CS1070: Taming Big Data

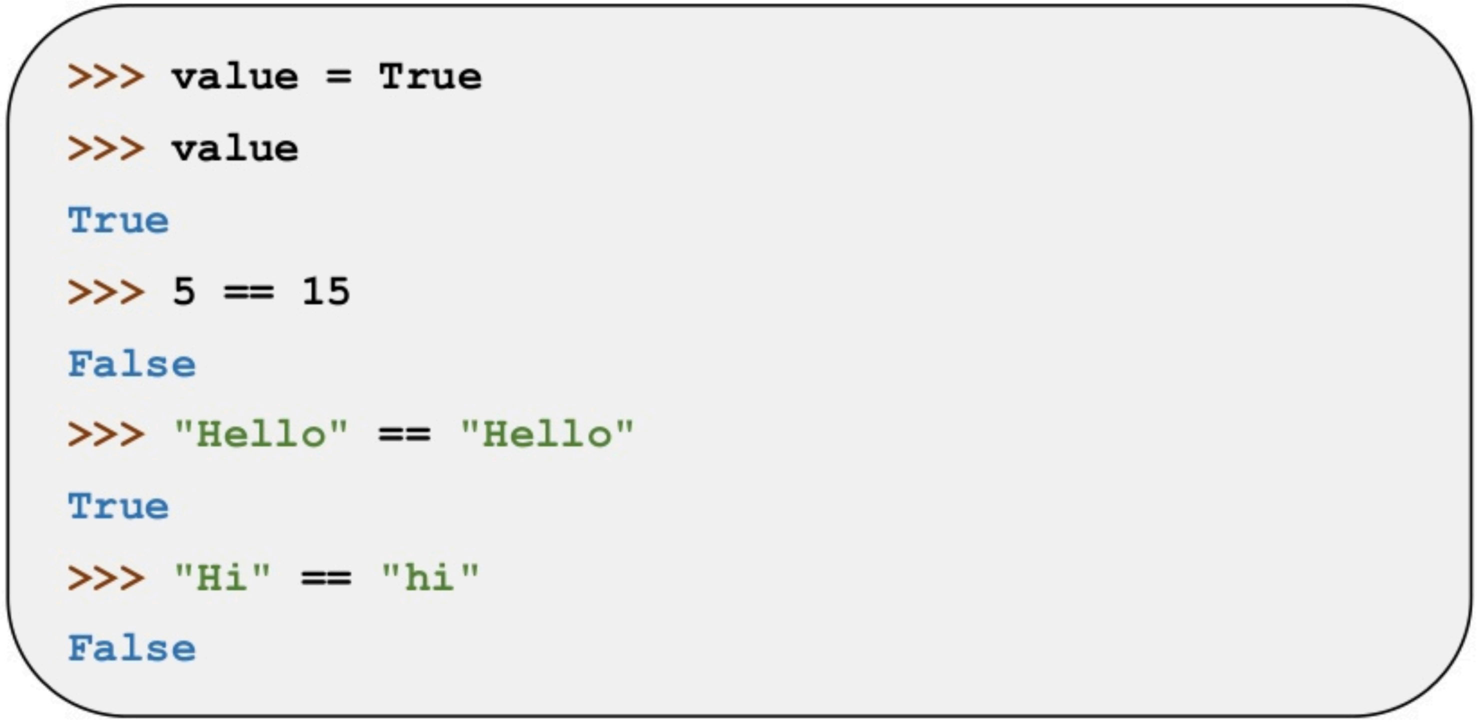
Comparisons, Conditionals, Control Structures

Logistics

- Try using lab machines during class hours
- Homework 1 out today, due in 1 week

Booleans & Comparisons

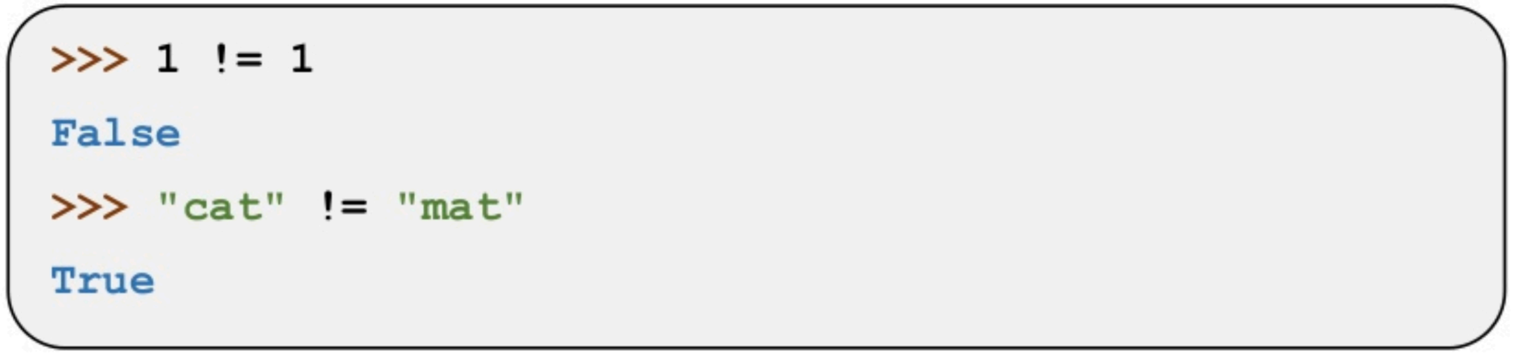
- Two Boolean values: **True** and **False**
- Can create directly or through comparisons:



```
>>> value = True
>>> value
True
>>> 5 == 15
False
>>> "Hello" == "Hello"
True
>>> "Hi" == "hi"
False
```

Booleans & Comparisons

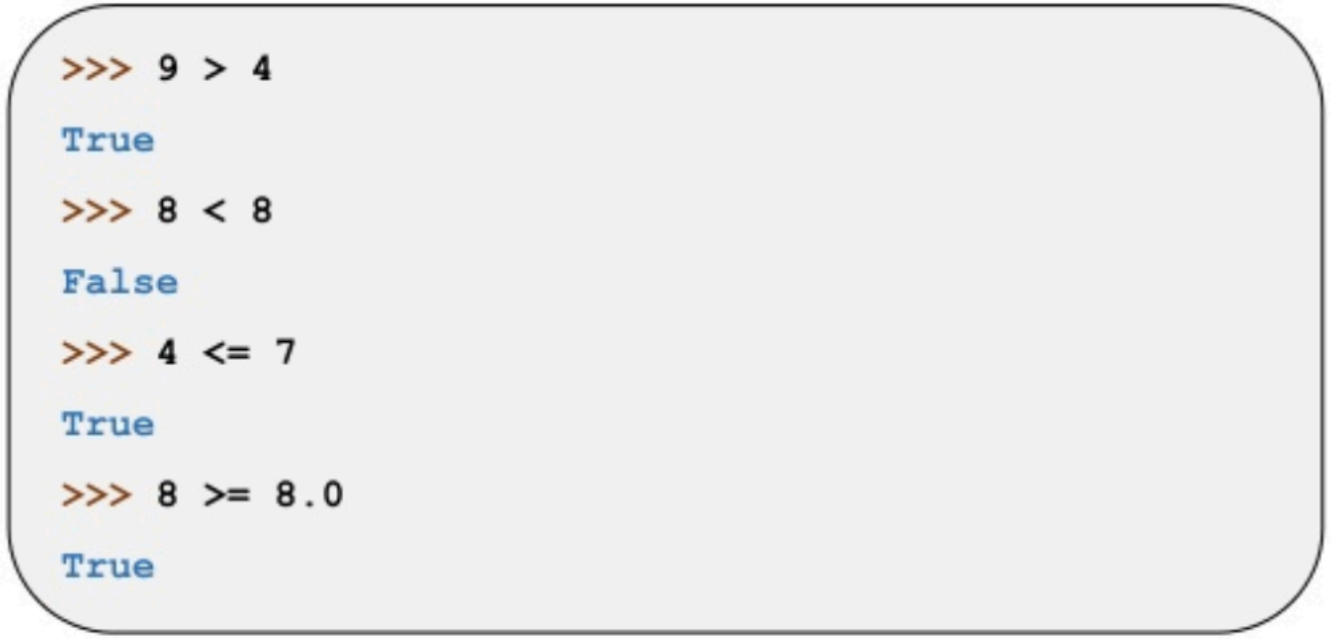
- 'is equal' comparison: **==**
  - Evaluates to true if items being compared (left and right sides) are equal, and False if they're not
- 'is not equal': **!=**
  - Evaluates to true if items being compared (left and right) are **NOT** equal, and False if they are



```
>>> 1 != 1
False
>>> "cat" != "mat"
True
```

Booleans & Comparisons

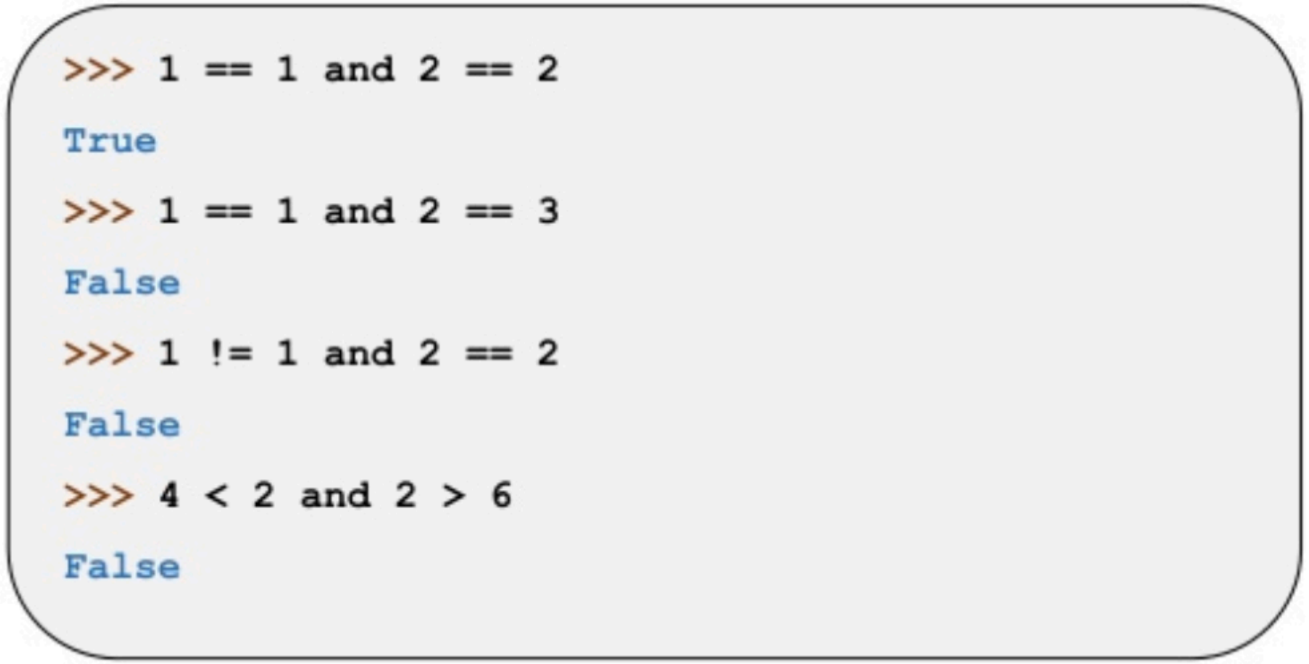
- Comparisons work for numbers, strings, bools, etc
- Additional comparisons: < , >, <=, >=



```
>>> 9 > 4
True
>>> 8 < 8
False
>>> 4 <= 7
True
>>> 8 >= 8.0
True
```

Boolean Logic

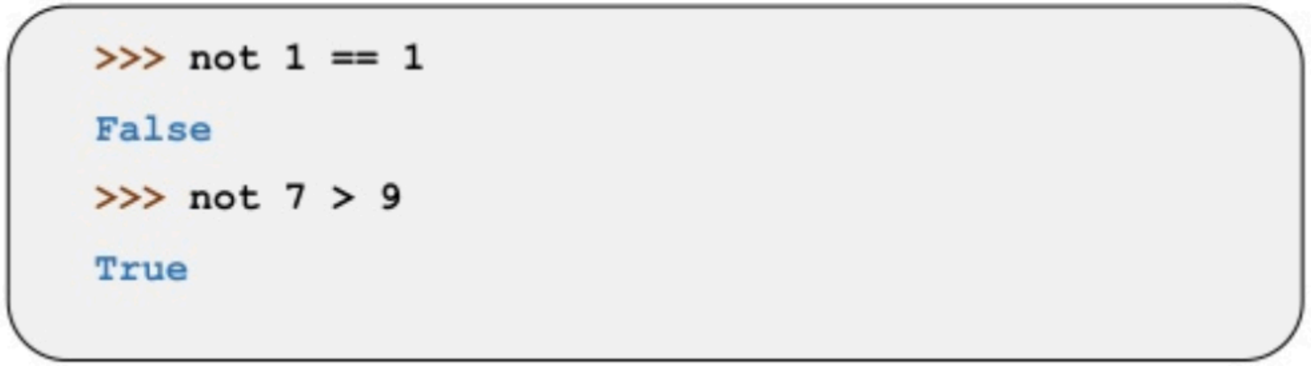
- Boolean operators: and, or and not
- and: takes two arguments, and returns True if and only if both arguments are True; otherwise returns False



```
>>> 1 == 1 and 2 == 2
True
>>> 1 == 1 and 2 == 3
False
>>> 1 != 1 and 2 == 2
False
>>> 4 < 2 and 2 > 6
False
```

Boolean Logic

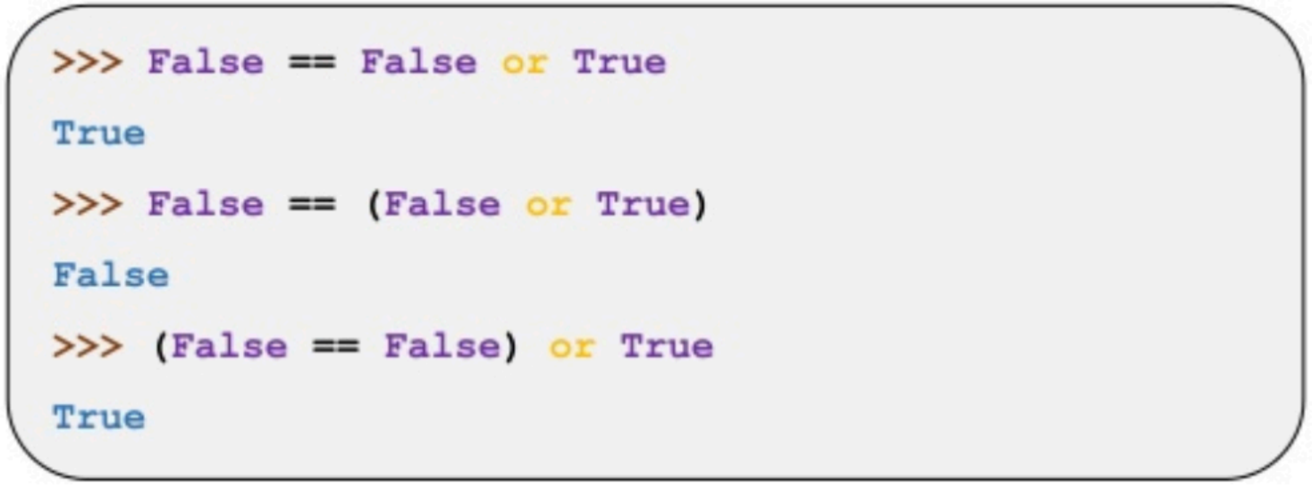
- or: takes in two arguments and returns True if either (or both) arguments are True; otherwise returns False (if both arguments are False)
- not: if an argument is True, returns False; if it’s False, returns True



```
>>> not 1 == 1
False
>>> not 7 > 9
True
```

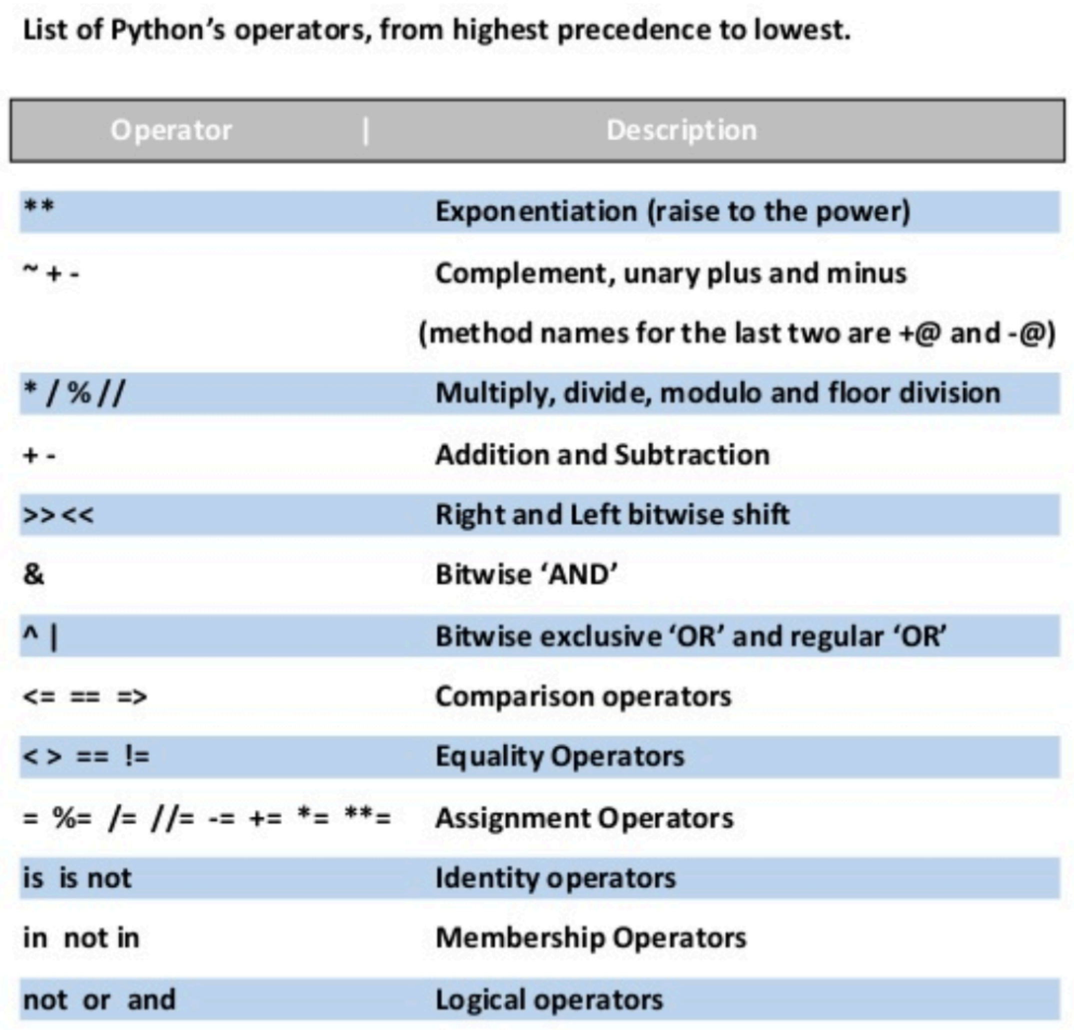
Operator Precedence

- Extension of mathematical order of operations
- == / != have higher precedence than Boolean logic operators (and/or)



```
>>> False == False or True
True
>>> False == (False or True)
False
>>> (False == False) or True
True
```

Operator Precedence



List of Python's operators, from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
	(method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and Subtraction
>> <<	Right and Left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive 'OR' and regular 'OR'
<= == =>	Comparison operators
< > == !=	Equality Operators
= %= /= //= -= += *= **=	Assignment Operators
is is not	Identity operators
in not in	Membership Operators
not or and	Logical operators

Control Structures

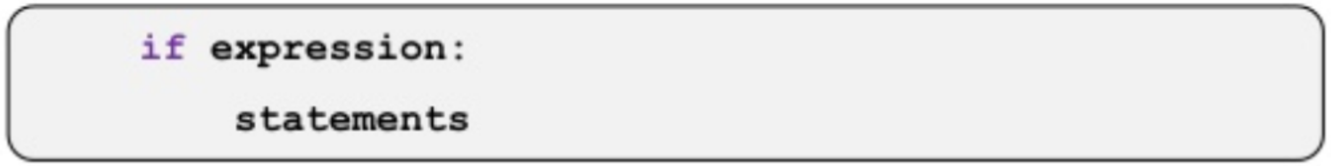
- Direct the order of execution of statements in a program

- conditionals (if / else): “If the weather is nice, I will mow the lawn, otherwise I’ll watch tv”

- iterators / loops:
  - for loop: for every element in X, do Y
  - while loop: while *condition* is True, do Y

If Statements

- Use ***if*** statements to run code if a condition is True

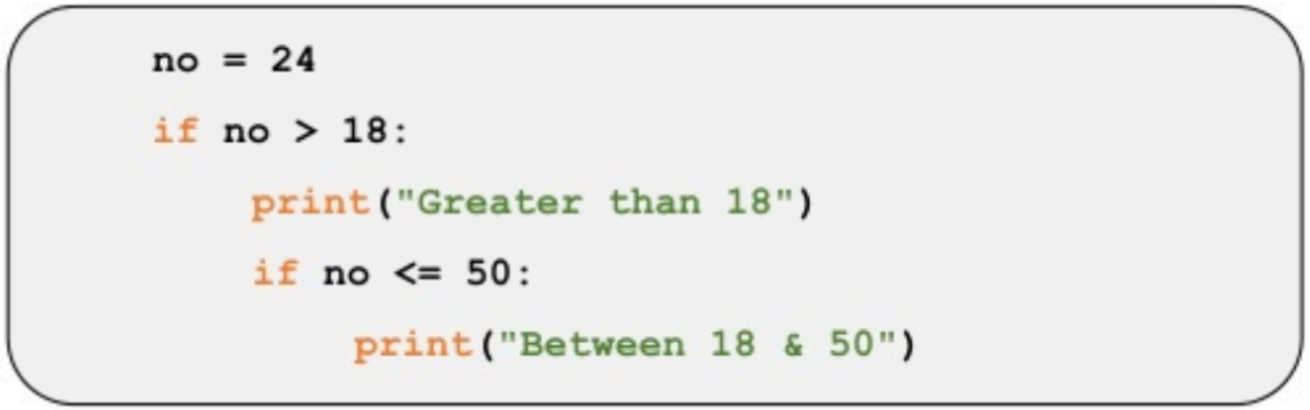


```
if expression:
    statements
```

- Python uses indentation (white space at start of the line) to note blocks of code
- Note the colon at the end of the if statement

If Statements

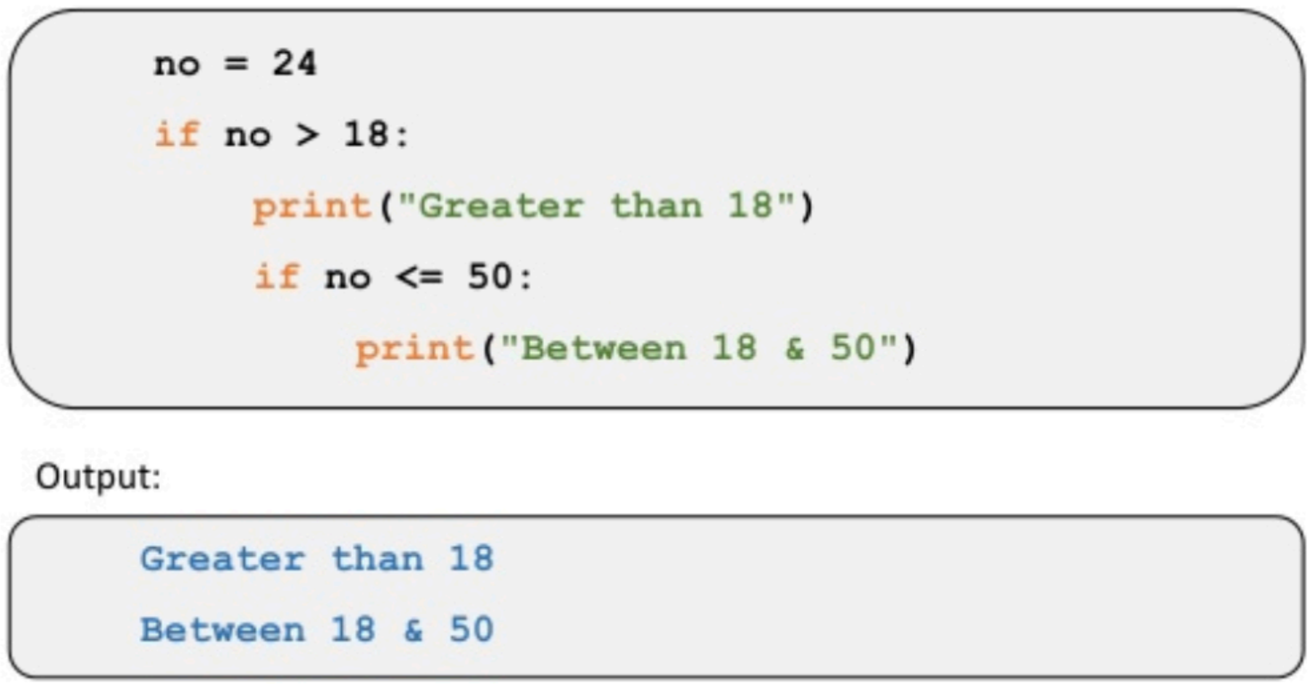
- If statements can be nested to perform more complicated checks



```
no = 24
if no > 18:
    print("Greater than 18")
    if no <= 50:
        print("Between 18 & 50")
```

If Statements

- If statements can be nested to perform more complicated checks



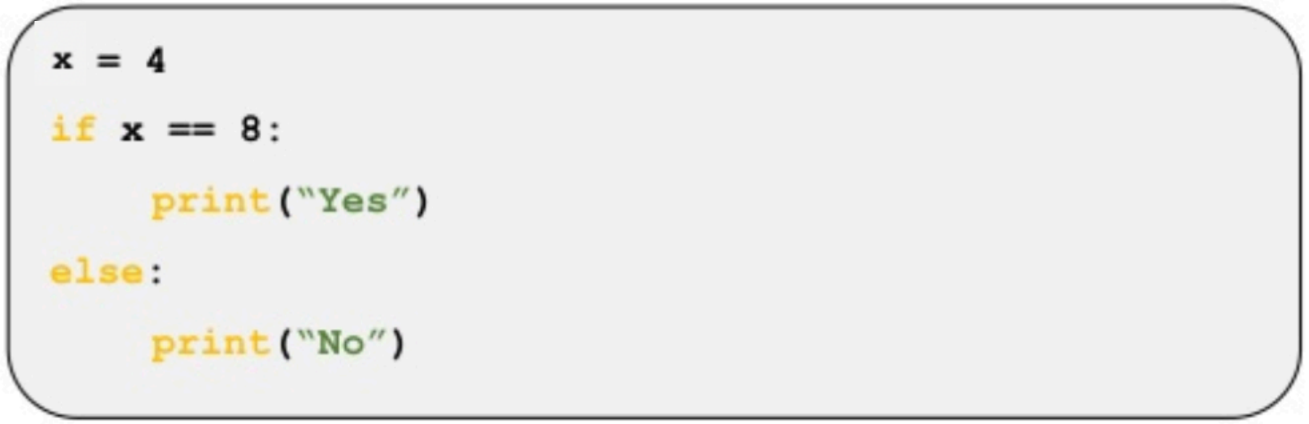
```
no = 24
if no > 18:
    print("Greater than 18")
    if no <= 50:
        print("Between 18 & 50")
```

Output:

```
Greater than 18
Between 18 & 50
```

Else Statements

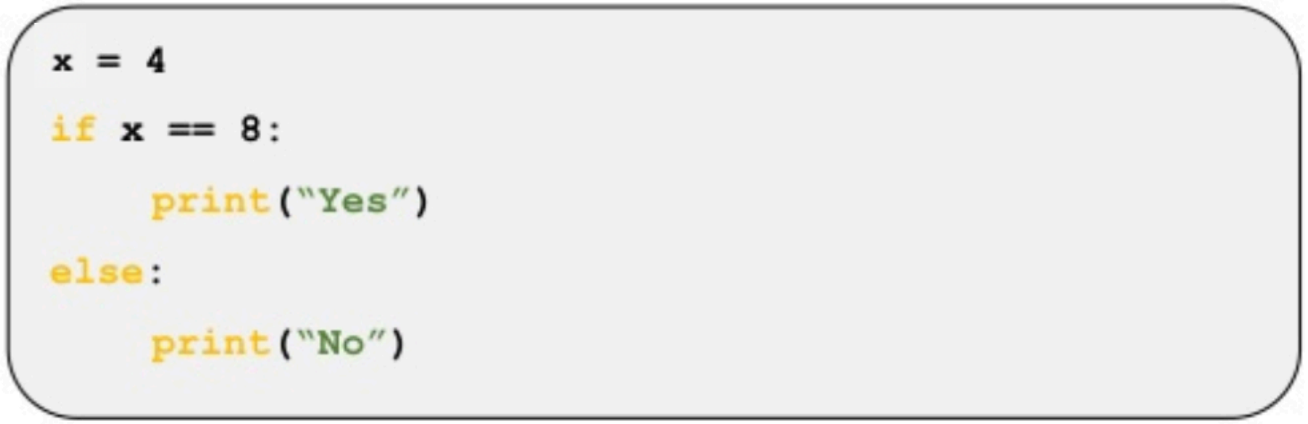
- An else statement follows an if statement
- The code in the else statement block gets executed if the if statements evaluates to False:



```
x = 4
if x == 8:
    print("Yes")
else:
    print("No")
```

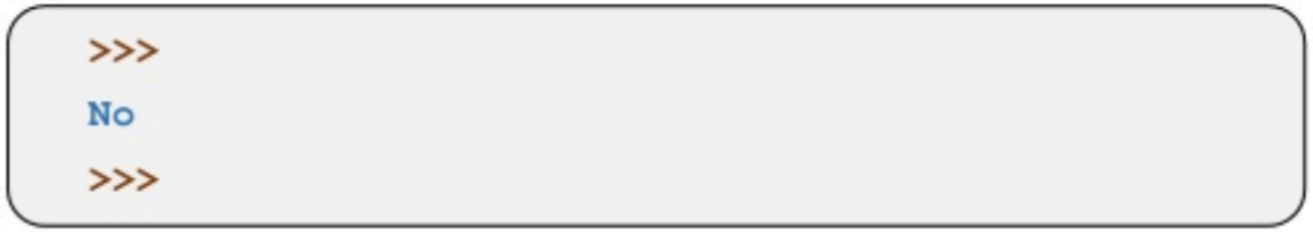
Else Statements

- An else statement follows an if statement
- The code in the else statement block gets executed if the if statements evaluates to False:



```
x = 4
if x == 8:
    print("Yes")
else:
    print("No")
```

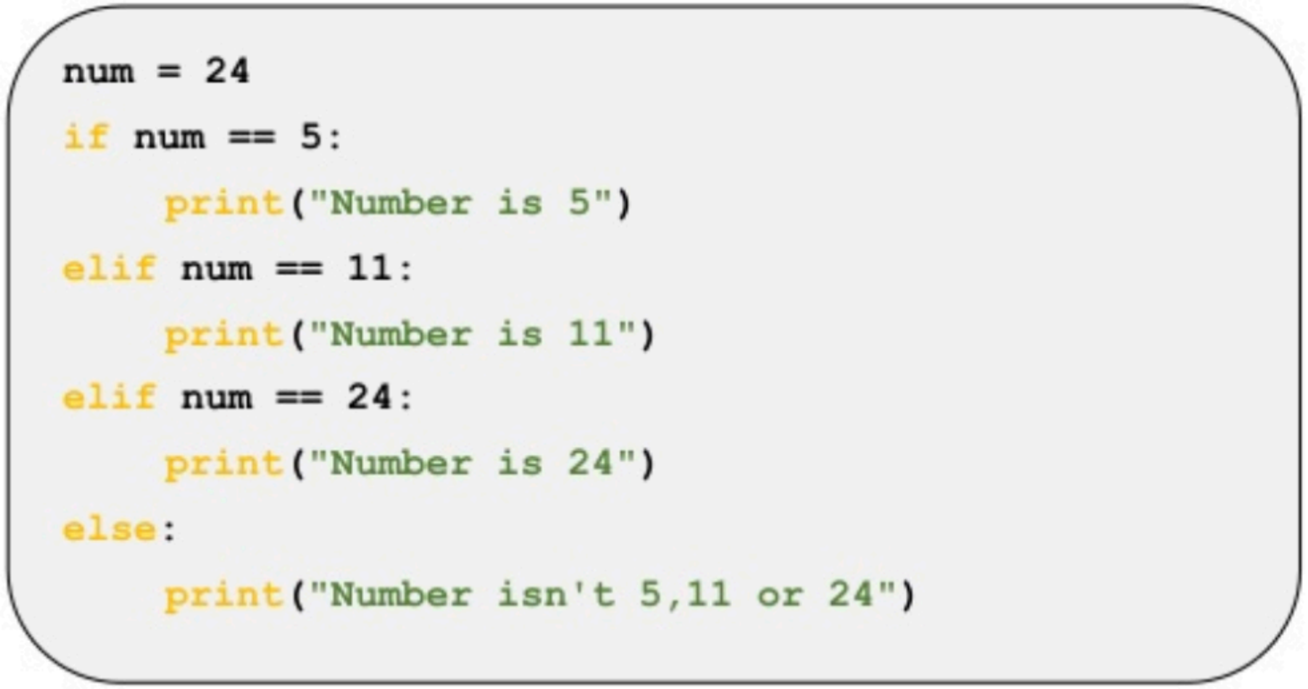
Output:



```
>>>
No
>>>
```

Elif Statements

- elif : stands for ‘else if’
- Used to create a chain of conditional checks



```
num = 24
if num == 5:
    print("Number is 5")
elif num == 11:
    print("Number is 11")
elif num == 24:
    print("Number is 24")
else:
    print("Number isn't 5,11 or 24")
```

while Loops

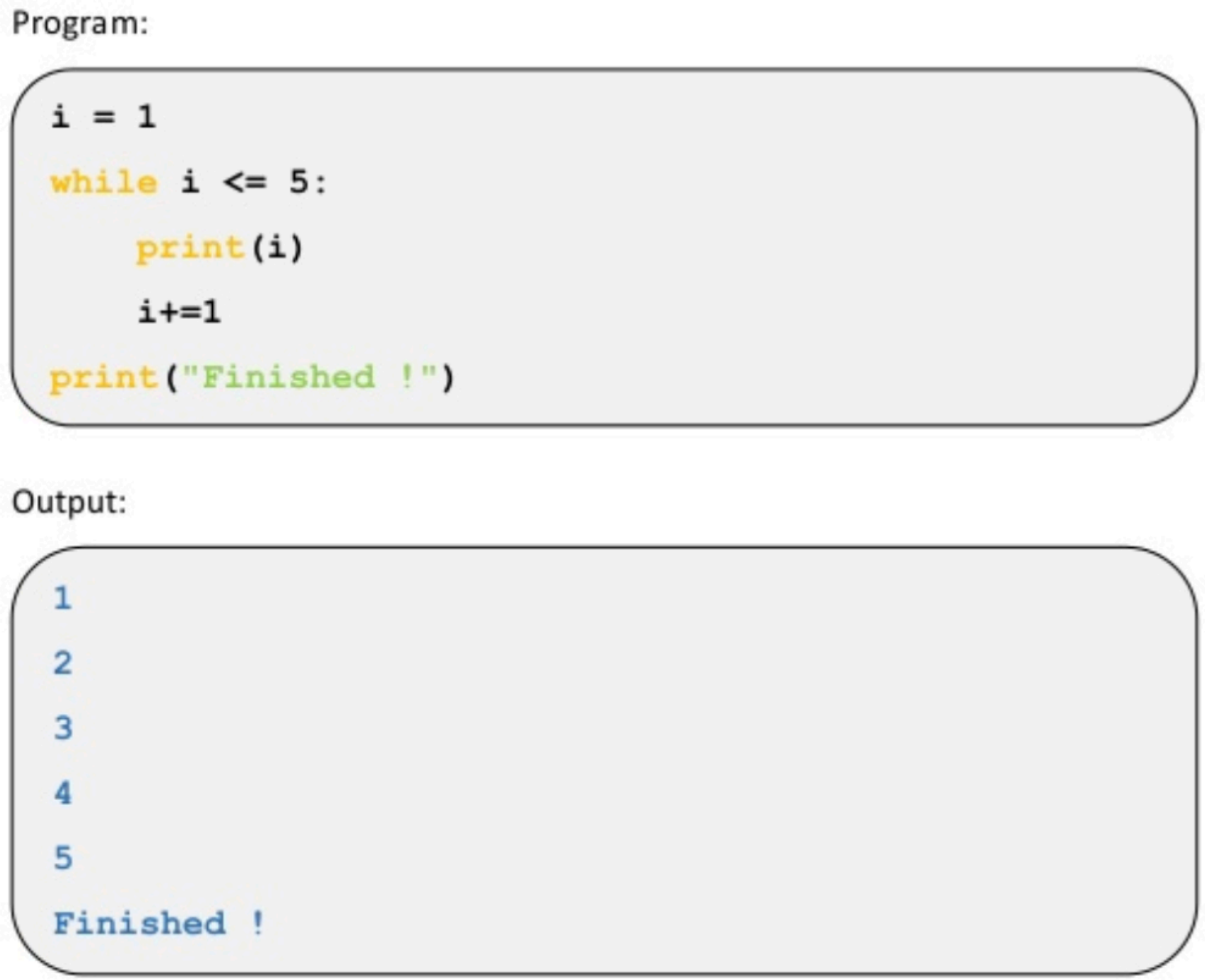
- An if statement is run ***once*** if it evaluates to true and is never run if it evaluates to false
- A while statement is similar, except it keeps running ***as long as*** the condition is true.
- Once it is evaluated to be False, the program moves on to the next block of code

while Loops

Program:

```
i = 1
while i <= 5:
    print(i)
    i+=1
print("Finished !")
```

while Loops



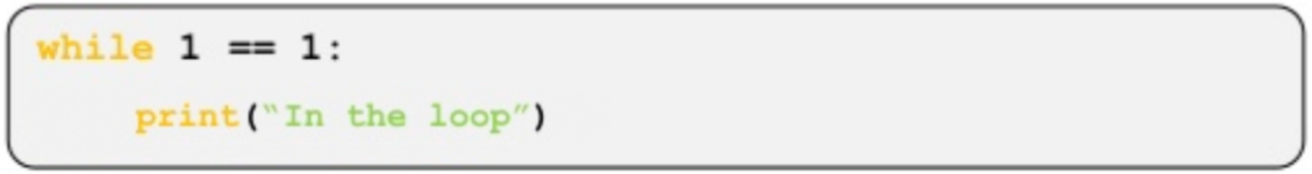
Program:

```
i = 1
while i <= 5:
    print(i)
    i+=1
print("Finished !")
```

Output:

```
1
2
3
4
5
Finished !
```

Infinite Loop



```
while 1 == 1:
    print("In the loop")
```

- Can stop the execution by pressing Control-C (or closing the window/program)

Control Structures

- Loop control statements: change execution from its normal sequence

- `Break`: exit out of a loop when a condition is triggered

```
for number in range(10):
    if number == 5:
        break
    print('Number is ' + str(number))
print('Out of loop')
```

Control Structures

- Loop control statements: change execution from its normal sequence

- `Break`: exit out of a loop when a condition is triggered

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
for number in range(10):
    if number == 5:
        break
    print('Number is ' + str(number))
print('Out of loop')
```

Control Structures

- Loop control statements: change execution from its normal sequence

- `break`: exit out of a loop when a condition is triggered

```
for number in range(10):
    if number == 5:
        break
    print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Out of loop
```

Control Structures

- Loop control statements: change execution from its normal sequence
- `continue`: skip over the part of the loop where a condition is triggered, but complete the rest of the loop

```
for number in range(10):
    if number == 5:
        continue
    print('Number is ' + str(number))
print('Out of loop')
```

Control Structures

- Loop control statements: change execution from its normal sequence
- `continue`: skip over the part of the loop where a condition is triggered, but complete the rest of the loop

```
for number in range(10):
    if number == 5:
        continue
    print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Number is 6
Number is 7
Number is 8
Number is 9
Out of loop
```

Control Structures

- Loop control statements: change execution from its normal sequence
- `pass`: handle the condition without the loop being impacted at all

```
for number in range(10):
    if number == 5:
        pass
    print('Number is ' + str(number))
print('Out of loop')
```

Control Structures

- Loop control statements: change execution from its normal sequence
- `pass`: handle the condition without the loop being impacted at all

```
for number in range(10):
    if number == 5:
        pass
    print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
Number is 6
Number is 7
Number is 8
Number is 9
Out of loop
```