APS106

UNIVERSITY OF TORONTO

while loops.

Week 4 | Lecture 1 (4.1)

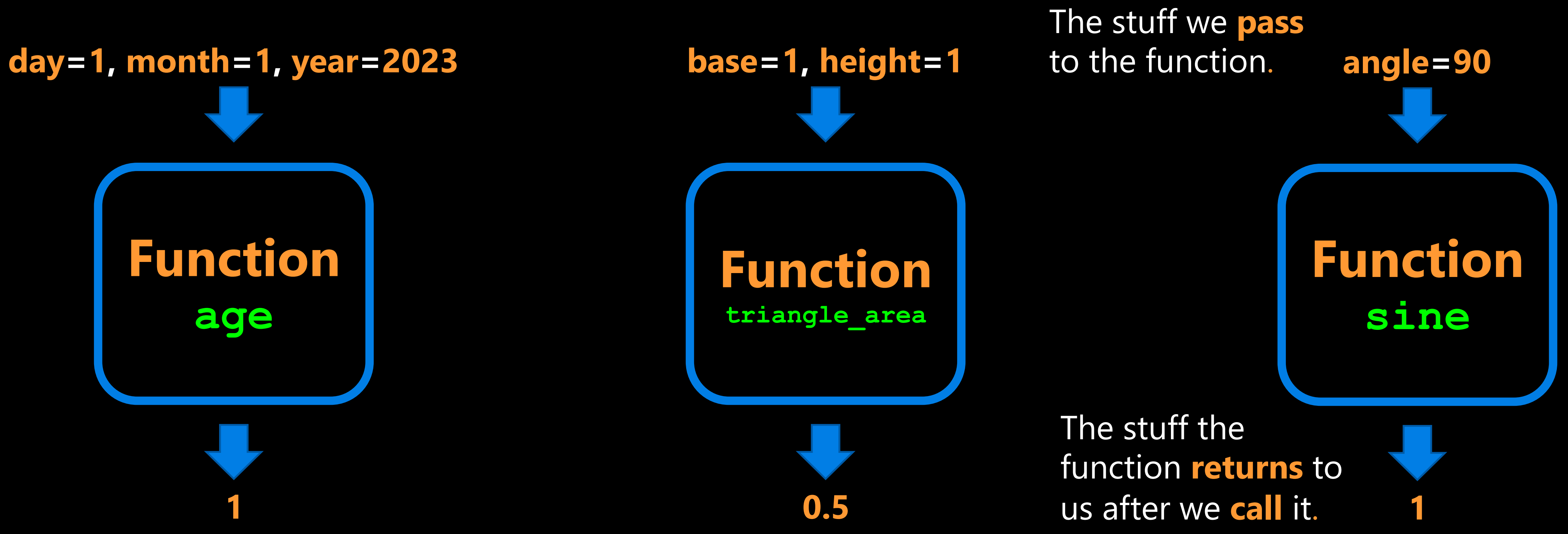
if nothing else, write #cleancode

function confusion

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

function, what are they?

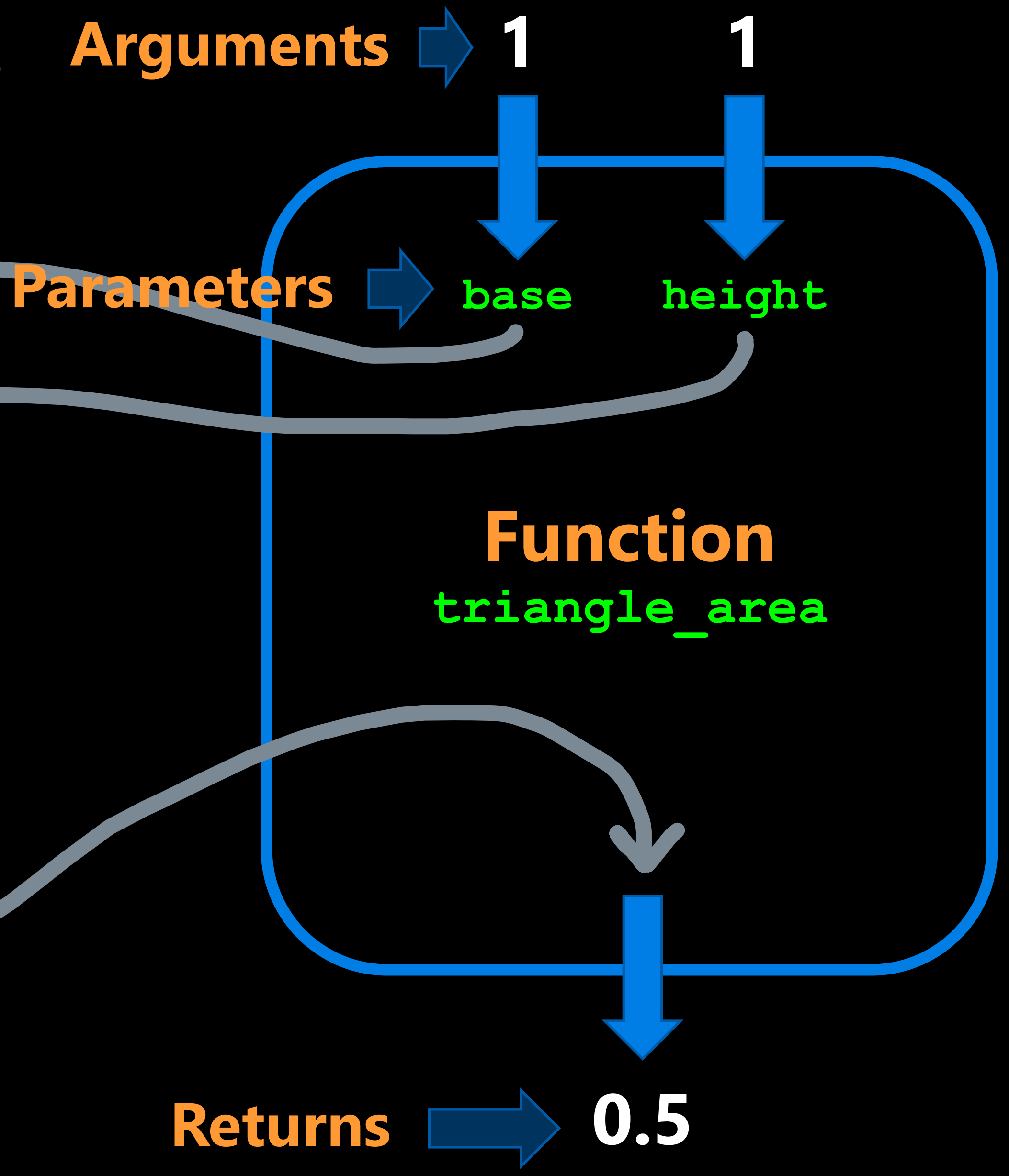
- A function is best explained as a self-contained piece of code that has inputs and an output.


day=1, month=1, year=2023
Function
age
1
base=1, height=1
Function
triangle_area
0.5
The stuff we pass to the function.
angle=90
Function
sine
The stuff the function returns to us after we call it.
1

parameters & arguments

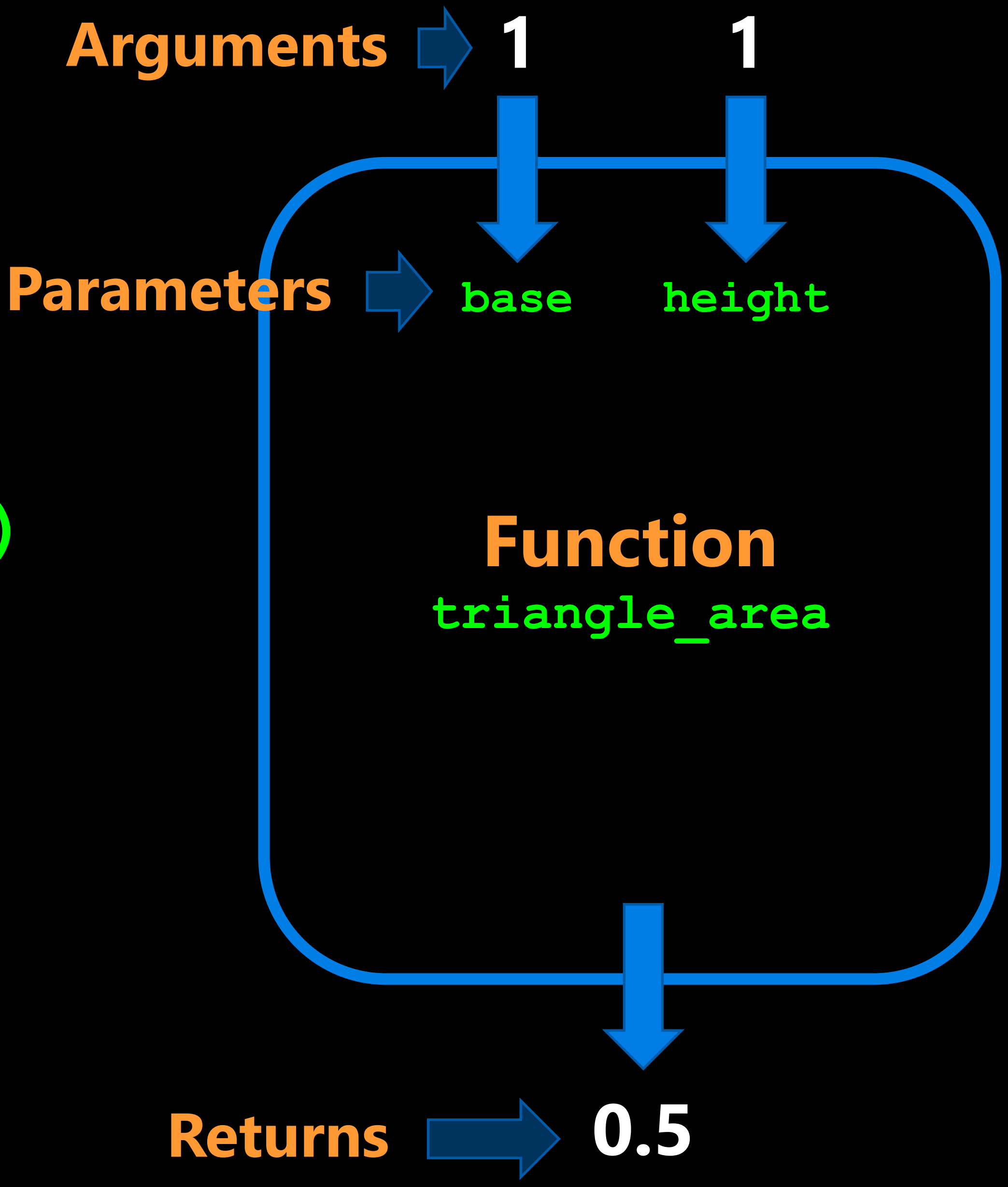
- function is a self-contained piece of code that has inputs and an output.

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



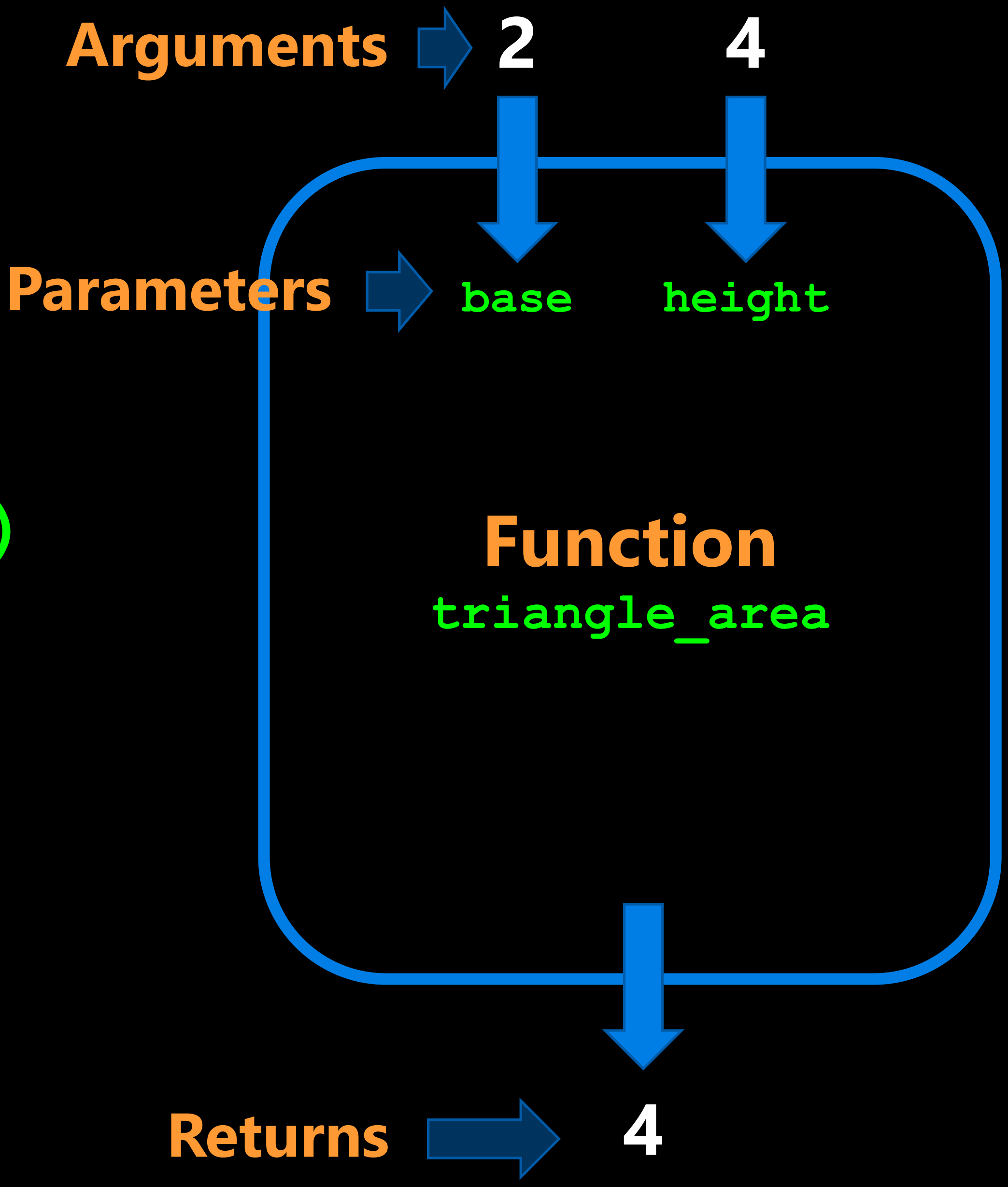
parameters & arguments

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



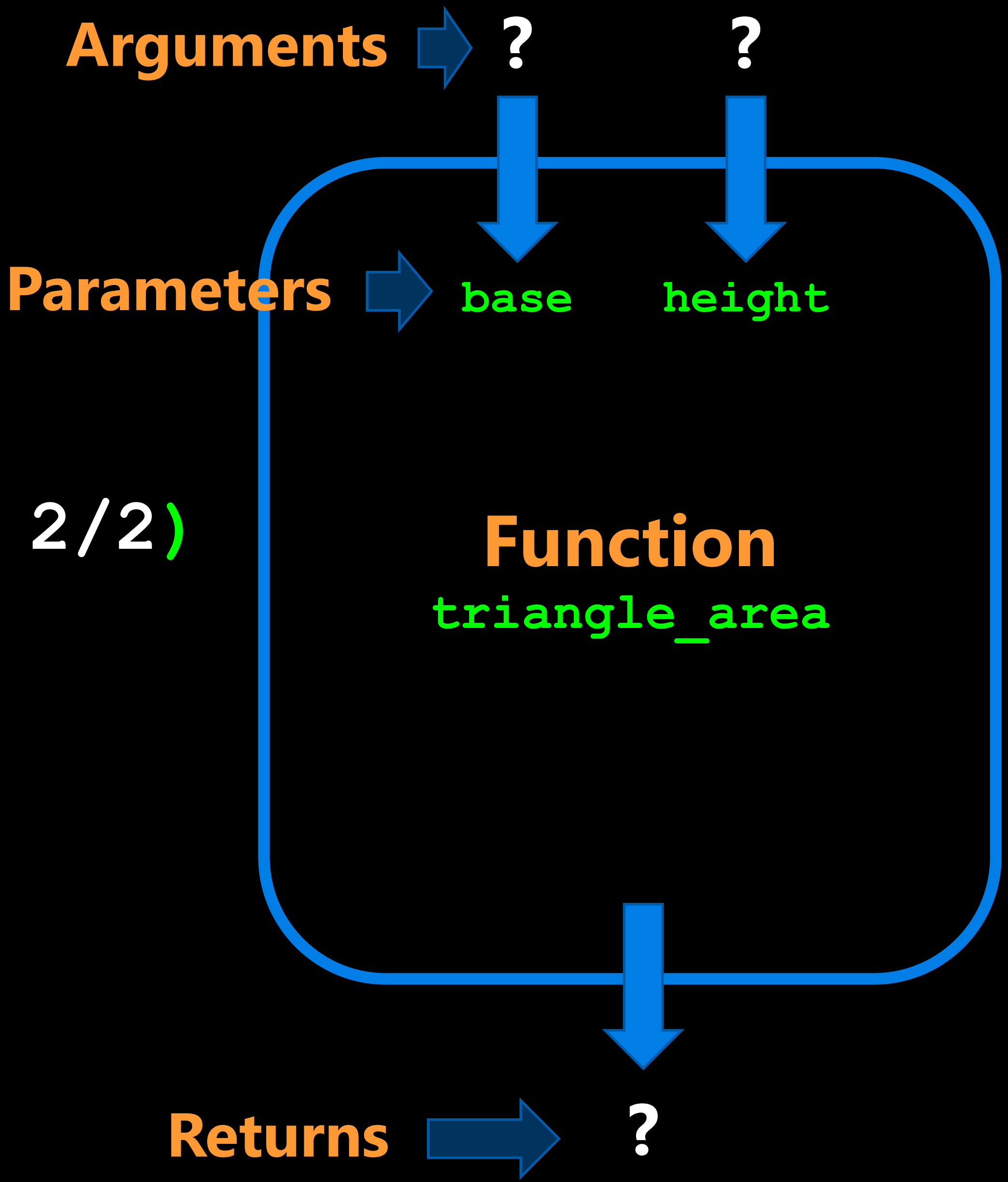
parameters & arguments

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



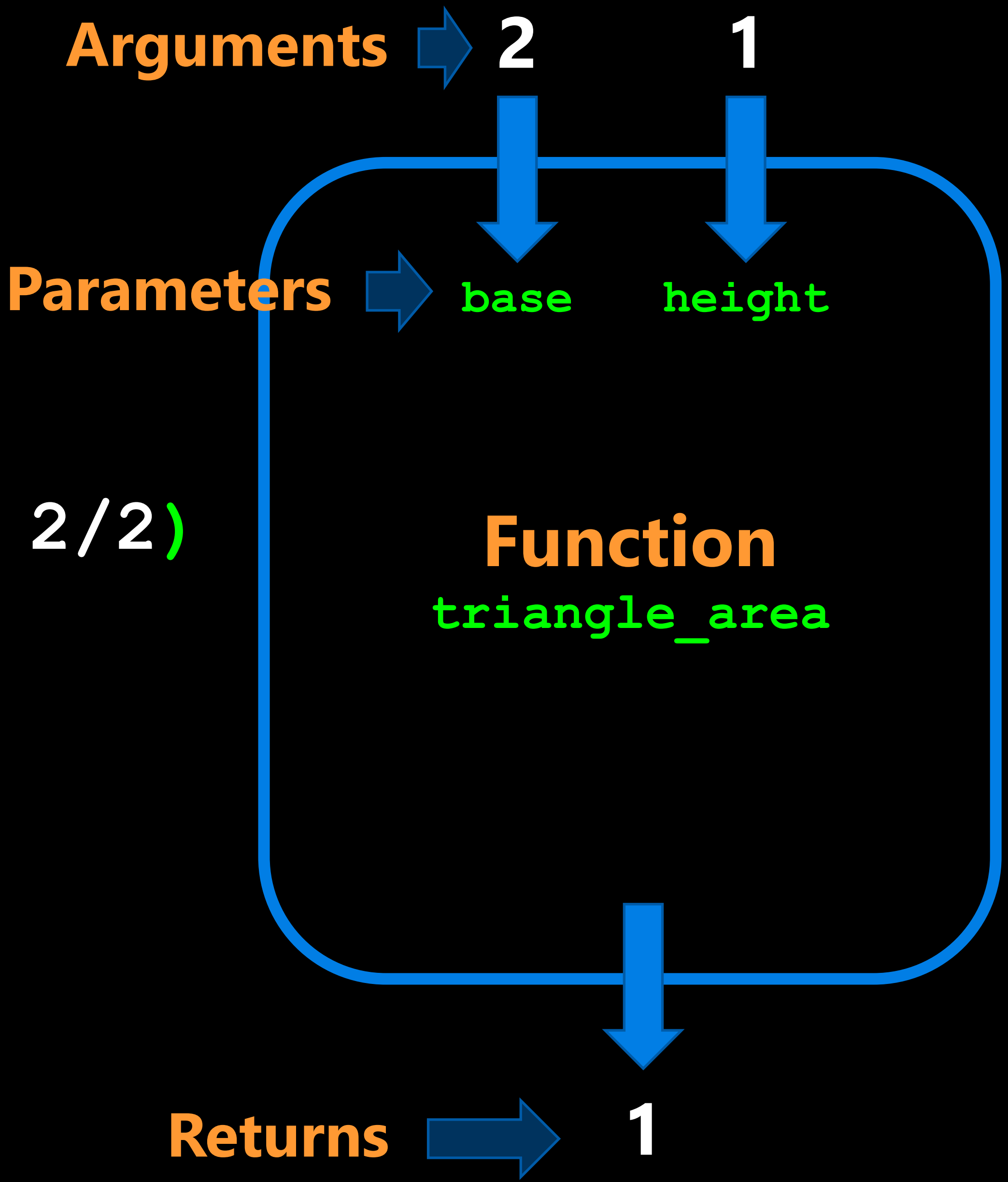
parameters & arguments

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



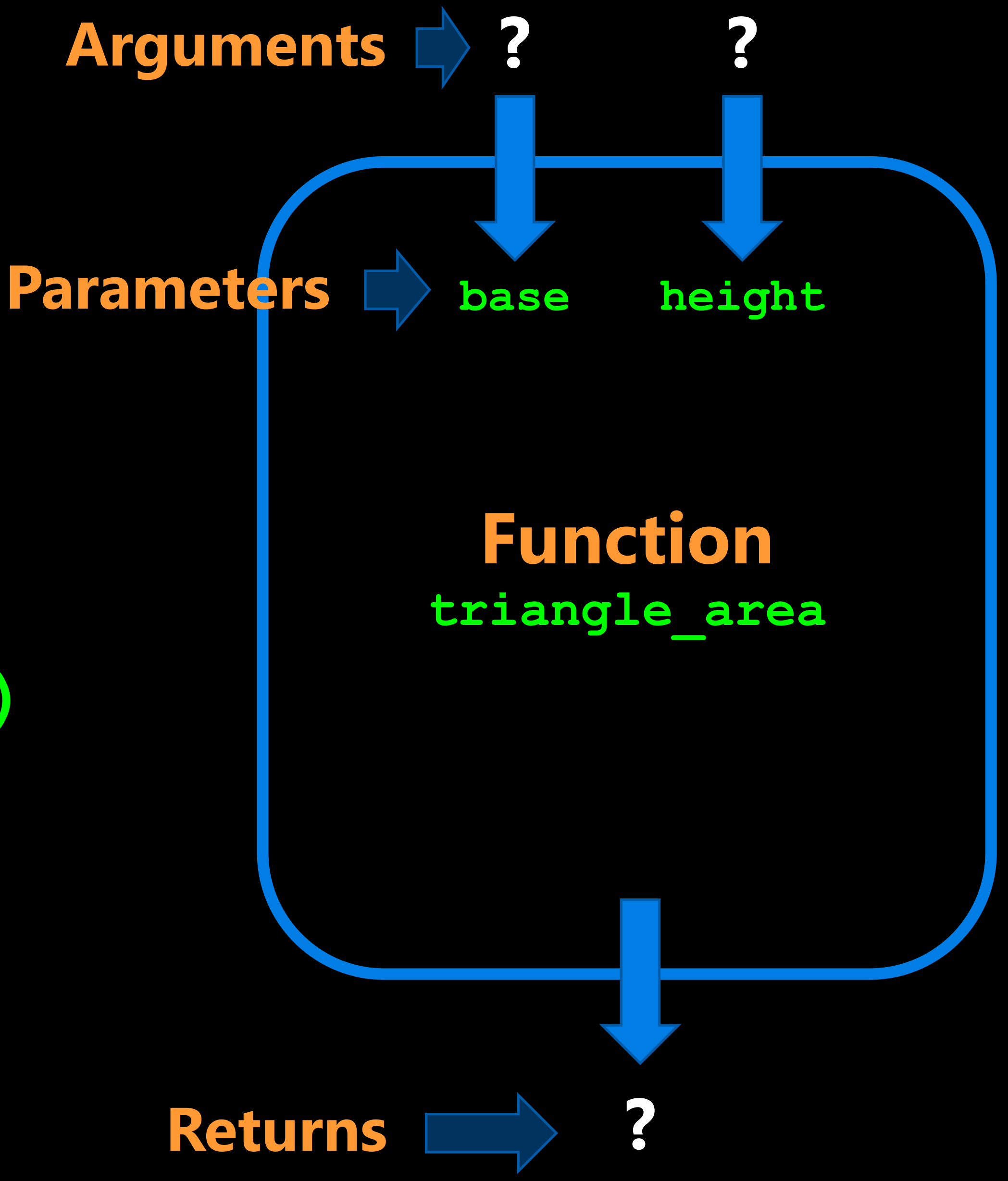
parameters & arguments

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



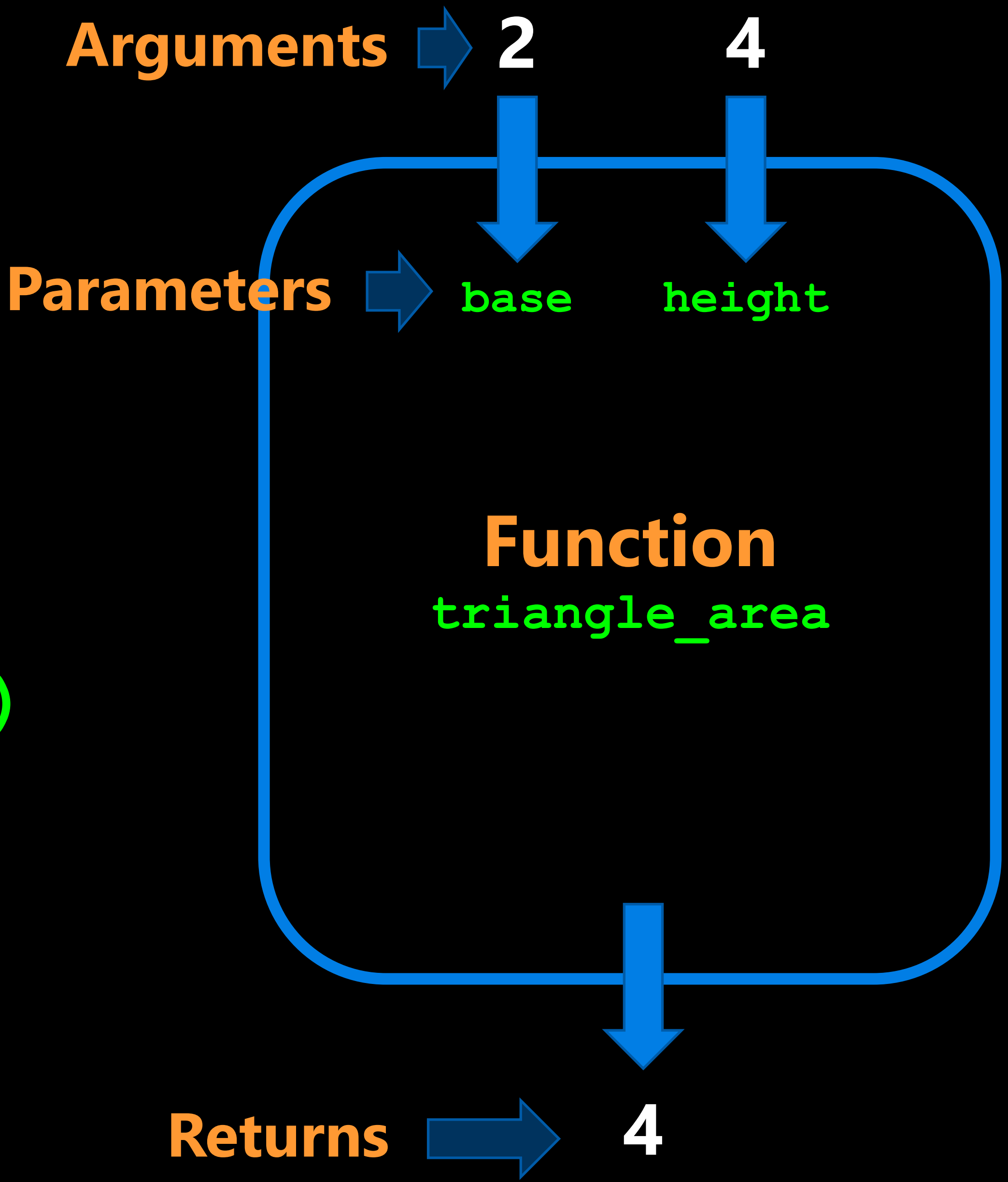
parameters & arguments

```
>>> 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
```

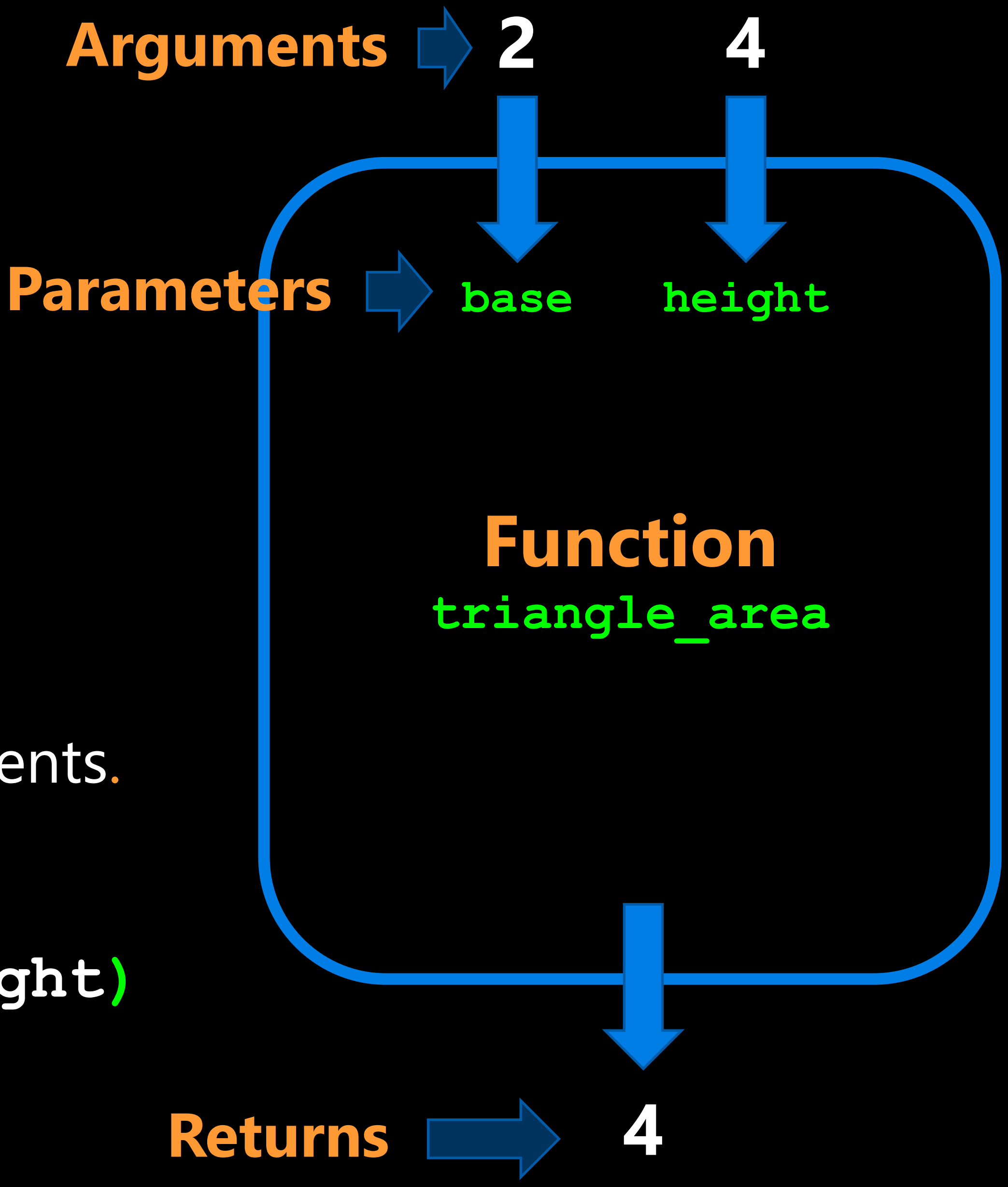


parameters & arguments

```
>>> 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
```

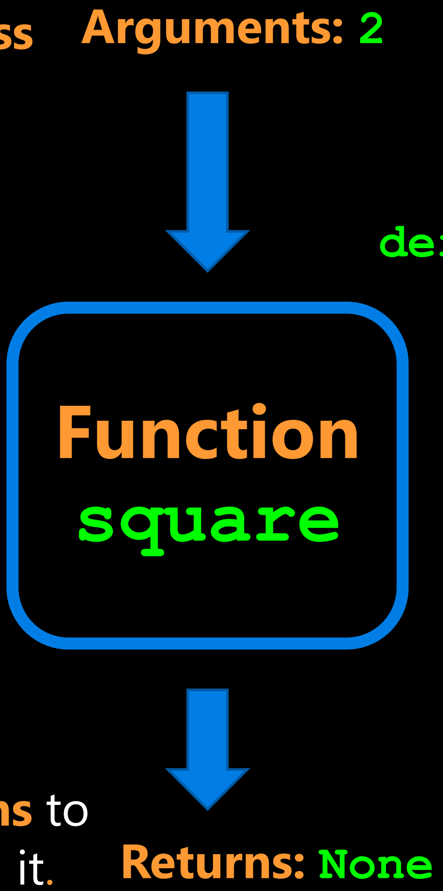
return

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

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

print

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

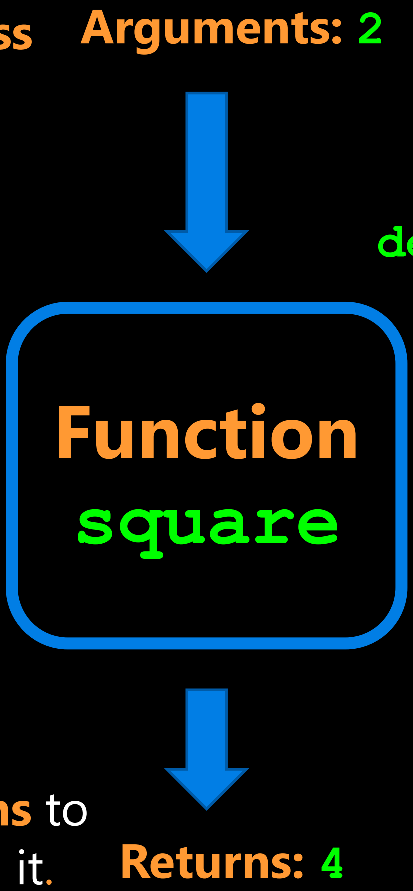


```
def square(x):
    output = x * x
    print(output)
```

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

return

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



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

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

print

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

`print` v.s. `return`

- Let's look at some examples.

Open your notebook

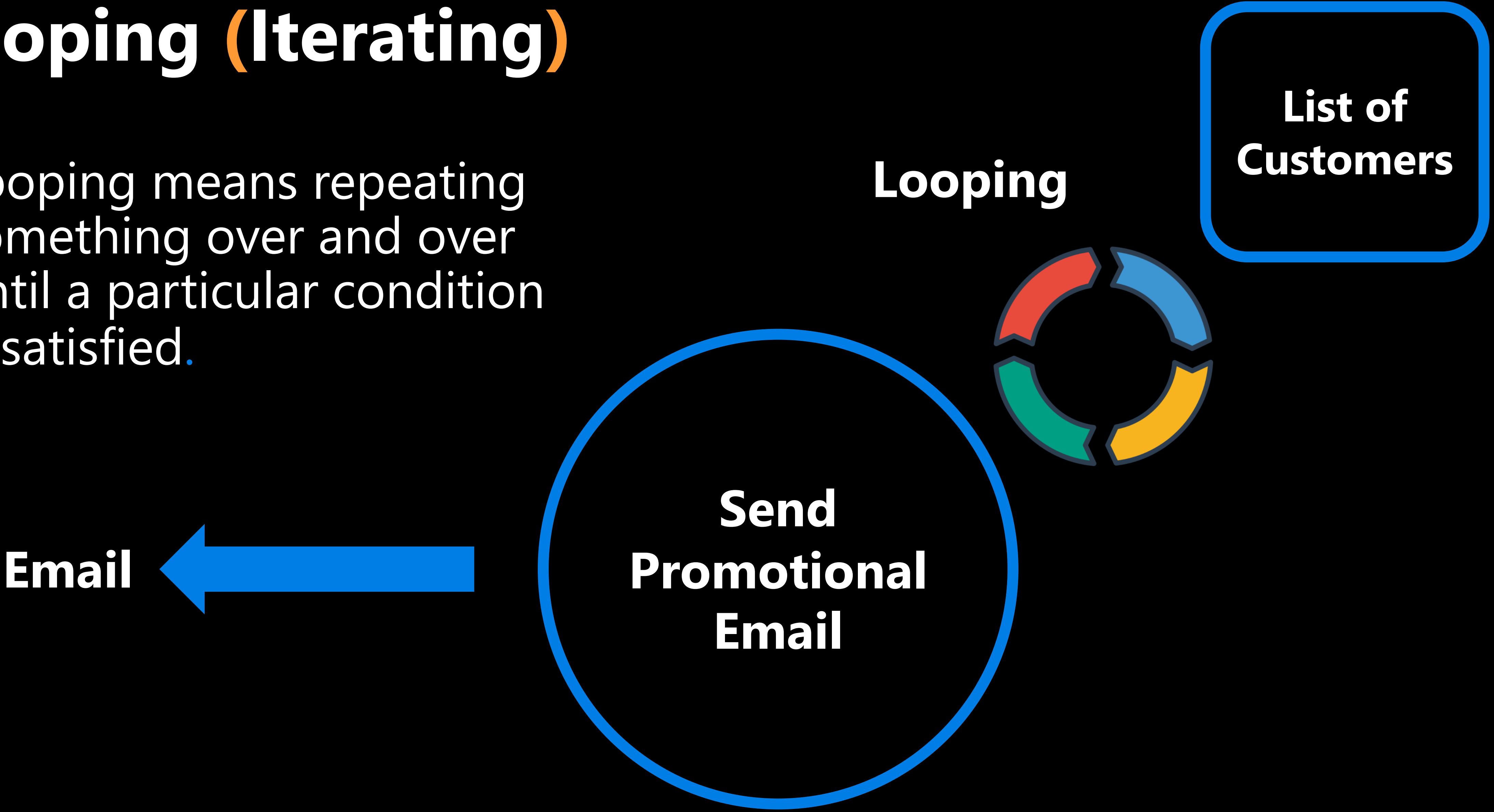
Click Link:
3. `print` v.s. `return`

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).

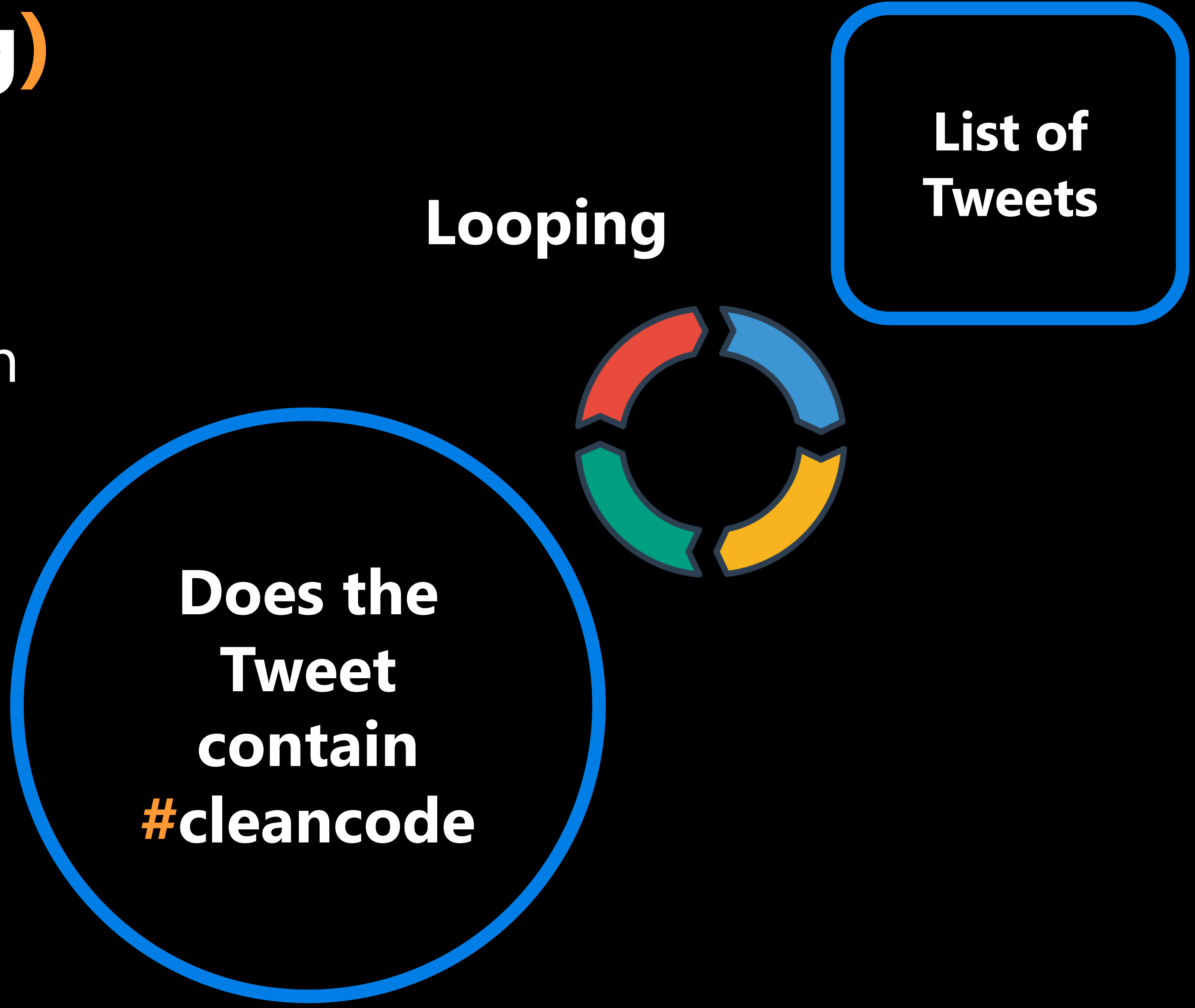
Looping (Iterating)

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



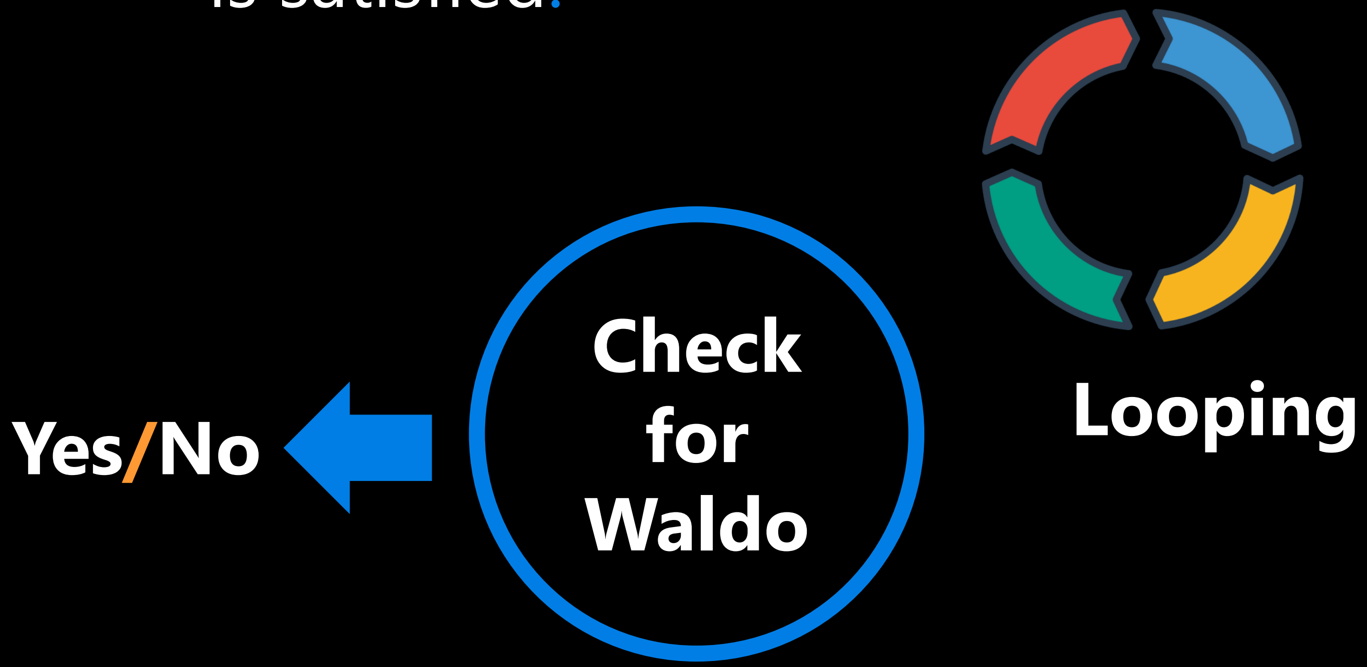
Looping (Iterating)

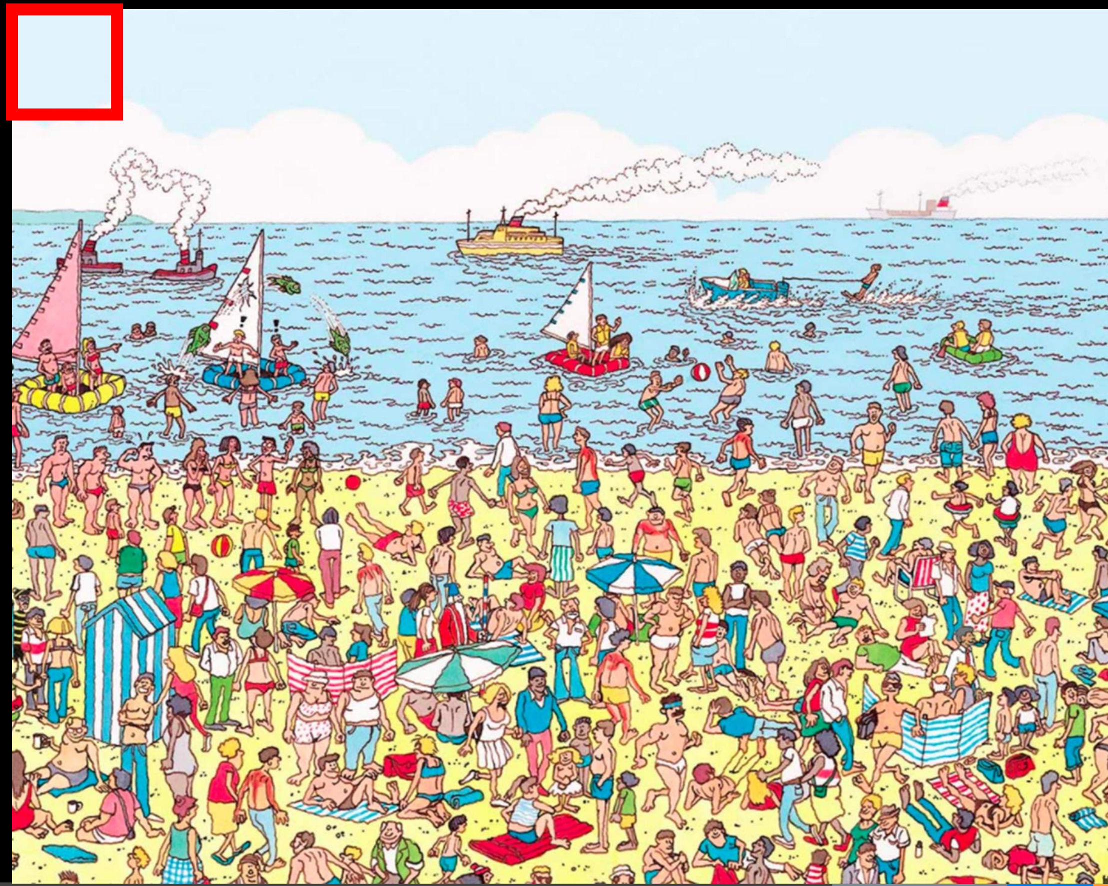
- Looping means repeating something over and over until a particular condition is satisfied.



Looping (Iterating)

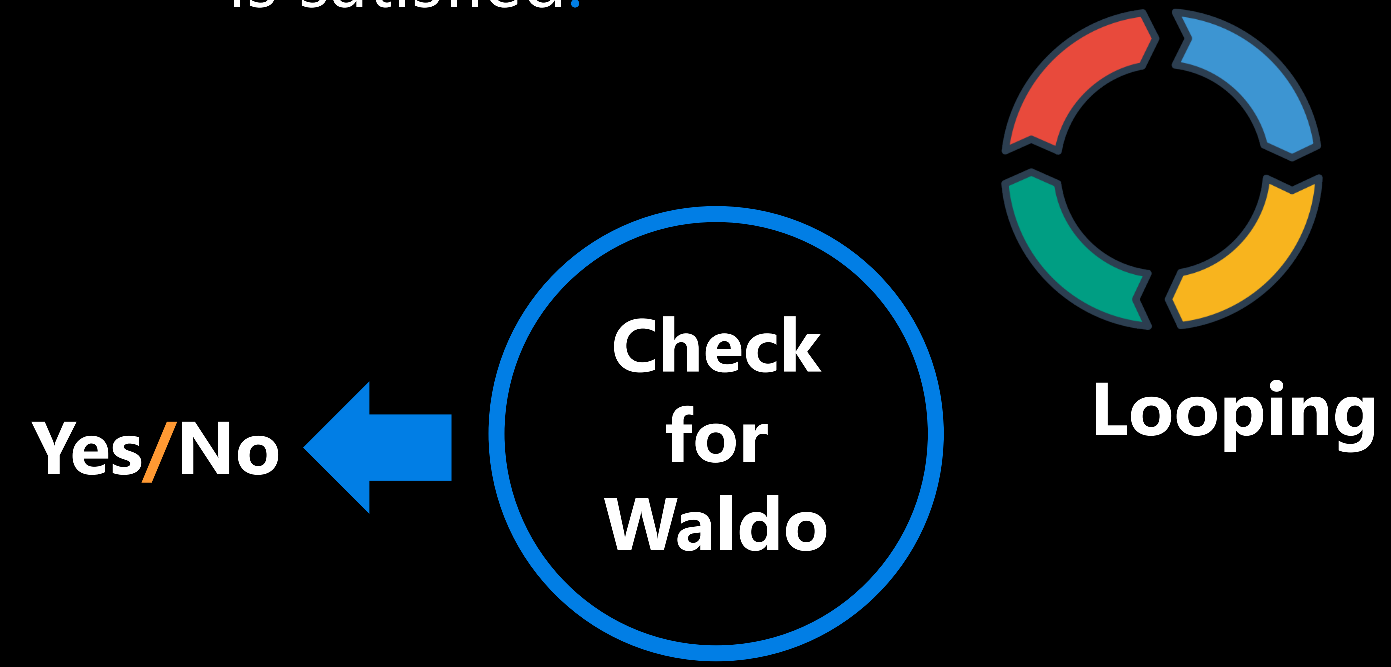
- Looping means repeating something over and over until a particular condition is satisfied.

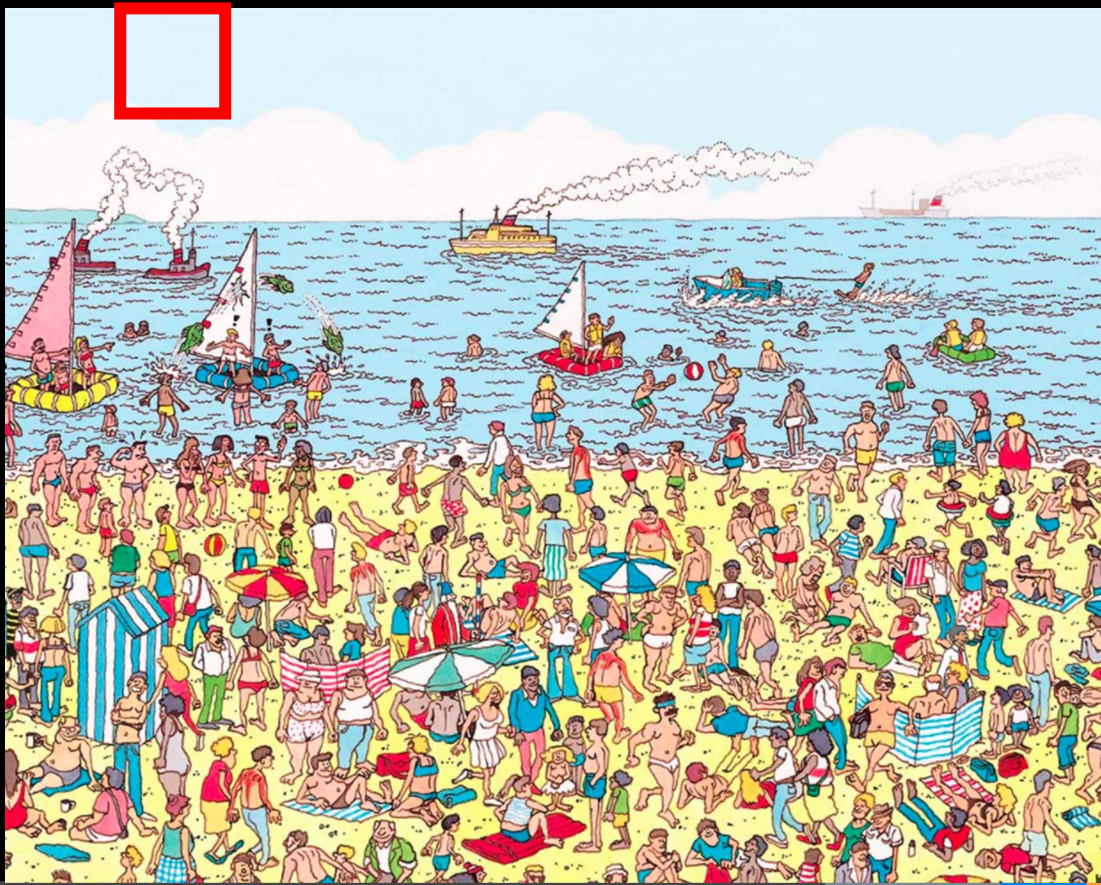




Looping (Iterating)

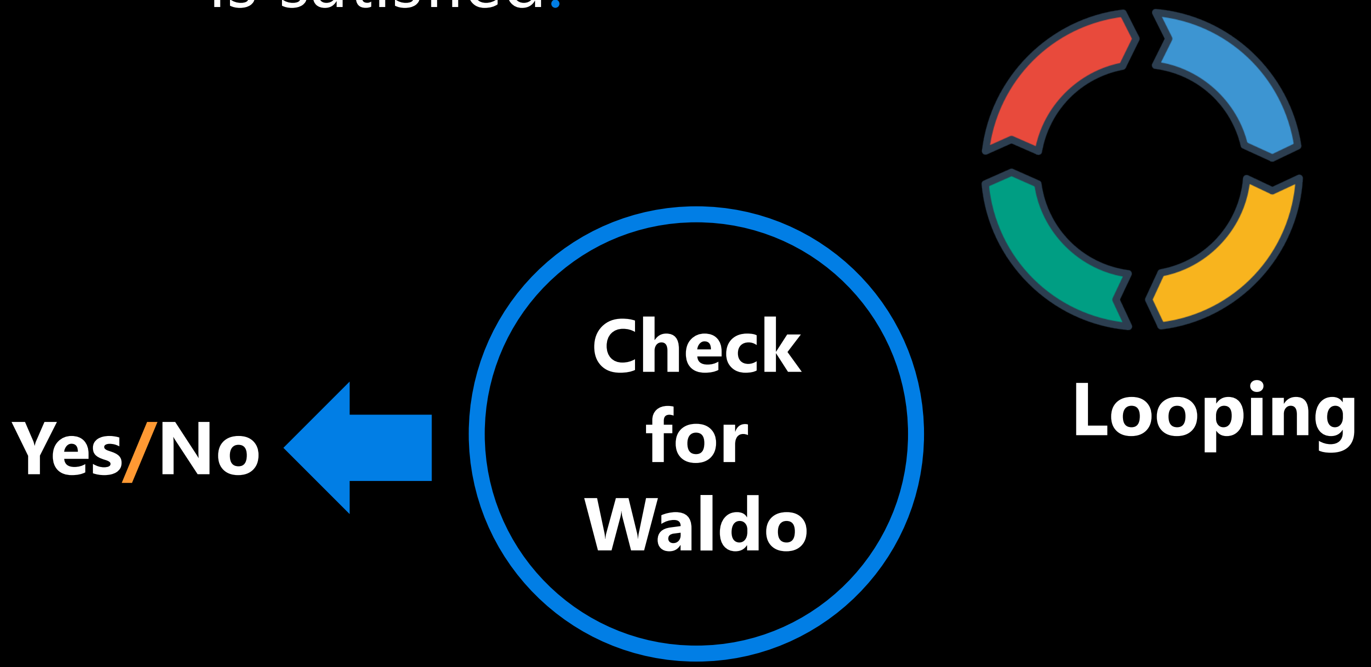
- Looping means repeating something over and over until a particular condition is satisfied.

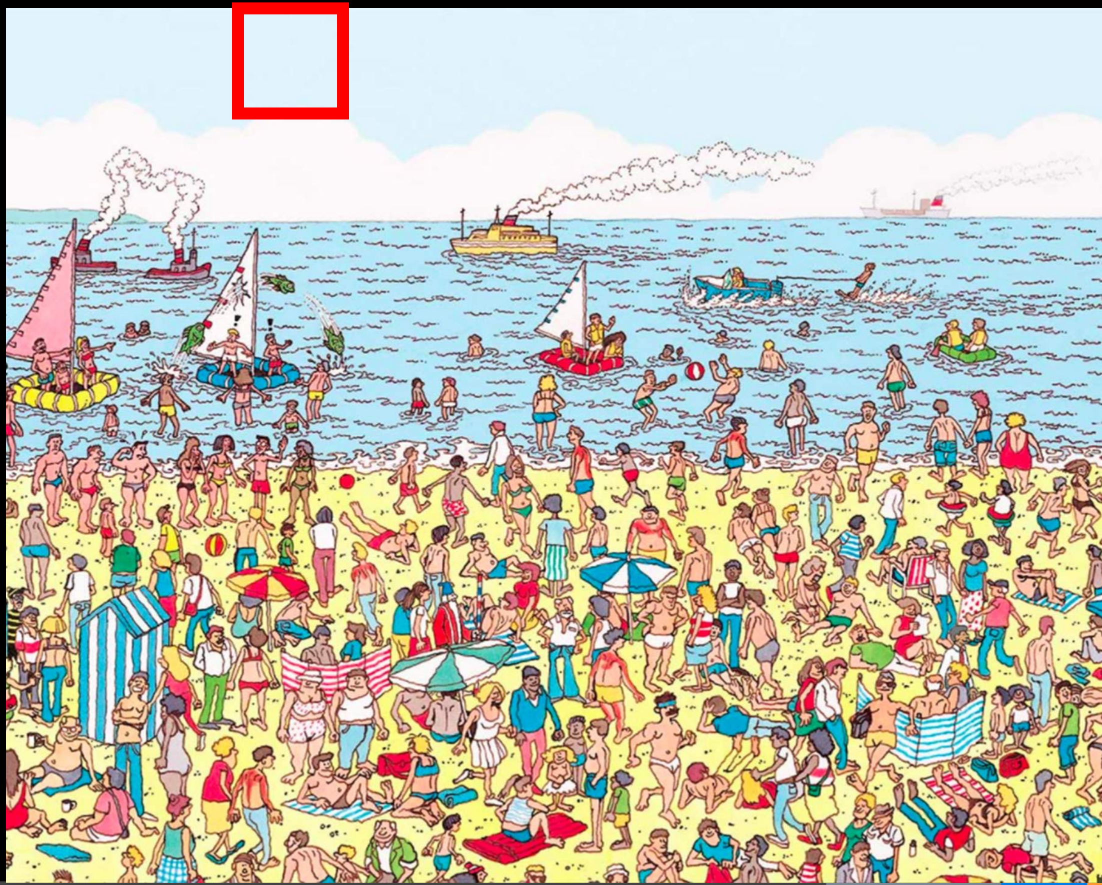




Looping (Iterating)

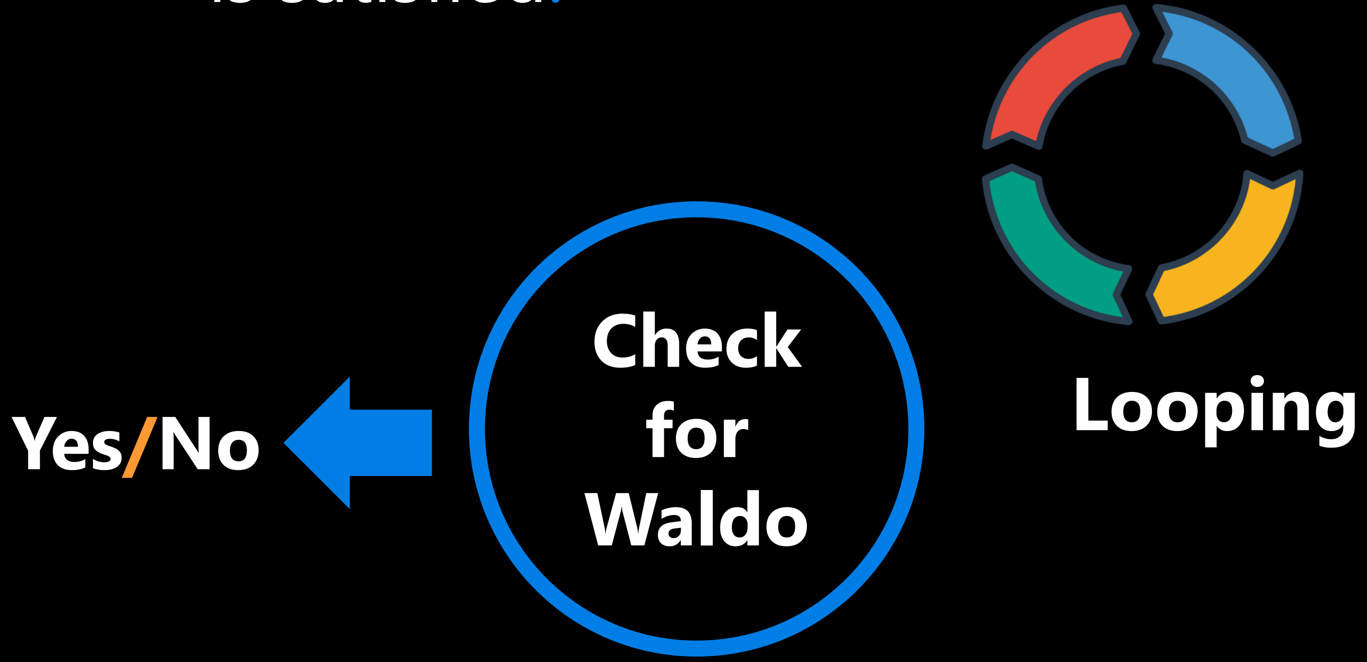
- Looping means repeating something over and over until a particular condition is satisfied.

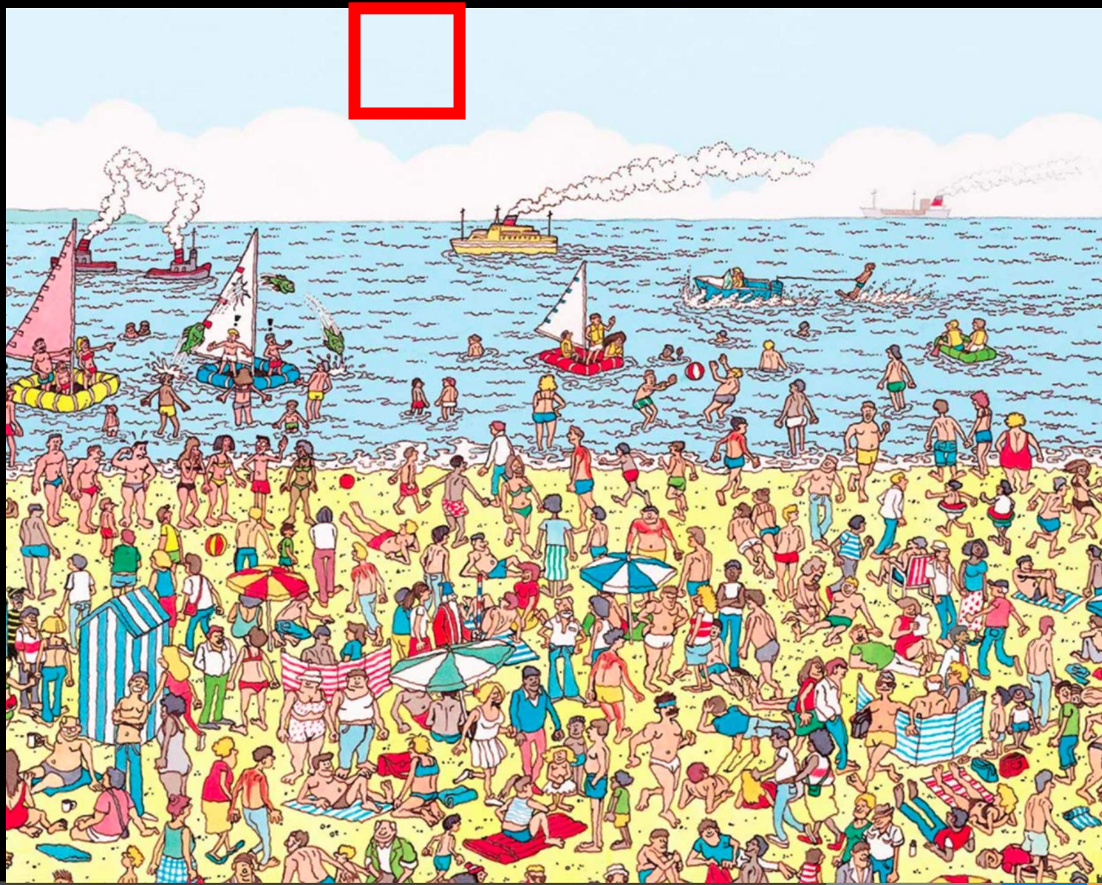




Looping (Iterating)

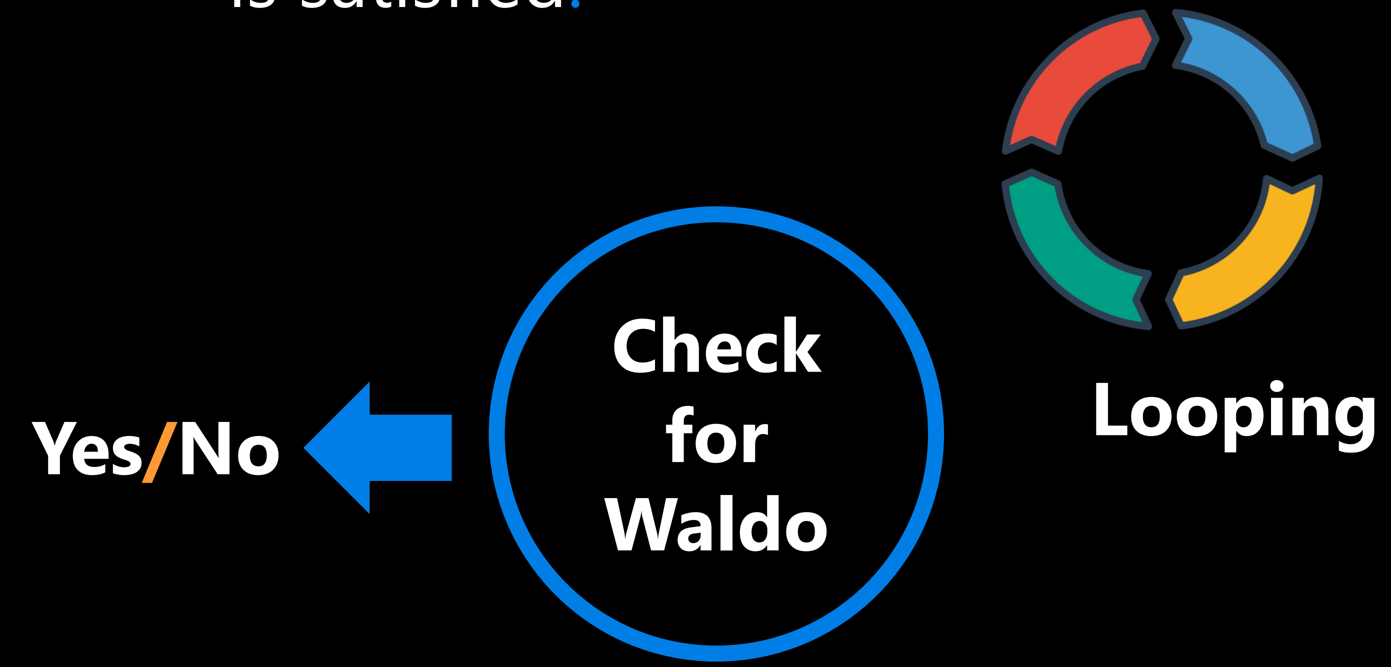
- Looping means repeating something over and over until a particular condition is satisfied.

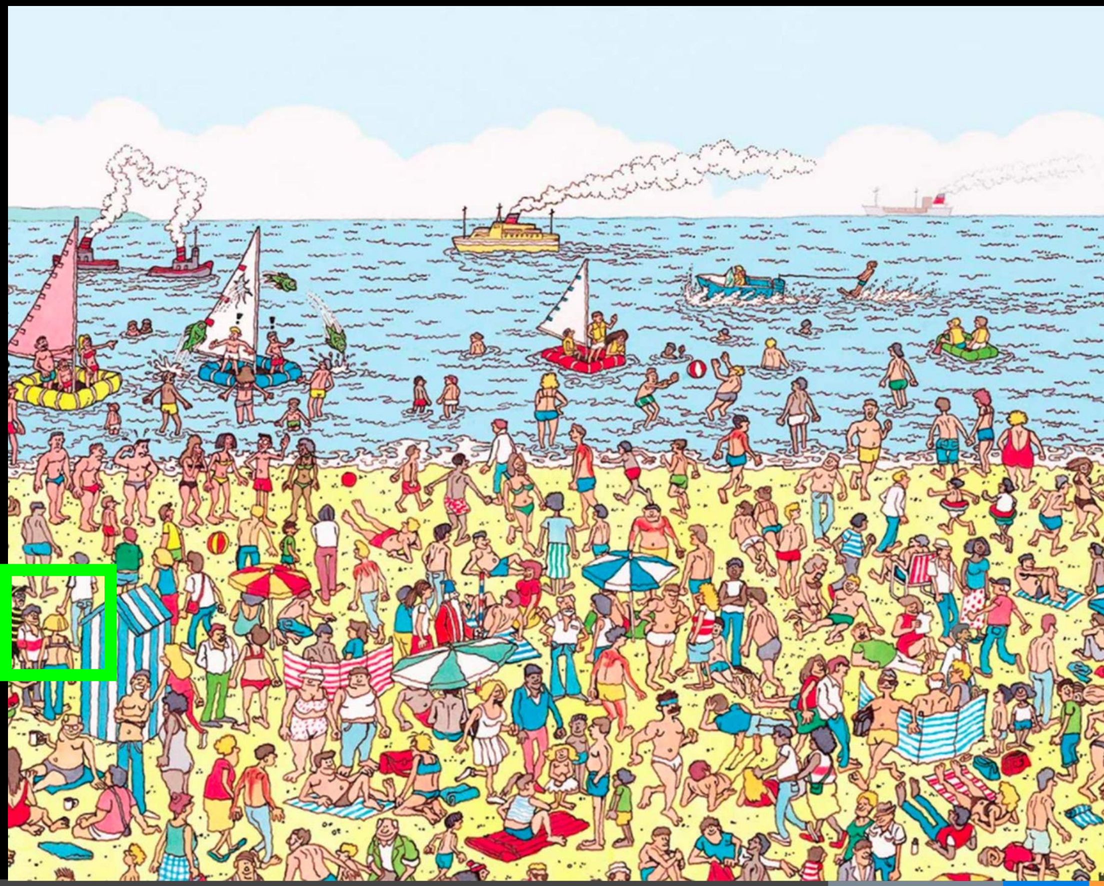




Looping (Iterating)

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



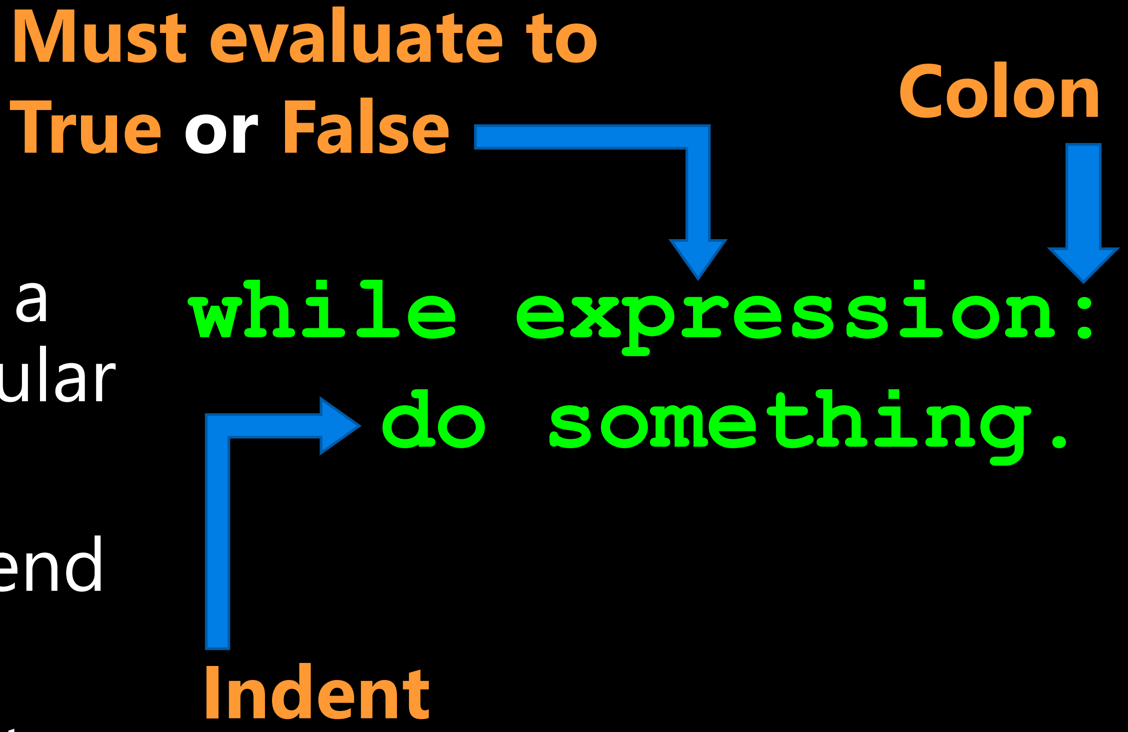


While Loops

- 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

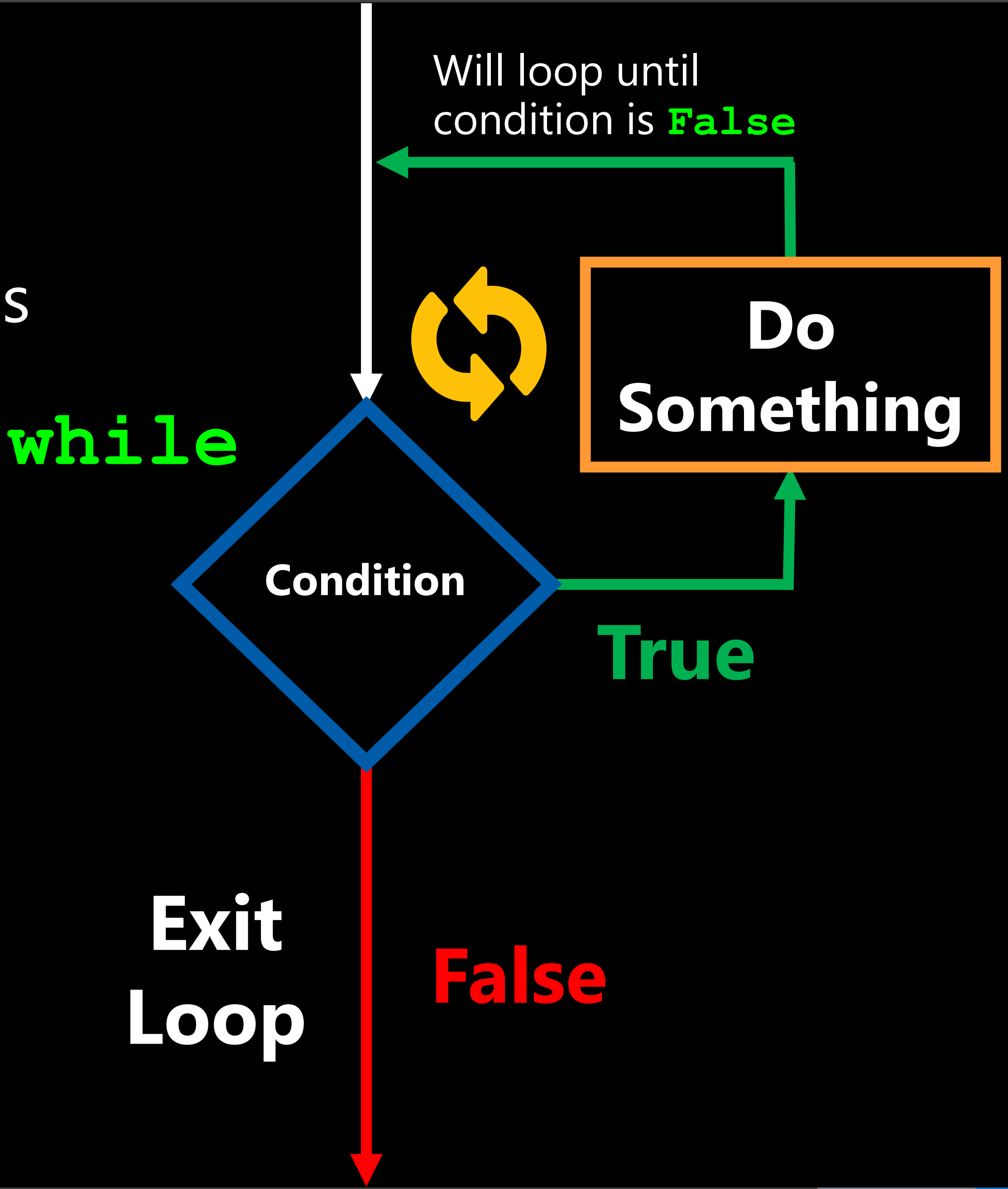
- The **while loop** keeps executing a piece of code as long as a particular condition is **True**.
- There must be a colon (:) at the end of the while statement.
- The action to be performed must be indented.

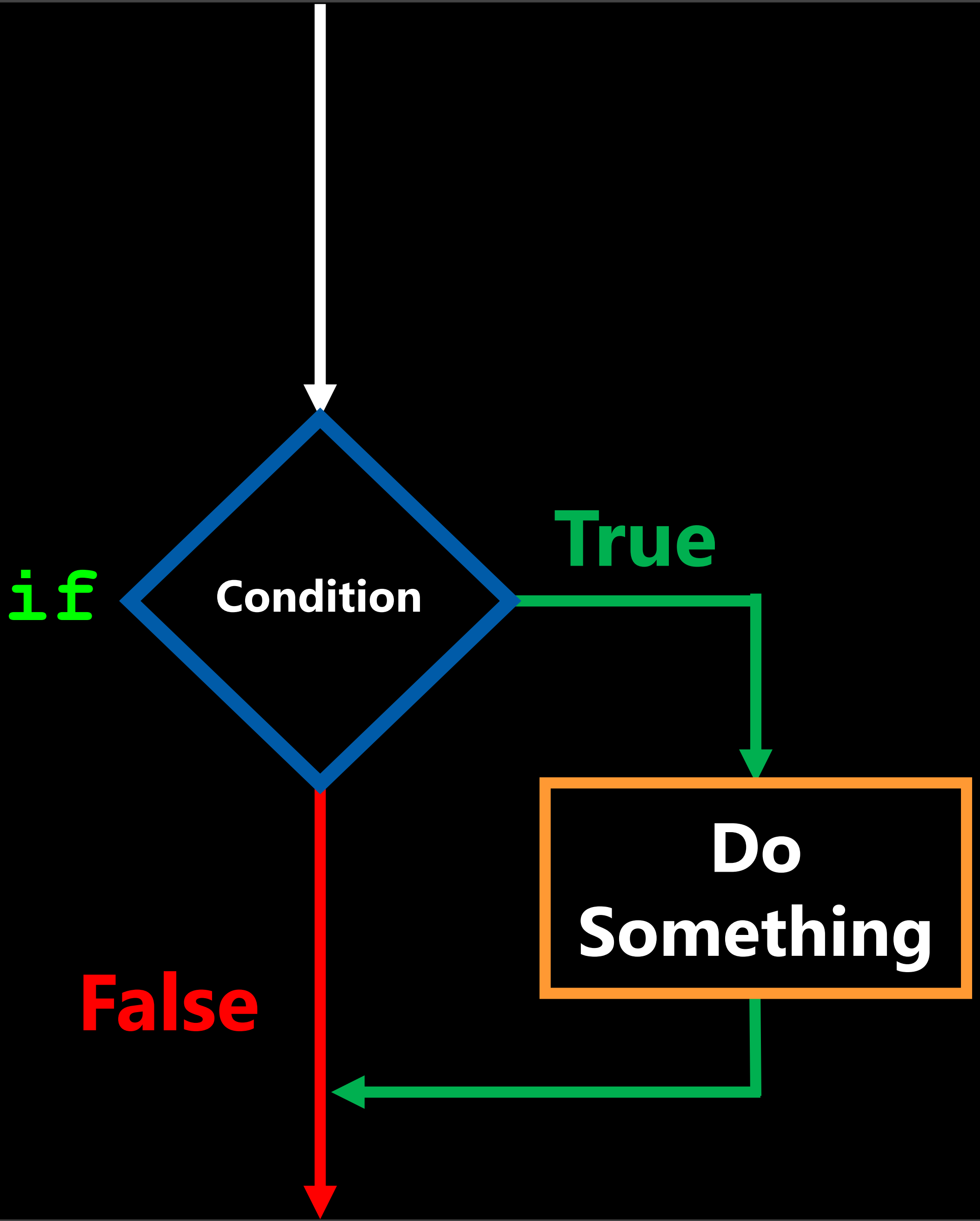


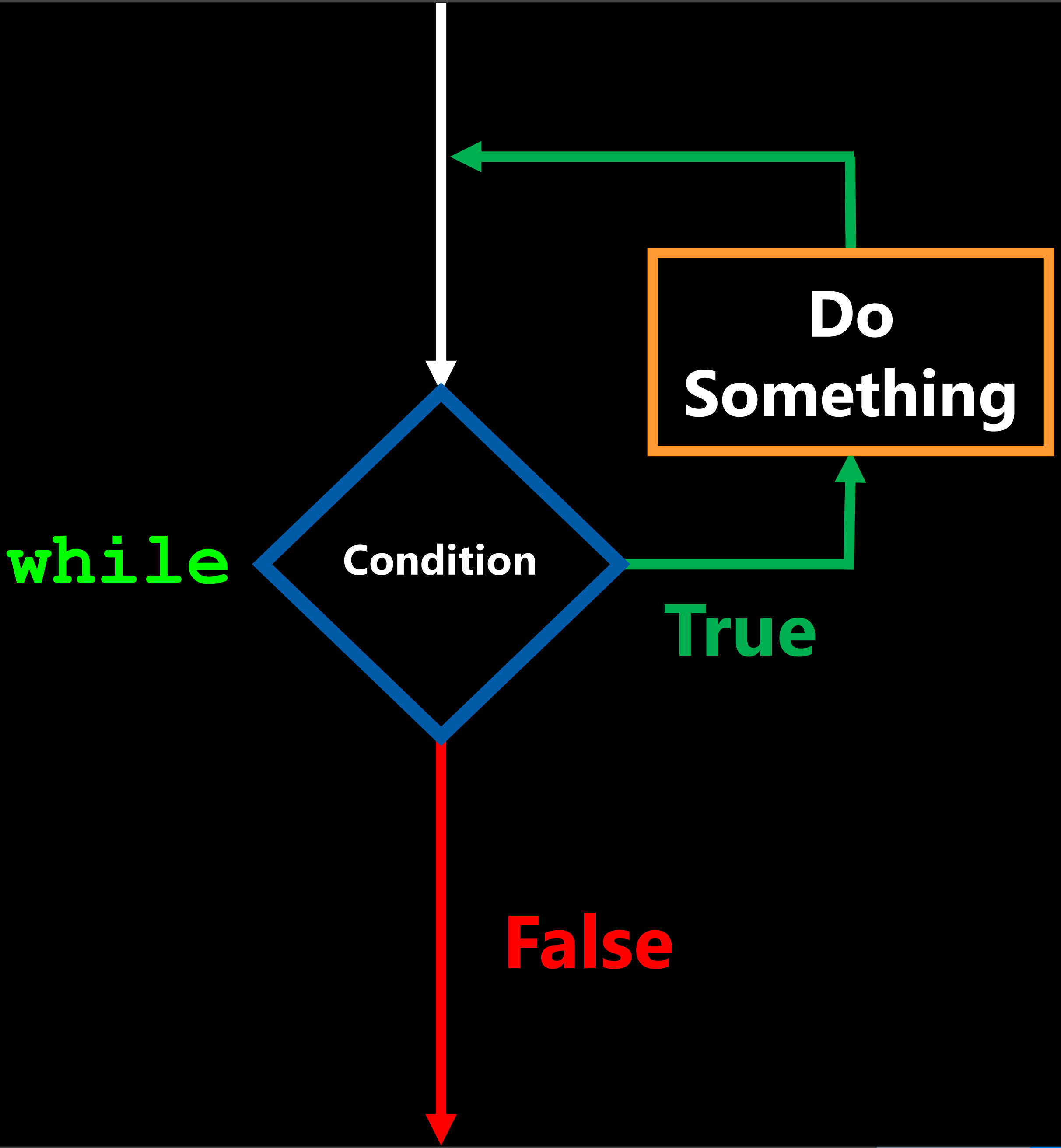
```
while expression:
    do something.
```

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**.

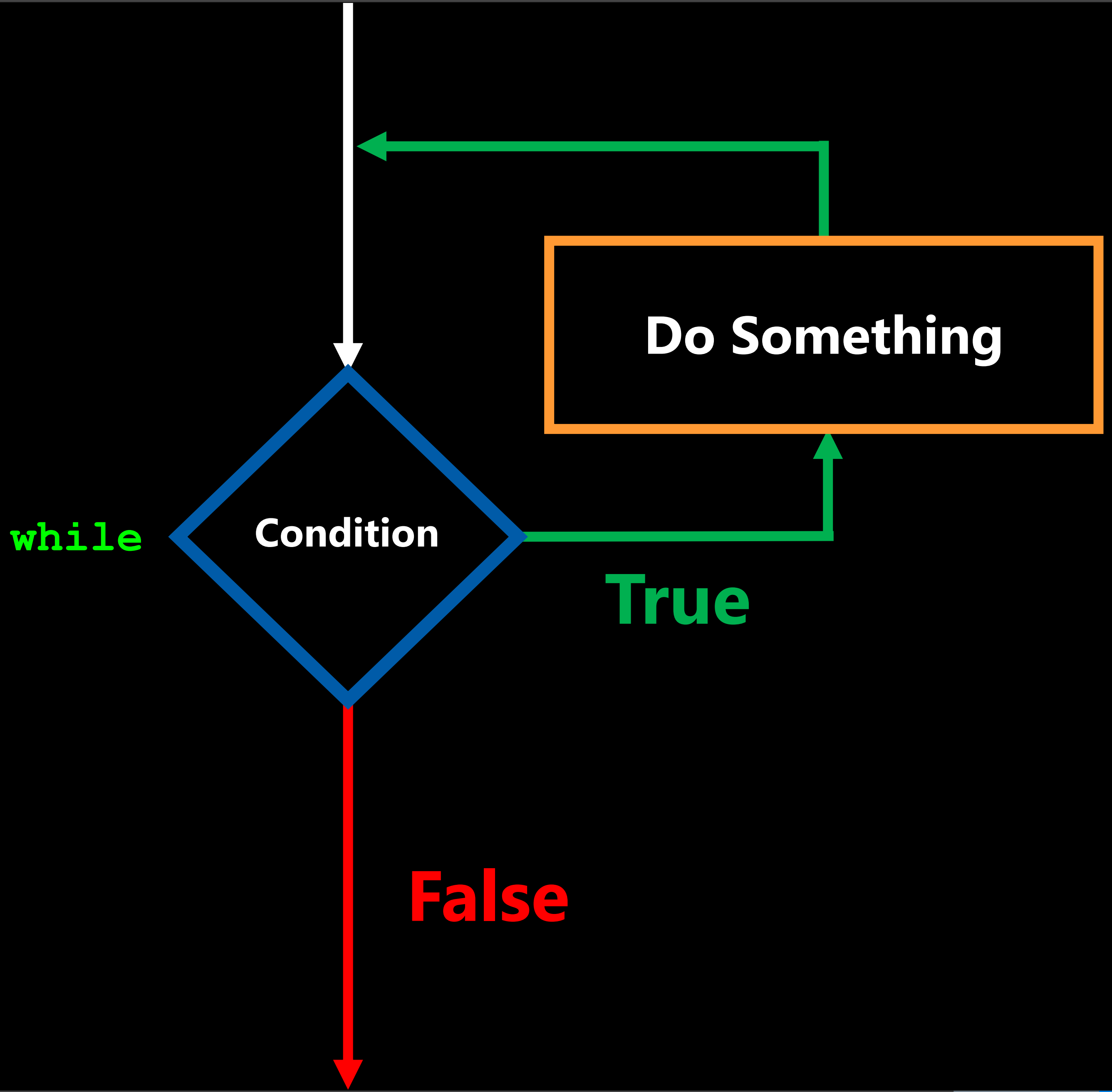



if
Condition
True
Do Something
False


while
Condition
True
Do Something
False

While Loops

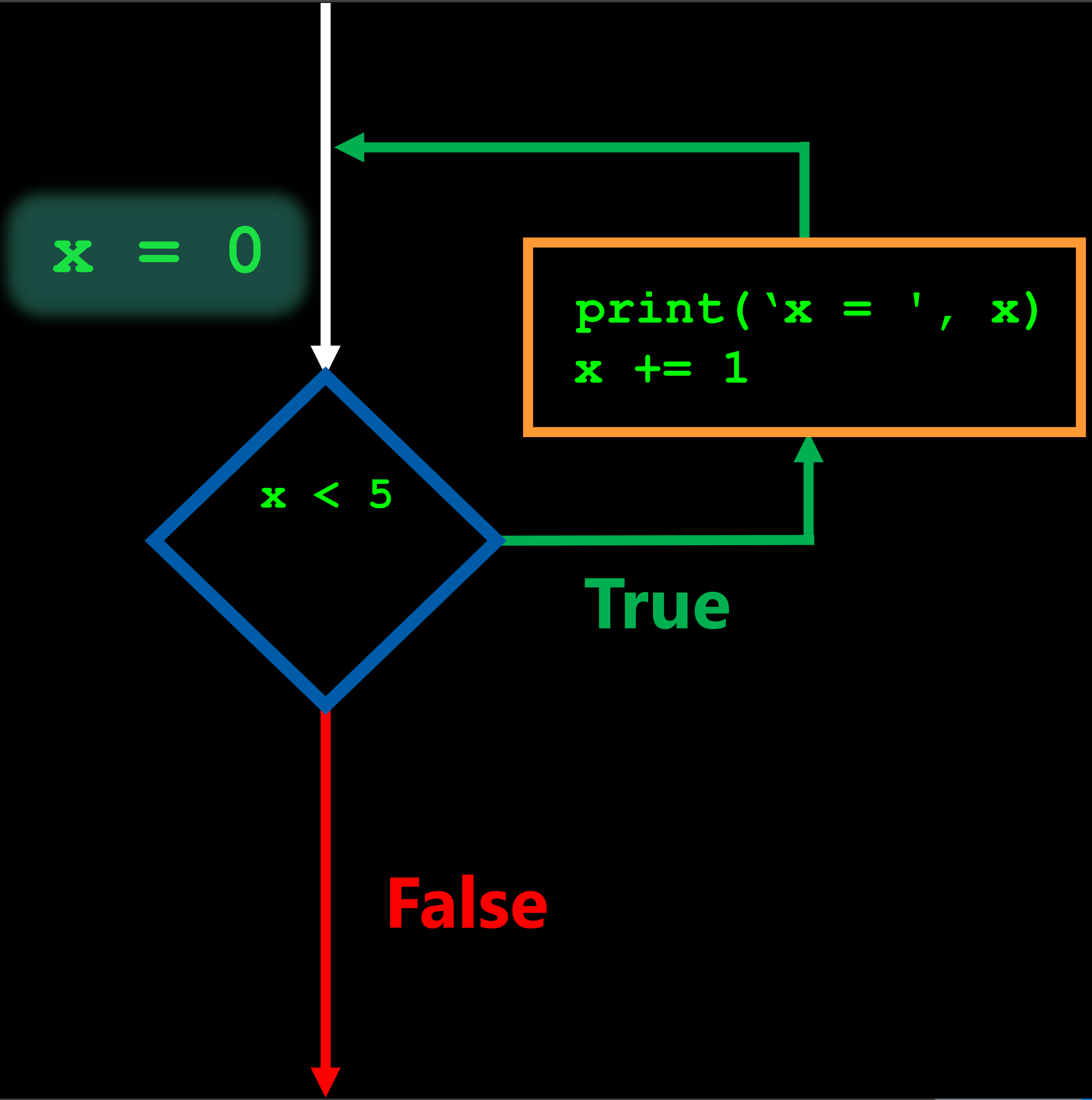
```
x = 0
while x < 5:
   print('x = ', x)
   x += 1
```



While Loops

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

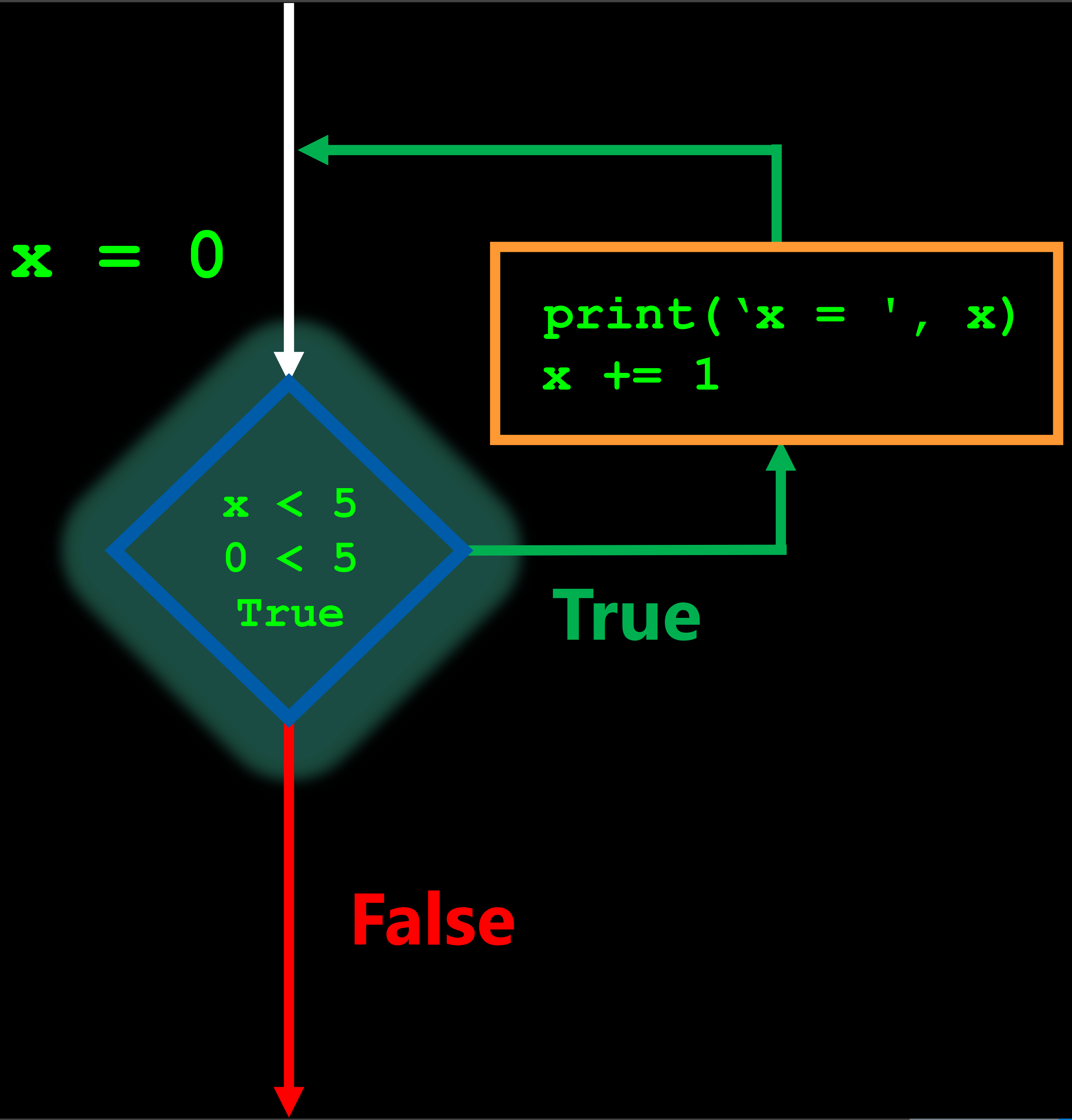
Standard Out.



While Loops

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

Standard Out.

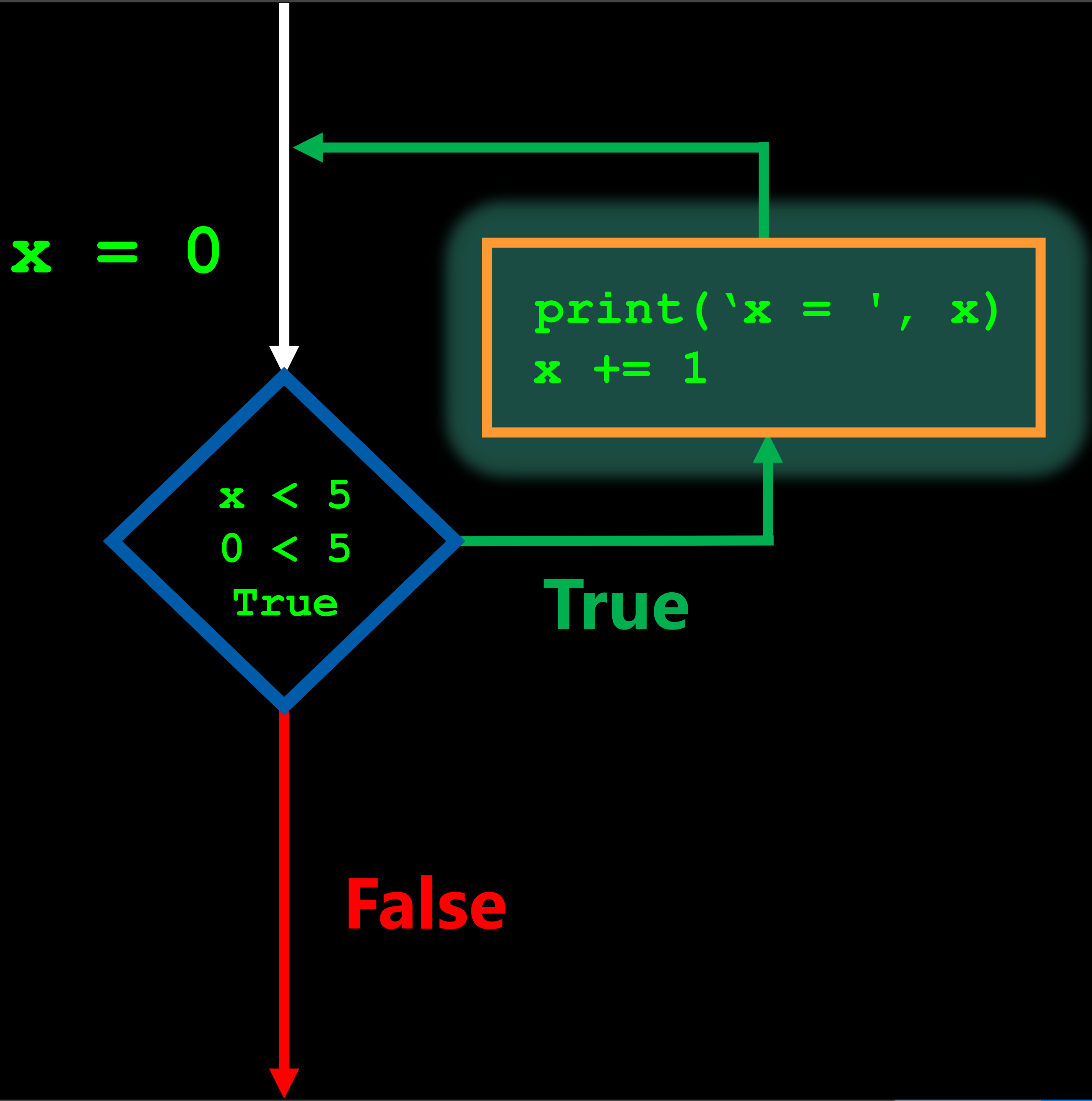


While Loops

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

Standard Out.

```
x = 0
```

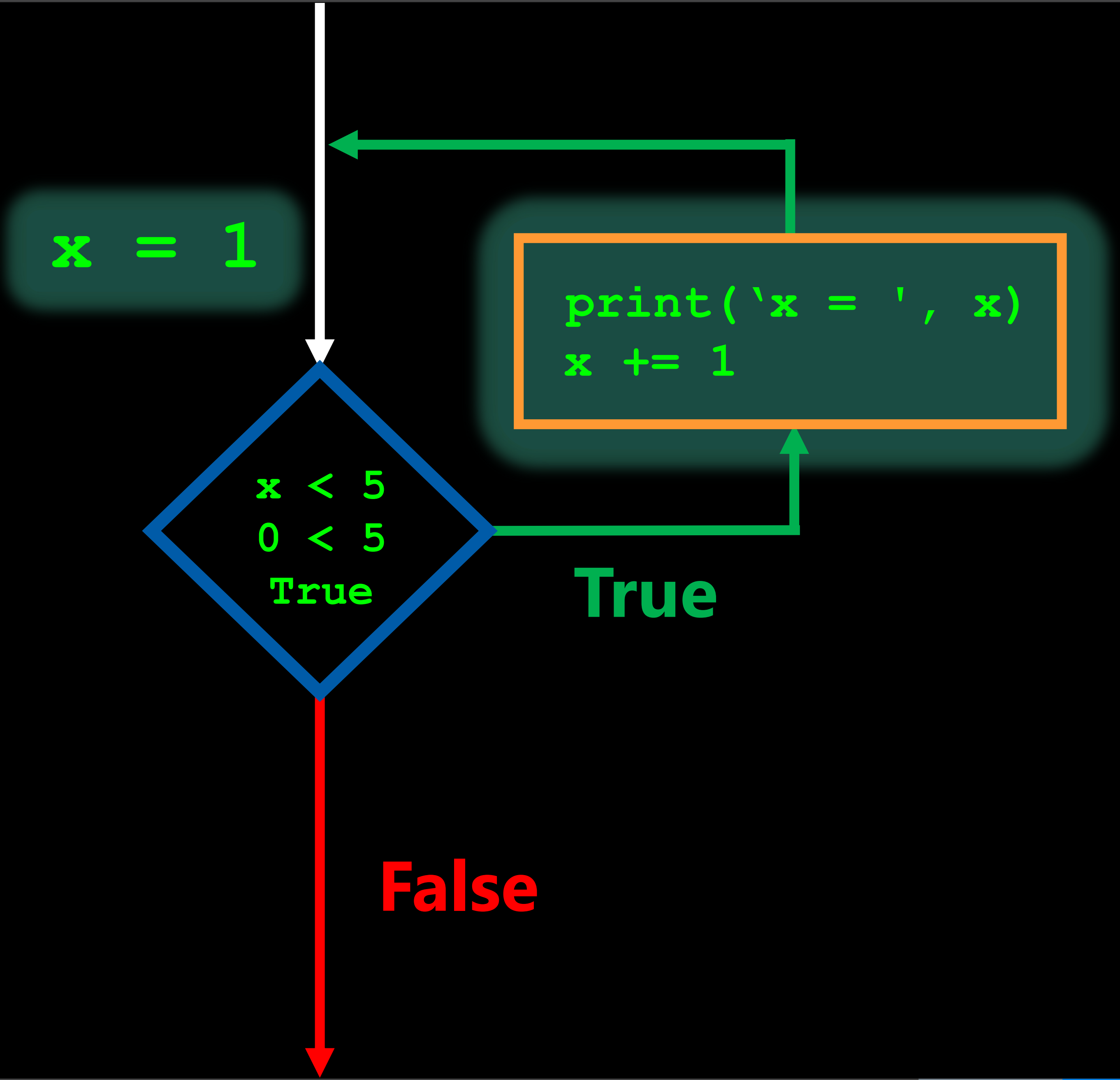


While Loops

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

Standard Out.

```
x = 0
```

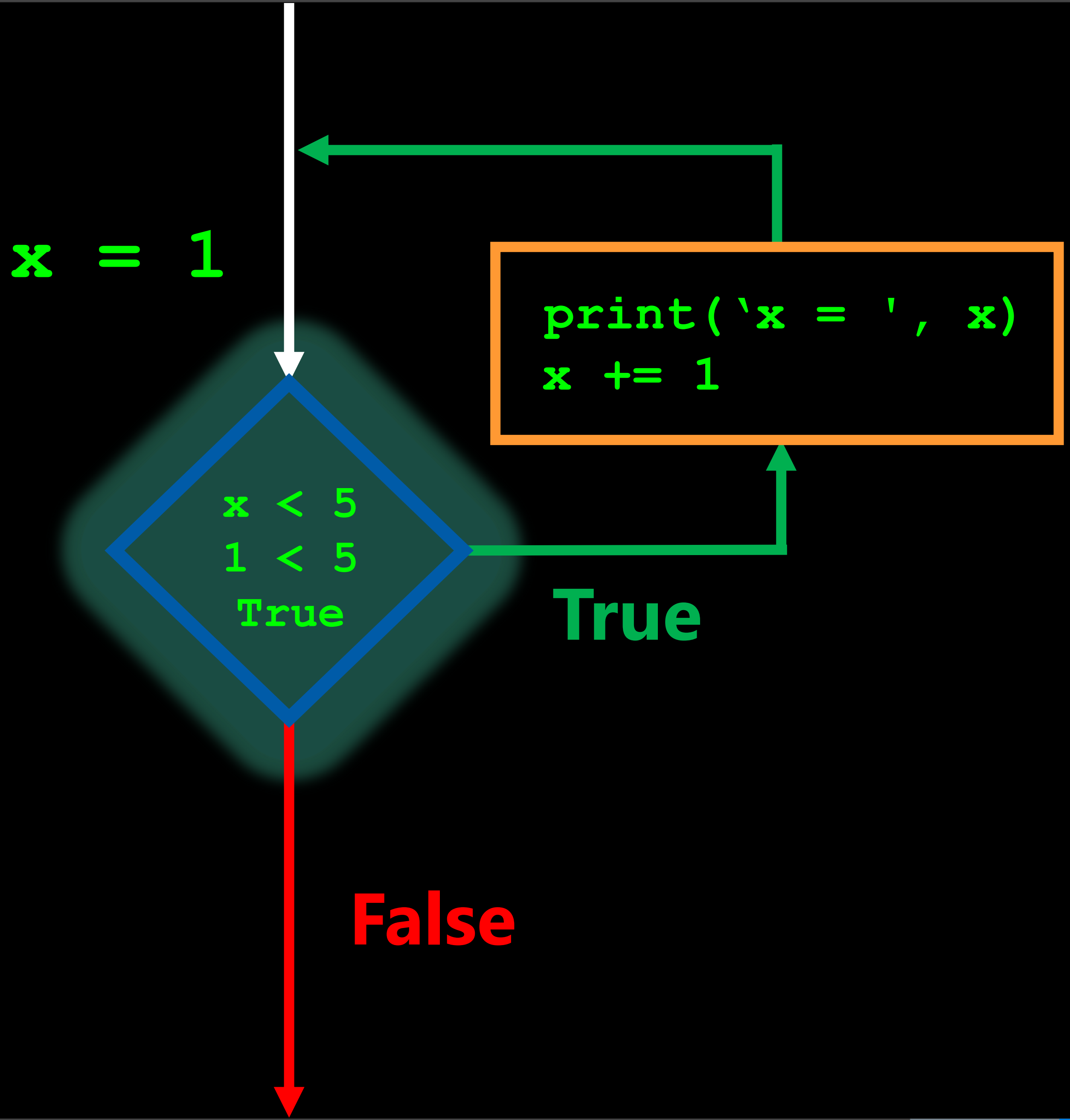


While Loops

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

Standard Out.

```
x = 0
```

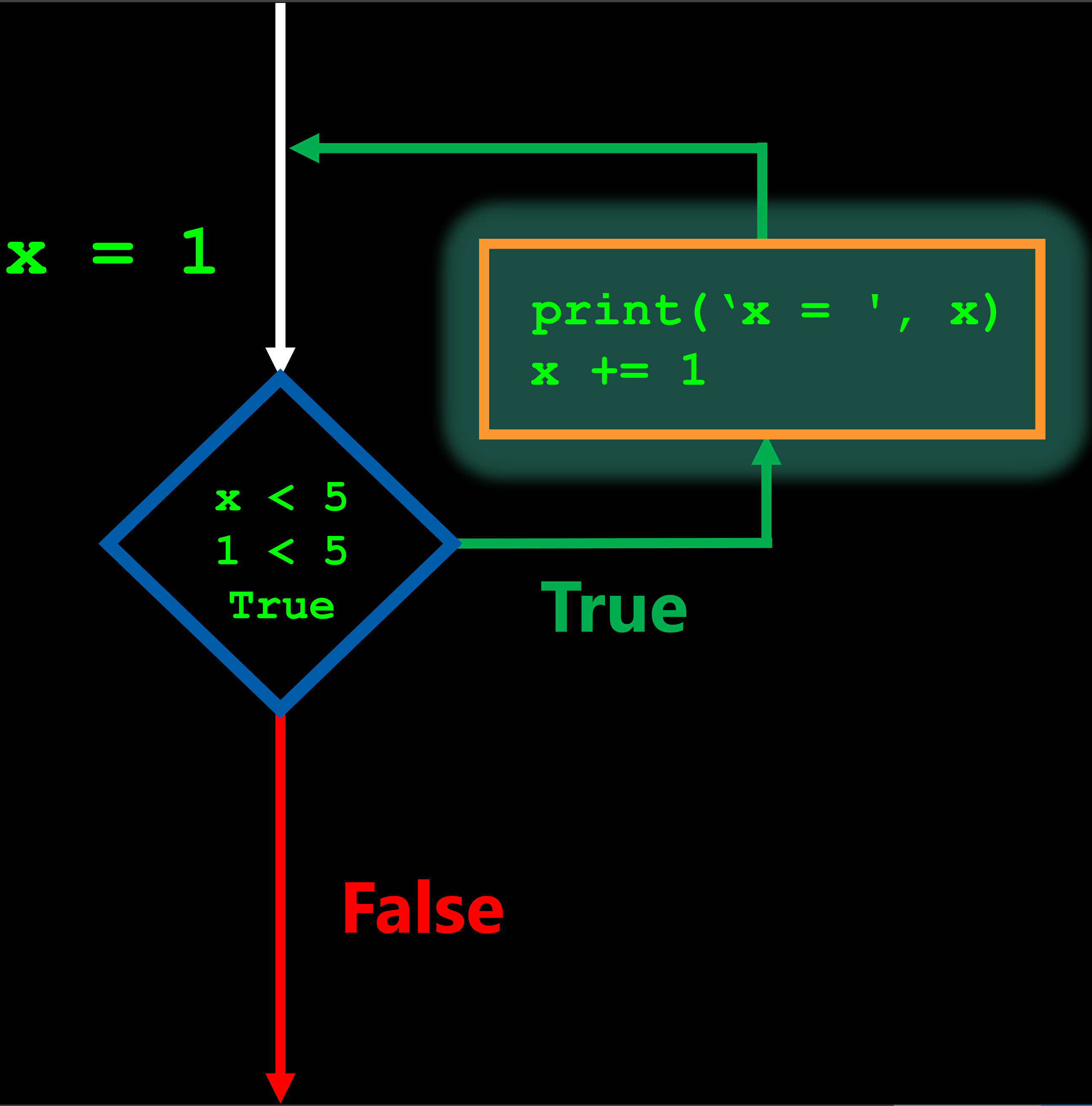


While Loops

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

Standard Out.

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

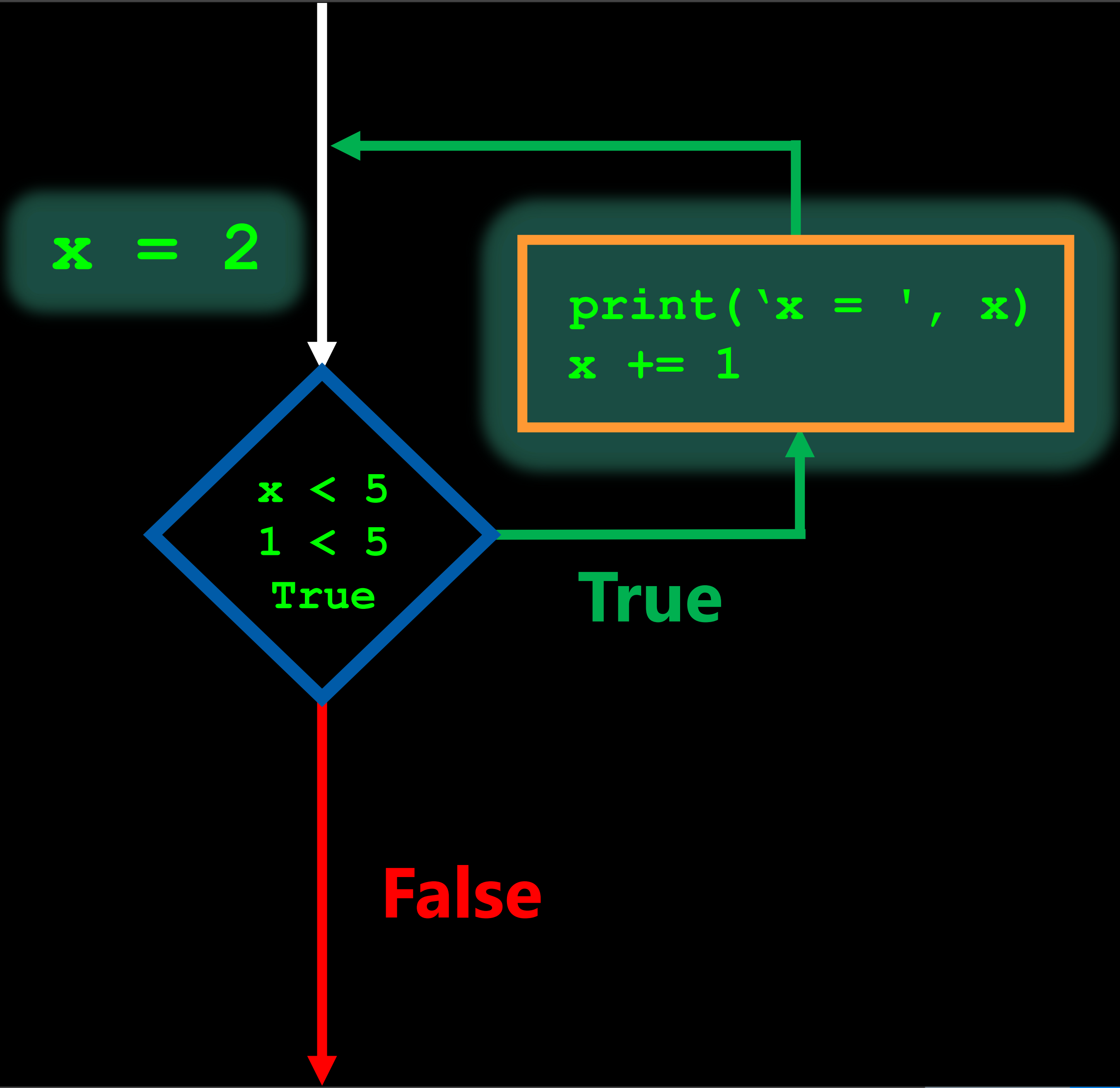


While Loops

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

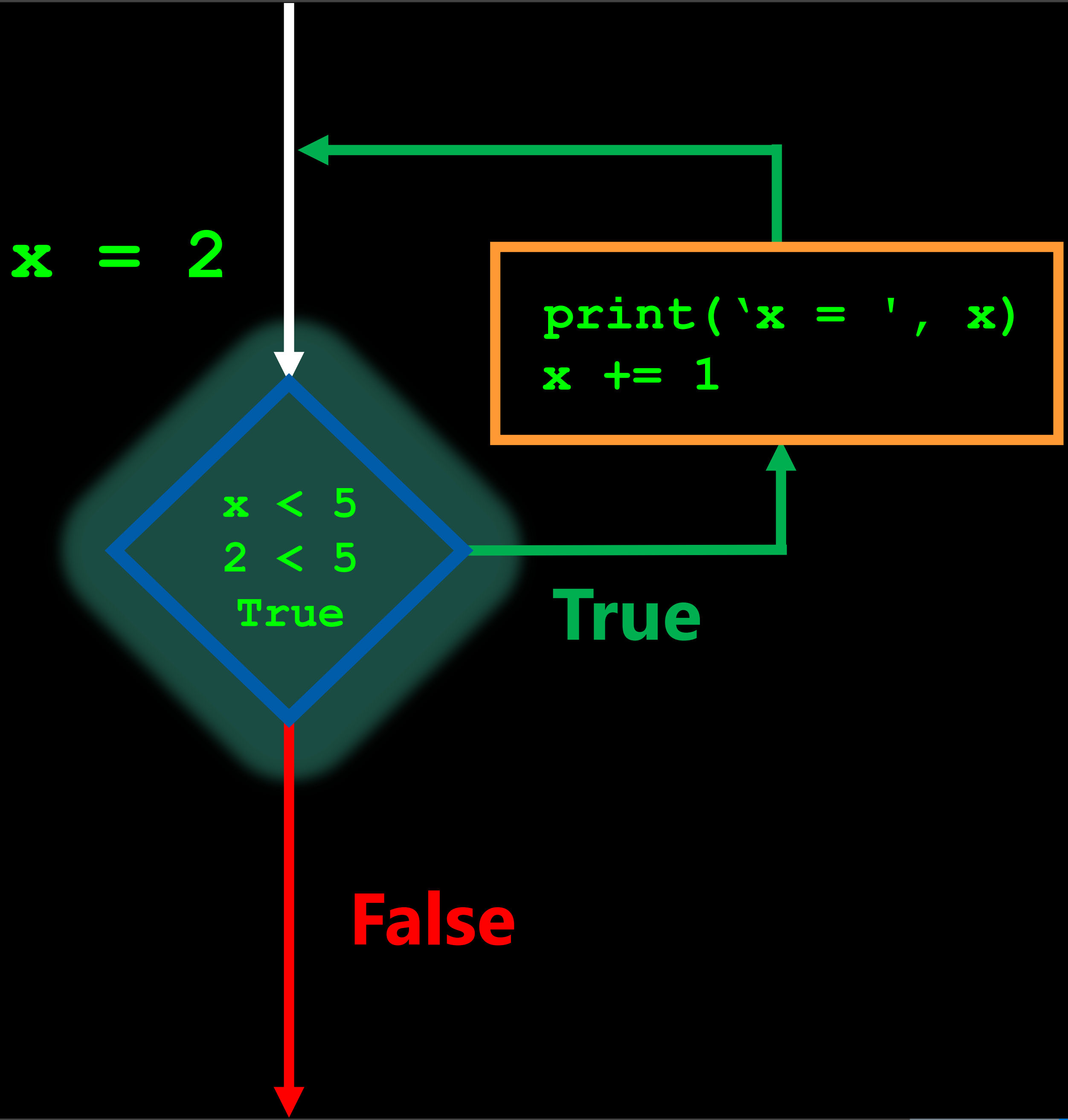
Standard Out.

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



While Loops

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



Standard Out.

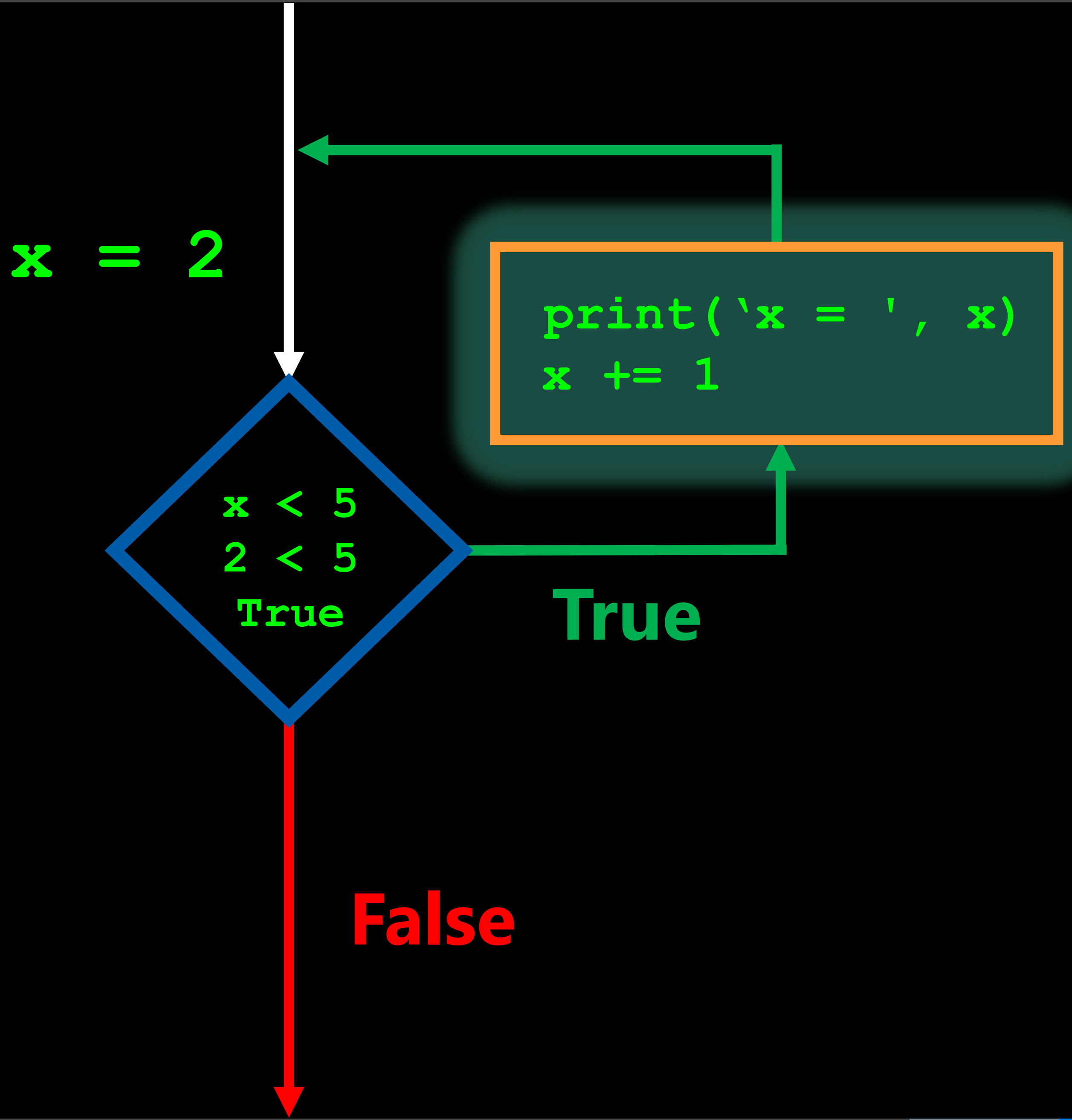
```
x = 0
x = 1
```

While Loops

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

Standard Out.

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

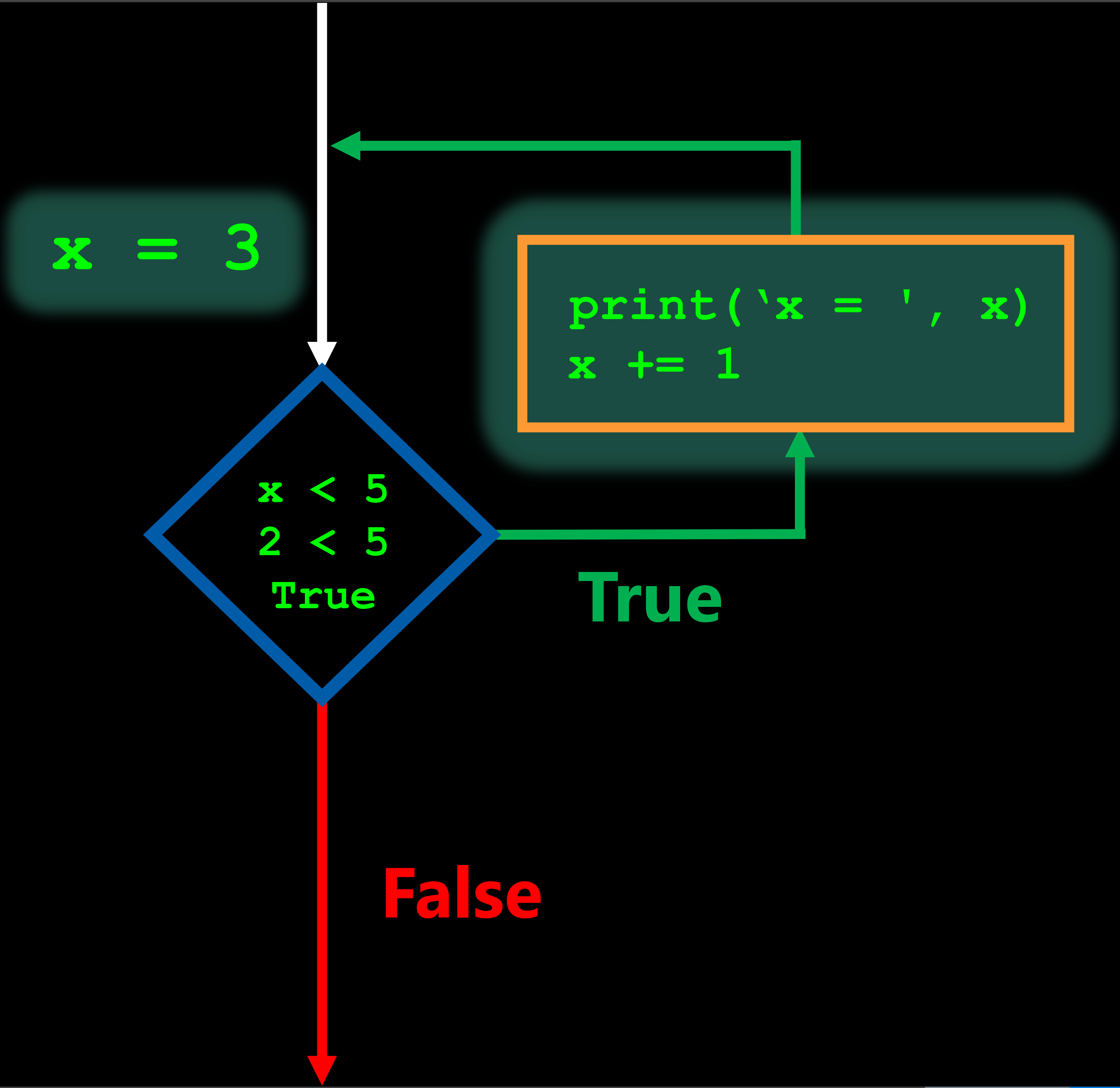


While Loops

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

Standard Out.

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

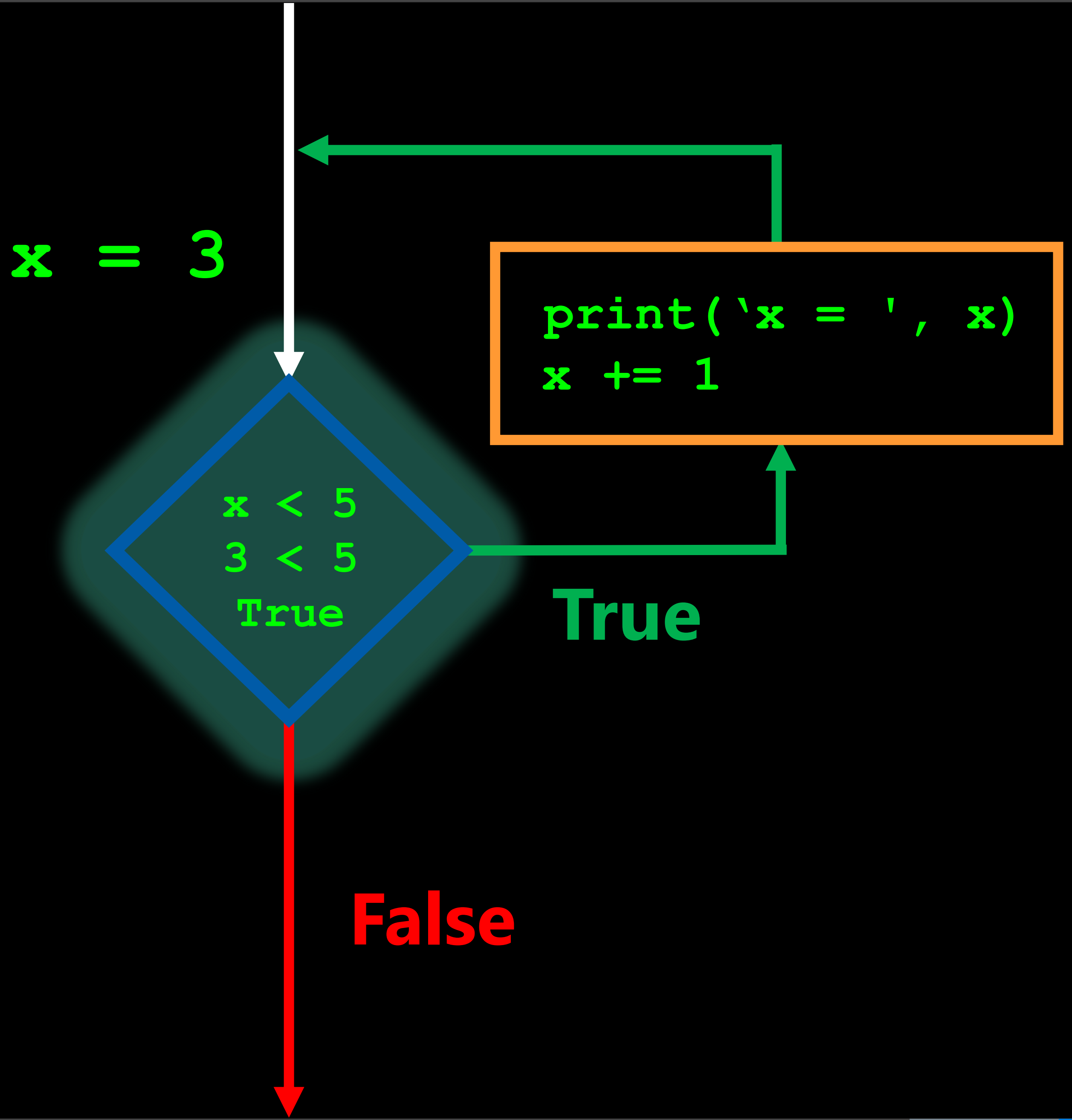


While Loops

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

Standard Out.

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

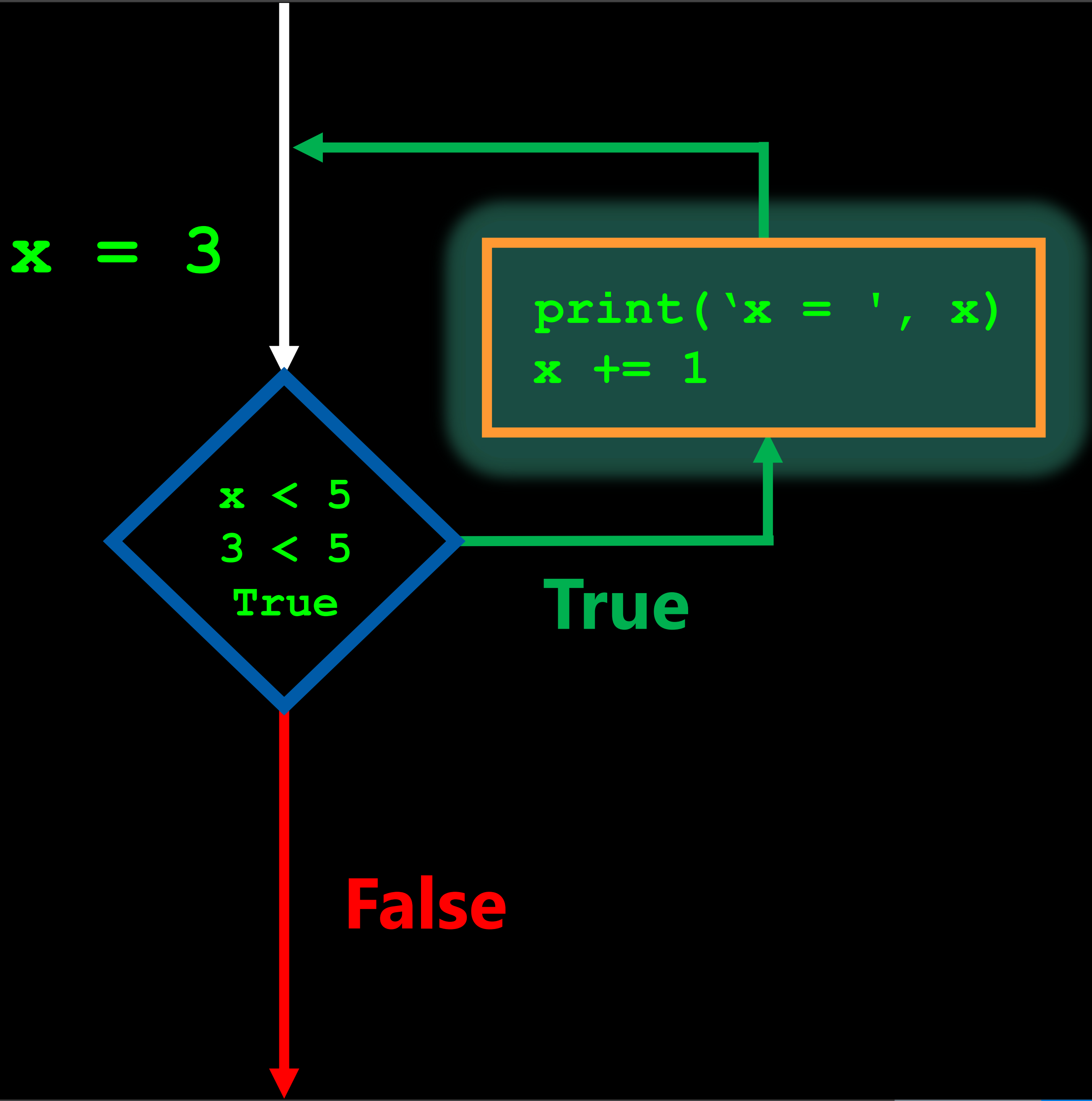


While Loops

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

Standard Out.

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



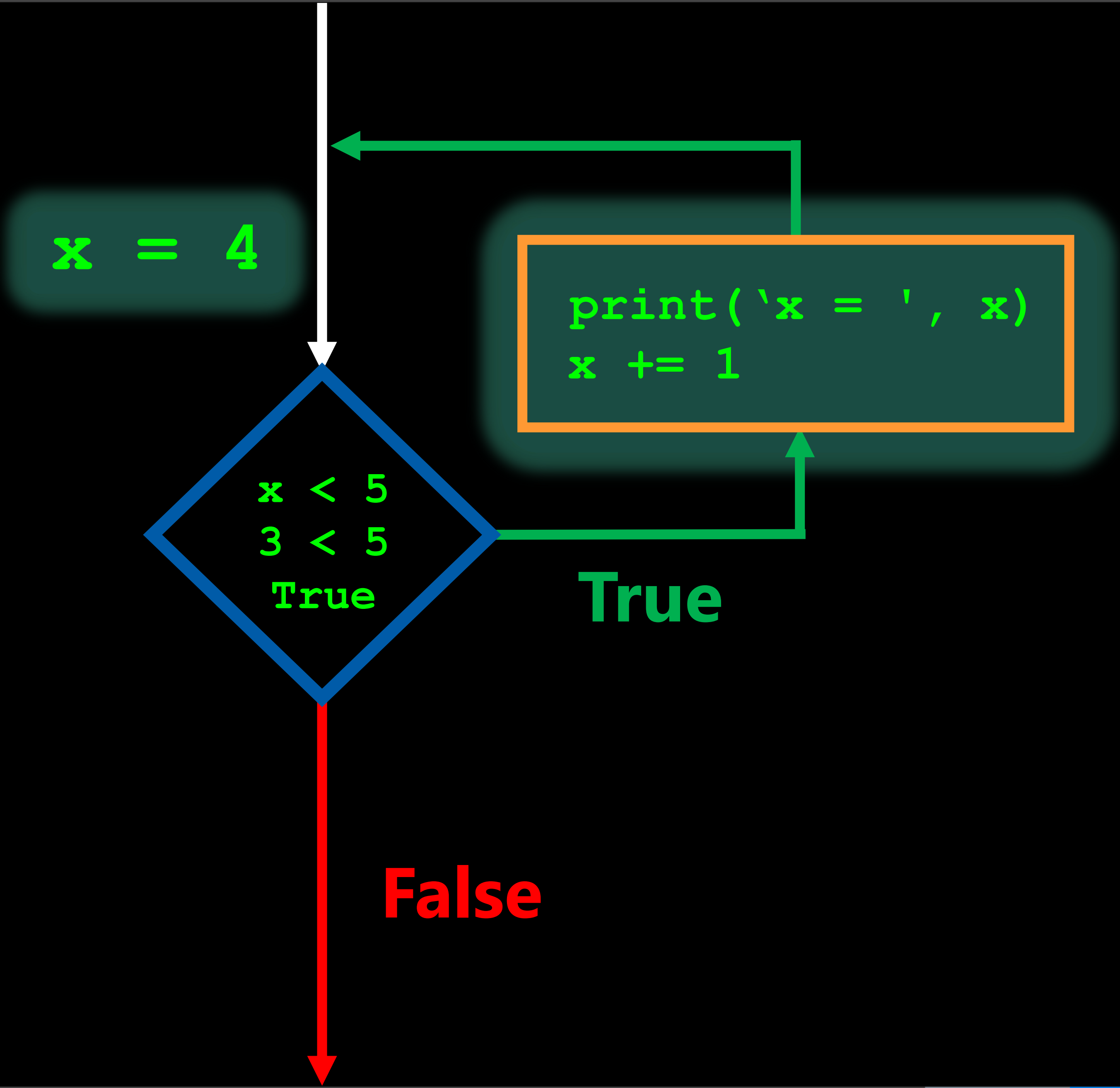
```
print('x = ', x)
x += 1
```

While Loops

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

Standard Out.

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

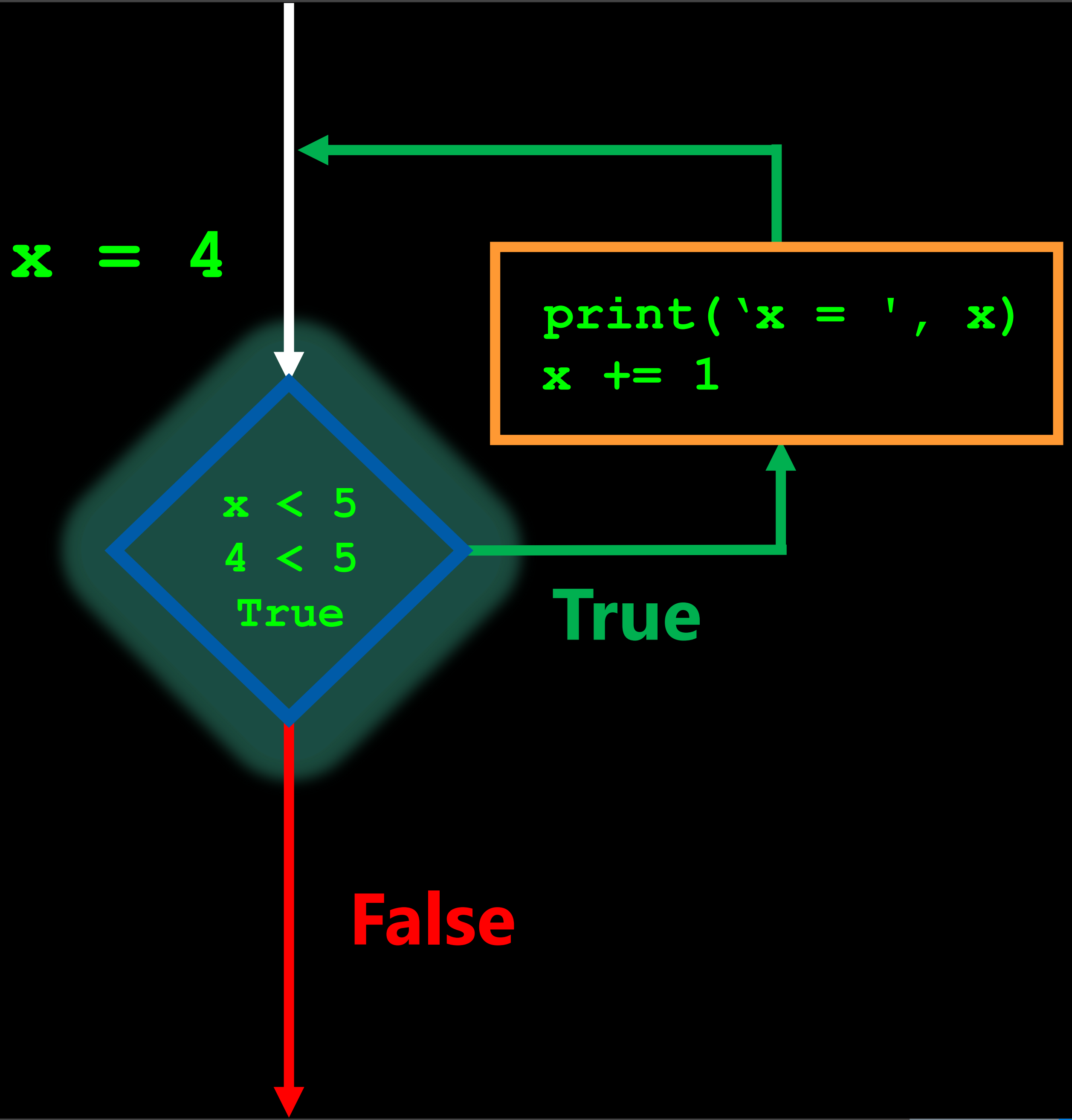


While Loops

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

Standard Out.

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

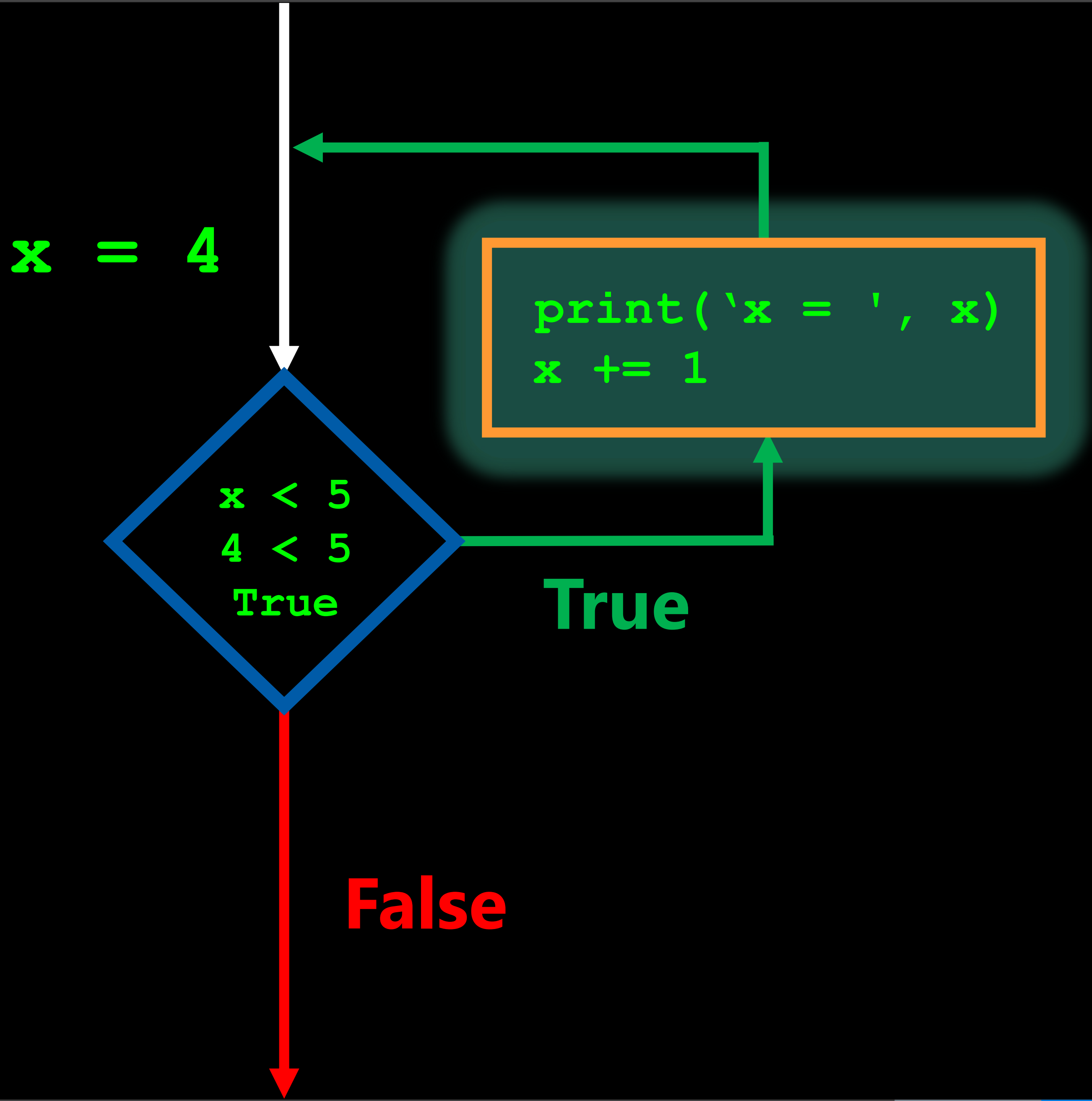


While Loops

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

Standard Out.

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

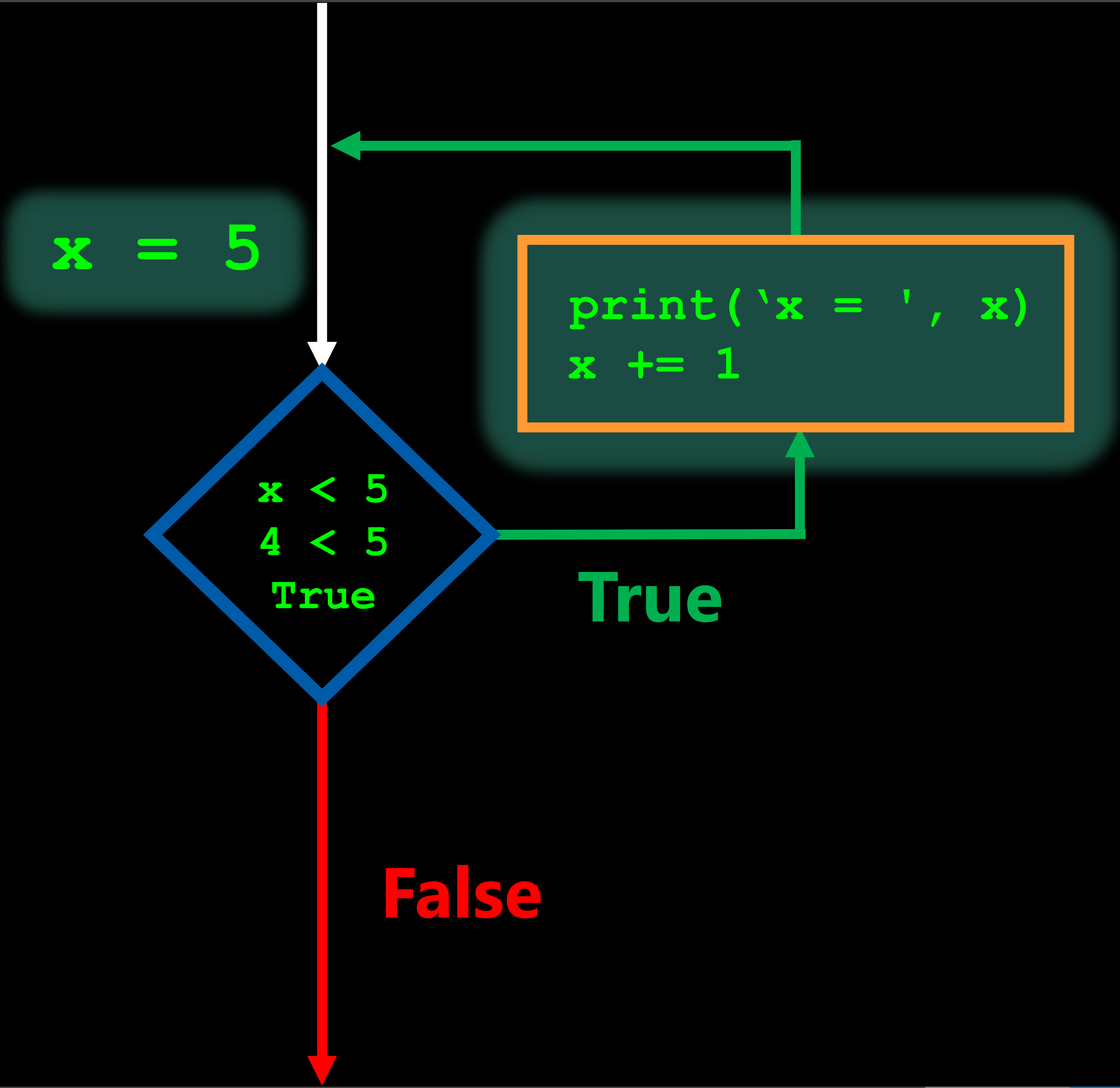


While Loops

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

Standard Out.

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

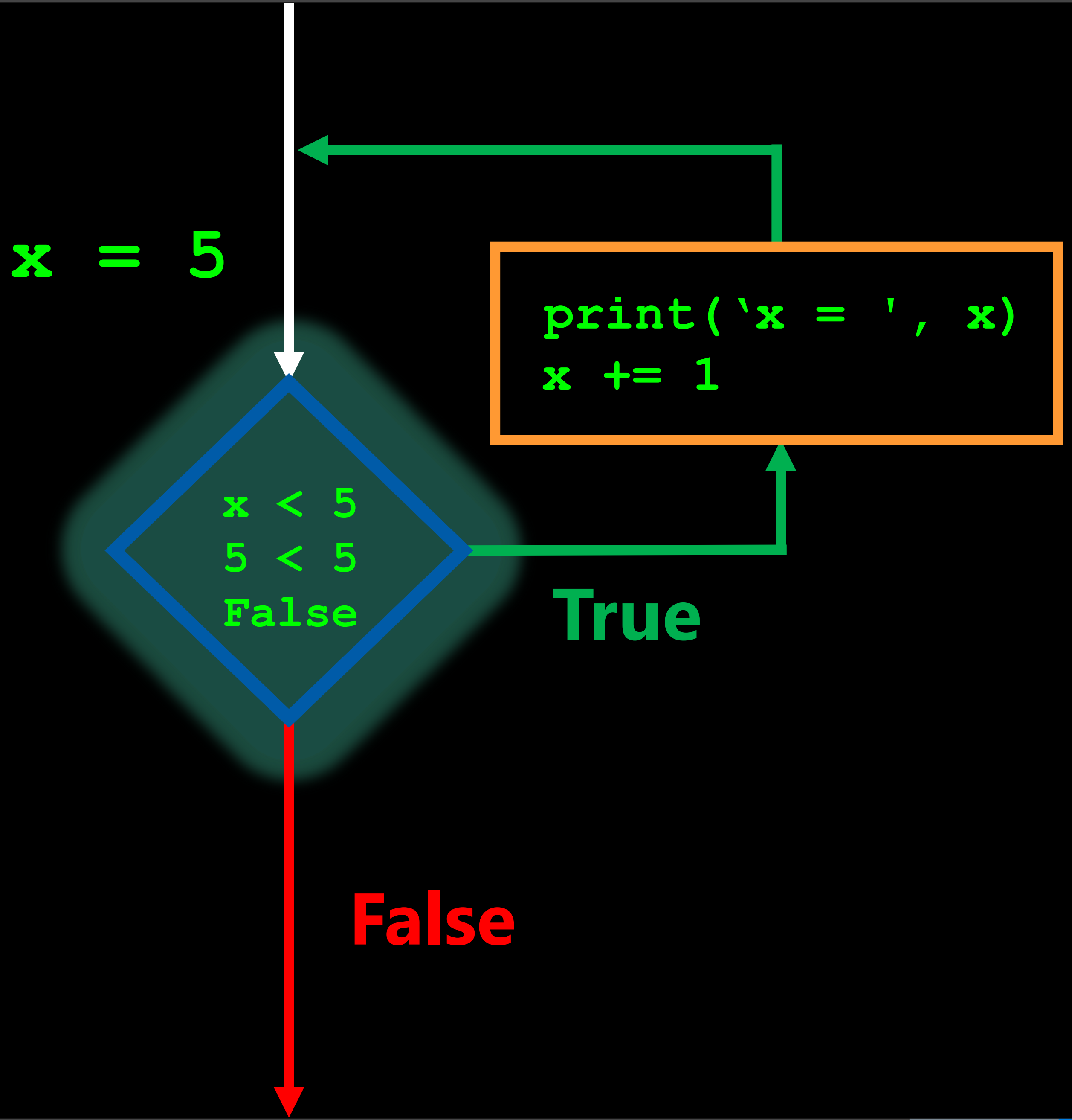


While Loops

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

Standard Out.

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

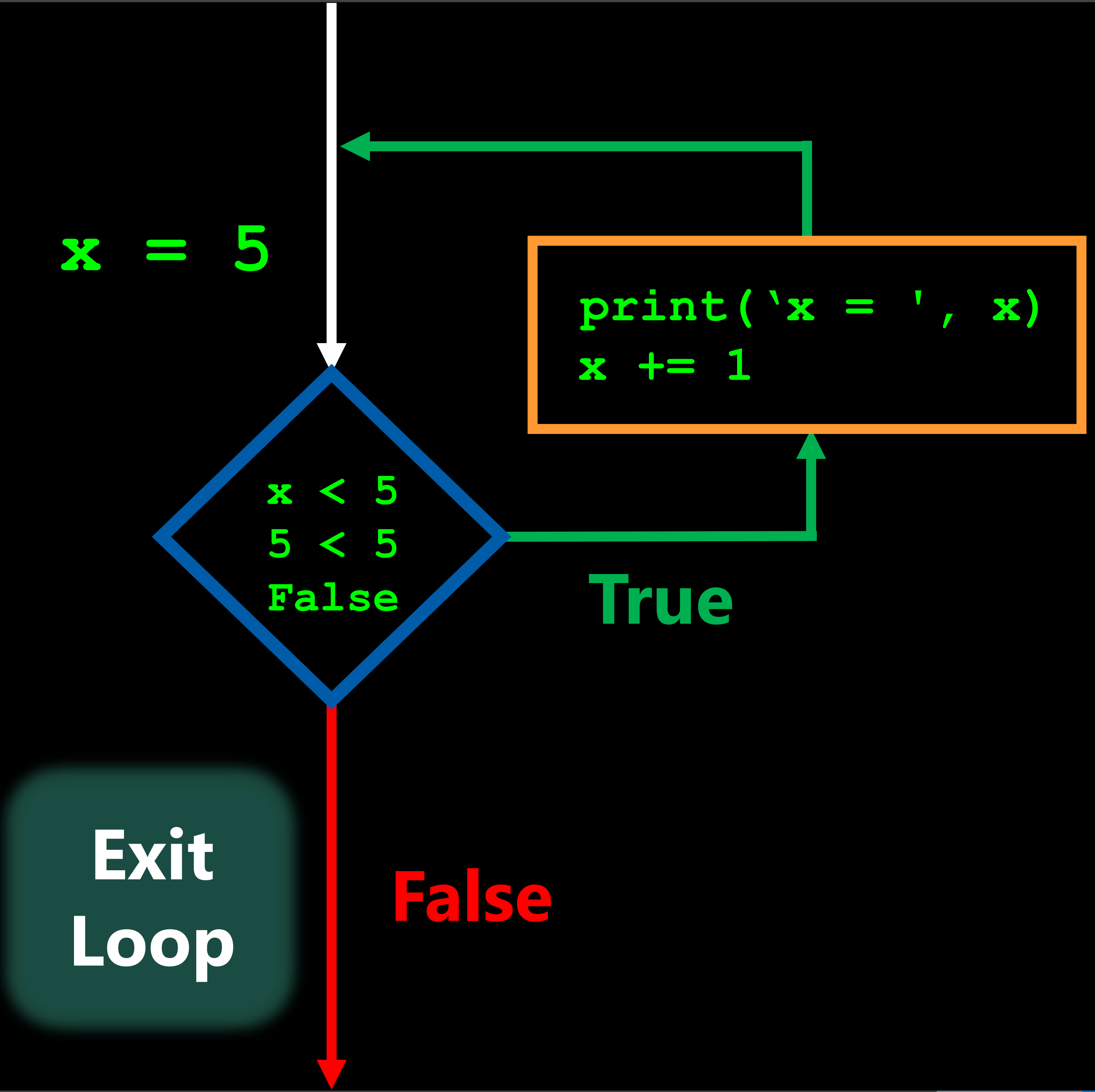


While Loops

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

Standard Out.

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



APS106

UNIVERSITY OF TORONTO

functions, input & output, importing modules.

Week 4 | Lecture 1 (4.1)

if nothing else, write #cleancode