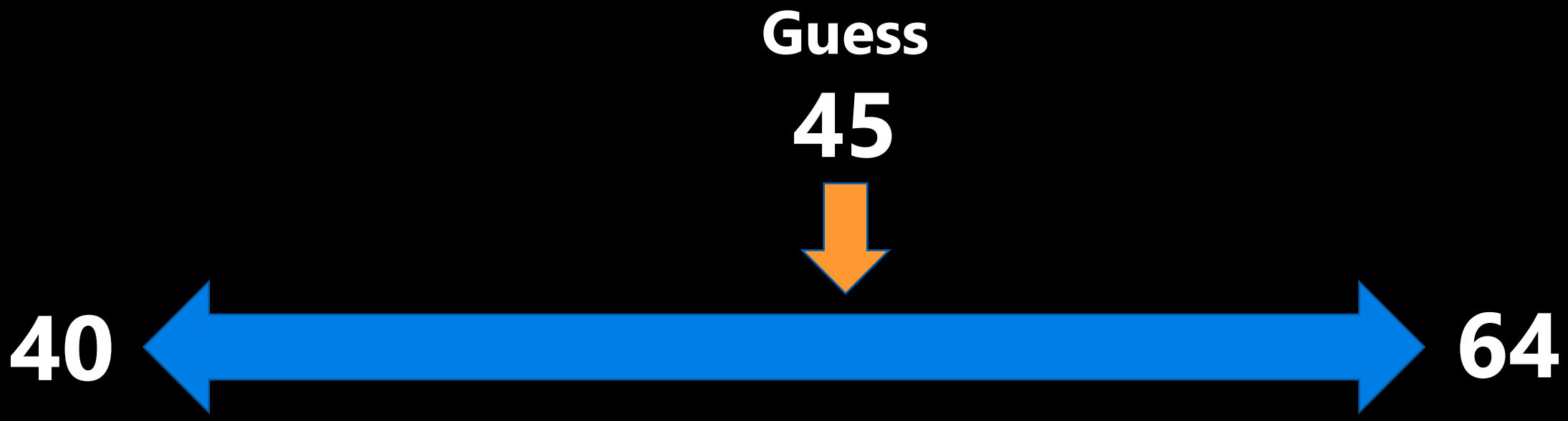


# Guessing Game



# Guessing Game

- The user guesses 45.
  - The computer says **YOU WIN**.



# Guessing Game

- Let's build a simple guessing game.
  1. Get the computer to choose a random integer from 0 to 100.
  2. Ask the user for a guess and allow the user to input a guess or "q".
  3. If the user inputs "q" print a nice message and end the program.
  4. If the user enters a guess, tell them if they should guess higher, lower, or if they got it right.
  5. If they got it right, print a nice message and quit.

**Open your notebook**

**Click Link:**
**8. A Simple Guessing Game**

# 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

# while loops.

**Week 2** | Lecture 2 (2.2.2)

if nothing else, write #cleancode