Pyhon

Conditions & Recursion

Learning Objectives

- Introduce integer division
- Introduce Boolean Arithmetic
- Discuss conditionals (flow control)
- Explore recursive algorithms
- Begin accepting input from the keyboard

Integer Division

- **Recall:** on a PC, floating point math is susceptible to rounding errors
  - How does one model store 1/3?
- There are two division operators which work in the integer domain

Integer (fractional) representation: $\frac{17}{5} = 3\ \frac{2}{5}$

- How do we get to these numbers?
  - Floor Division (//)
    - Returns the integer portion of a division
    - 17 // 5 = 3
  - Modulo Division
    - Returns the remainder
    - 17 % 5 = 2
  - With the two operations, we can obtain a fractional view

Integer Division

- **Recall:** on a PC, floating point math is susceptible to rounding errors
  - How does one model store 1/3?
- There are two division operators which work in the integer domain

Integer (fractional) representation: $\frac{17}{5} = 3\frac{2}{5}$

- How do we get to these numbers?
  - Floor Division (//)
    - Returns the integer portion of a division
    - 17 // 5 = 3
  - Modulo Division
    - Returns the remainder
    - 17 % 5 = 2
  - With the two operations, we can obtain a fractional view

Applications of Integer Division

- How could we obtain just the 1’s place of a number? (ex. Given 153, we want the 3)

Applications of Integer Division

- How could we obtain just the 1’s place of a number? (ex. Given 153, we want the 3)
  - Use modulo division 153 % 10 = 3 (assuming base 10)

Applications of Integer Division

- How could we obtain just the 1’s place of a number? (ex. Given 153, we want the 3)
  - Use modulo division 153 % 10 = 3 (assuming base 10)
- How can we check to see if one number is evenly divisible by another? (ex. Is 57 evenly divisible by 3?)

Applications of Integer Division

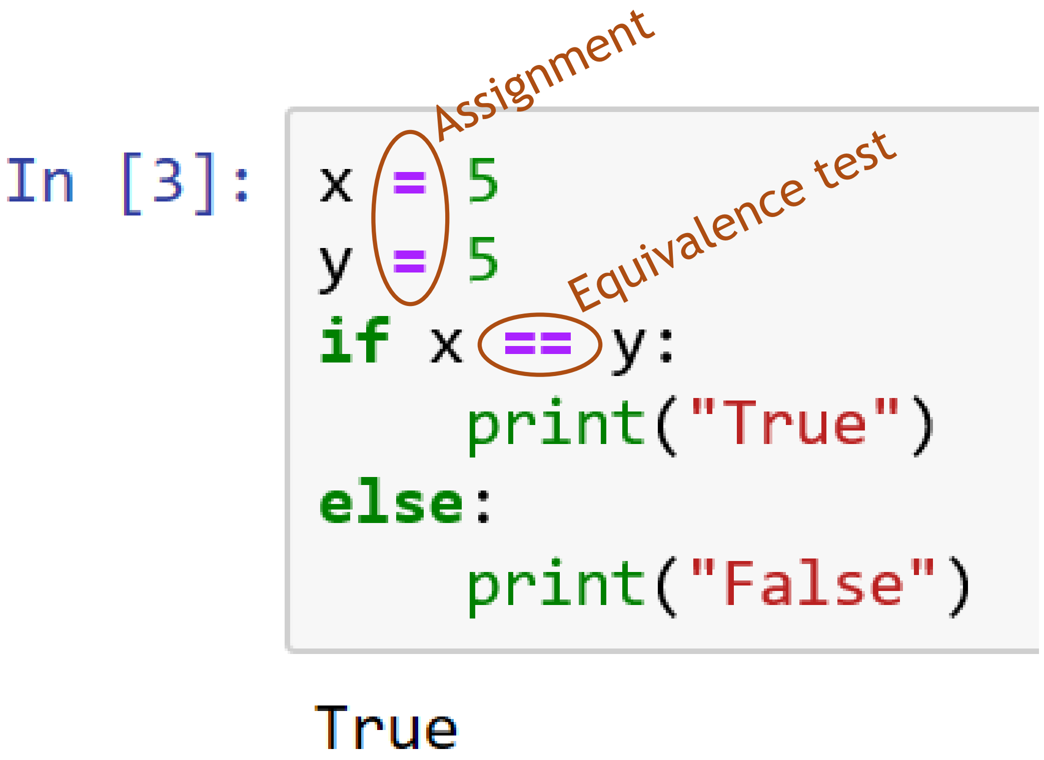
- How could we obtain just the 1’s place of a number? (ex. Given 153, we want the 3)
  - Use modulo division 153 % 10 = 3 (assuming base 10)
- How can we check to see if one number is evenly divisible by another? (ex. Is 57 evenly divisible by 3?)
  - Check to see if the remainder is zero (does 57 % 3 == 0 ?)
  - How can we use this to detect oddness or evenness?

Boolean Algebra

- Named after Mathematician George Boole who introduced these concepts to the world in “The Mathematical Analysis of Logic”
- Concepts are either logically true or logically false
  - Boolean Algebra supports two values, True and False
    - In most of computer science, including Python, 0 is considered false. Not zero is true.
- Uses relational and logical operators
- Note how this lines up to a machine which only works with 0s and 1s

Relational Operators

- **Recall:** Python uses the = operator to assign a value to a variable
- Equivalence == (two equal signs)
- Using a single = for an equivalence test will generate a syntax error



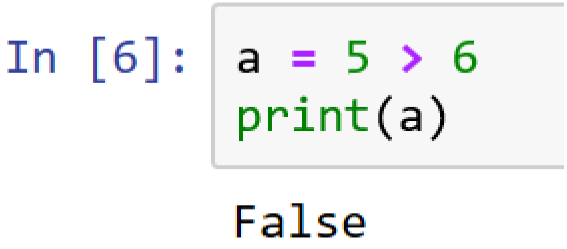
```
In [3]: x = 5
        y = 5
        if x == y:
            print("True")
        else:
            print("False")

        True
```

Relational Operators

Operation	Meaning
X == Y	True when X is equal to Y
X != Y	True when X is not equal to Y
X > Y	True when X is greater than Y
X < Y	True when X is less than Y
X >= Y	True when X is greater than or equal to Y (logically inverted from X < Y)
X <= Y	True when X is less than or equal to Y (logically inverted from X > Y)

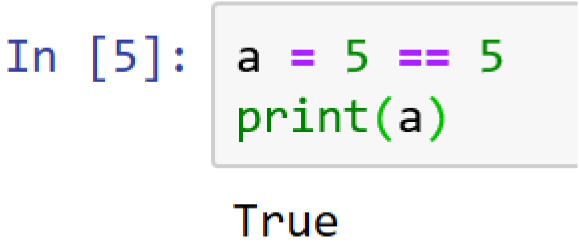
Each of these operators return a value of type Boolean. That value can be stored.



In [6]:

```
a = 5 > 6
print(a)
```

```
False
```



In [5]:

```
a = 5 == 5
print(a)
```

```
True
```

Logical Operators

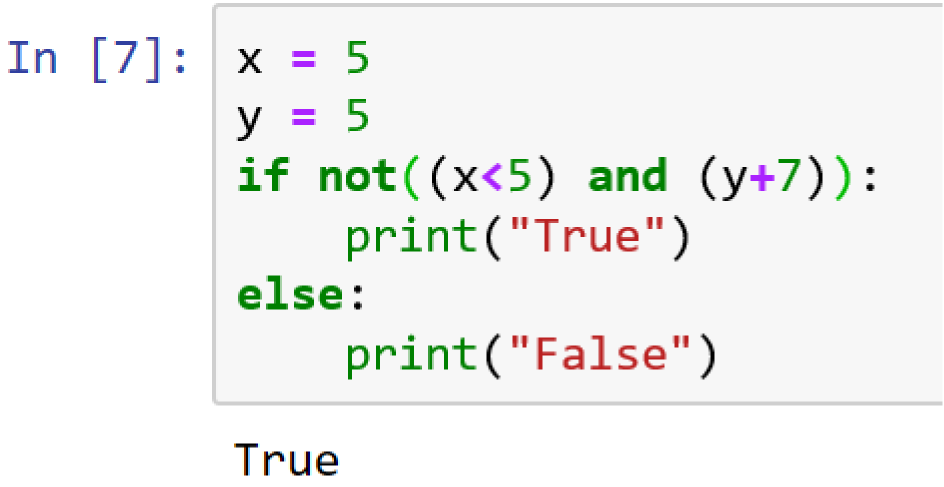
- Operators to link Boolean expressions together to create more complex semantics

Operator	Function
X and Y	True if and only if X is true and Y is true
X or Y	True if X is true or Y is true
not X	True if X is false, false if X is true

Logical Operators

- Operators to link Boolean expressions together to create more complex semantics

Operator	Function
X and Y	True if and only if X is true and Y is true
X or Y	True if X is true or Y is true
not X	True if X is false, false if X is true



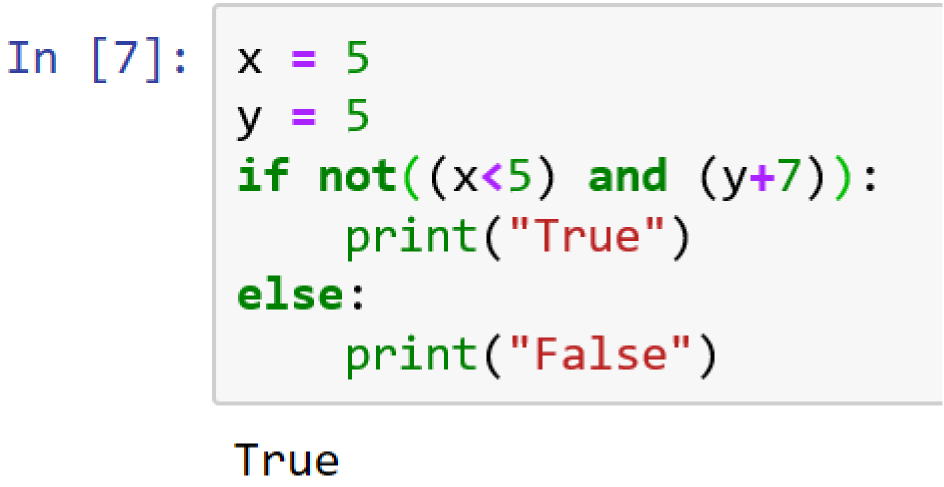
```
In [7]: x = 5
        y = 5
        if not((x<5) and (y+7)):
            print("True")
        else:
            print("False")
```

```
True
```

Logical Operators

- Operators to link Boolean expressions together to create more complex logical expressions

Operator	Function
X and Y	True if and only if X is true and Y is true
X or Y	True if X is true or Y is true
not X	True if X is false, false if X is true



```
In [7]: x = 5
        y = 5
        if not((x<5) and (y+7)):
            print("True")
        else:
            print("False")

        True
```

Note the use of parenthesis and the mixing of various operators

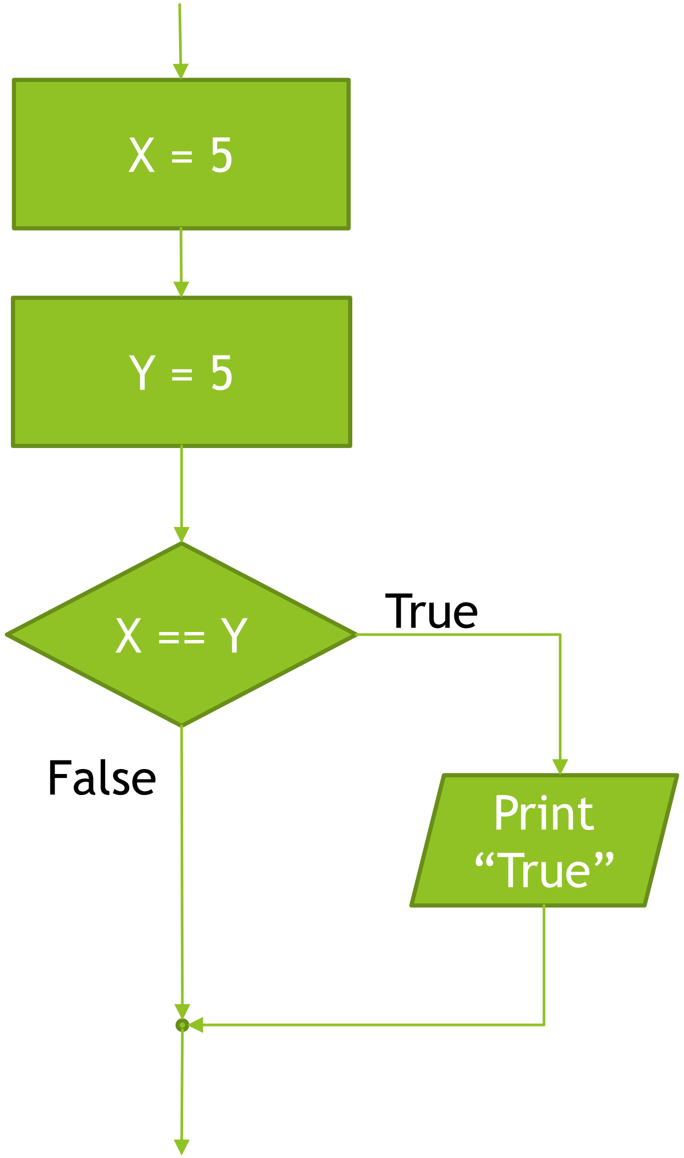
Order of Operations (Precedence)

Operator
()
**
+ - (unary)
* / % //
+ - (binary)
< <= > >=
== !=
= %= /= //= -= += *= **=
not
and
or

As in traditional Algebra, operators at the same level of precedence are evaluated from left to right

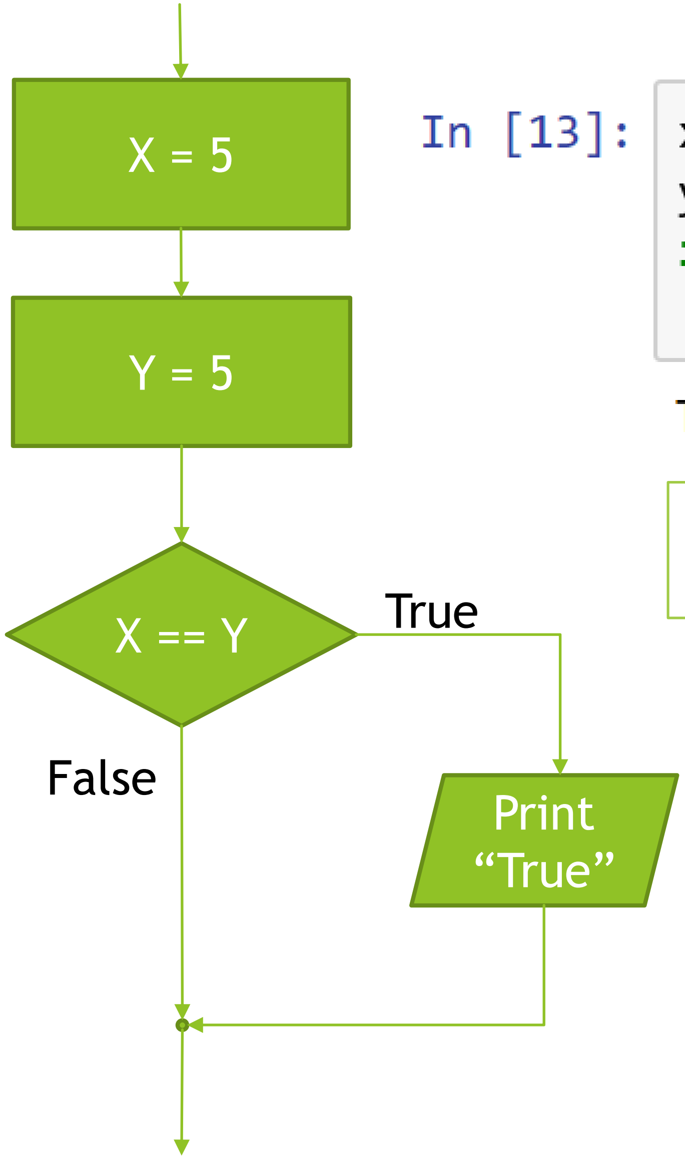
Conditional Execution

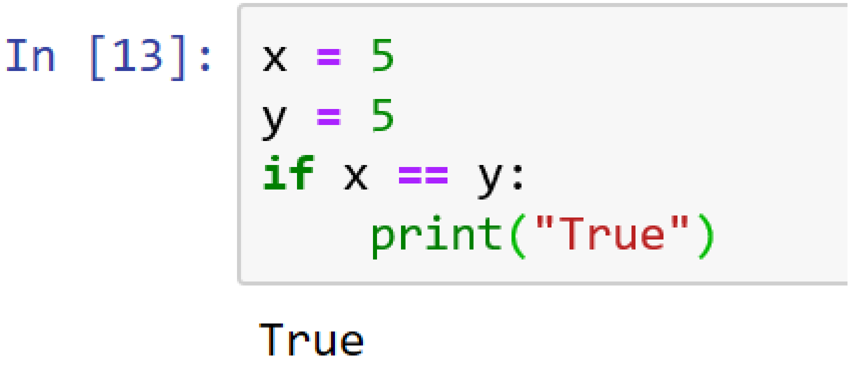
- Sometimes called alternation or decisions
- Run a set of statements, if some condition is true



Conditional Execution

- Sometimes called alternation or decisions
- Run a set of statements, if some condition is true
- Uses the if keyword
- All indented commands run
  - There isn't a reasonable limit to the number of commands
- The logic controlling whether the instructions run is called the boundary condition





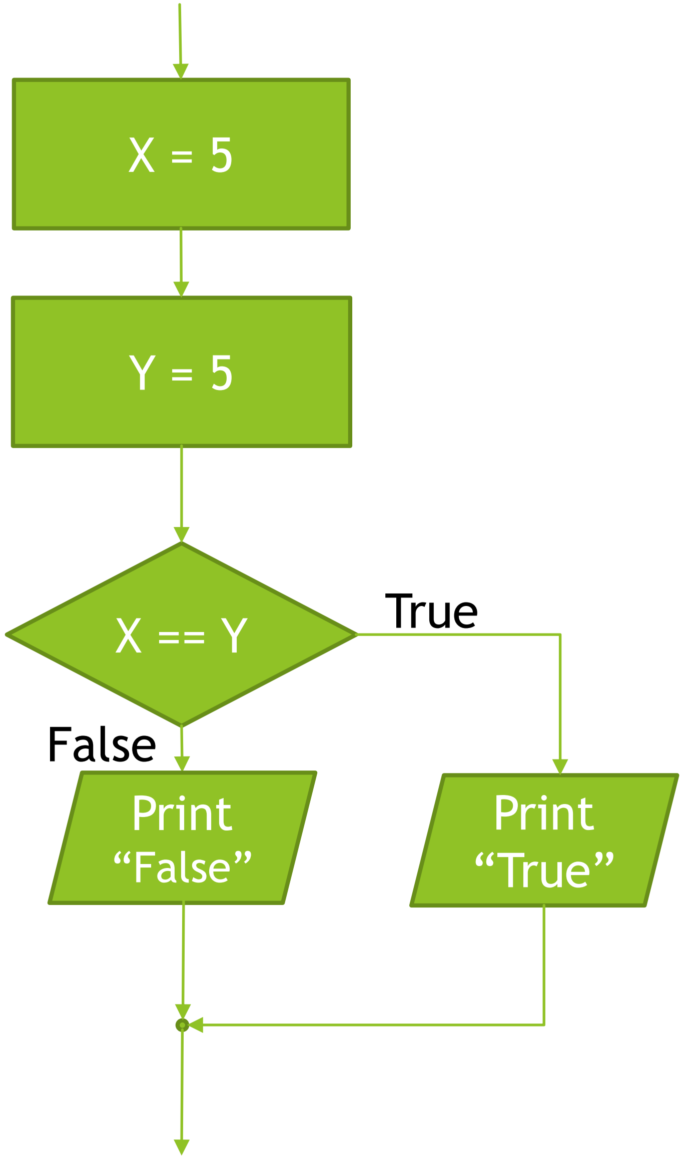
```
In [13]: x = 5
         y = 5
         if x == y:
             print("True")
```

```
True
```

`(x == y)` *is the boundary condition in this example*

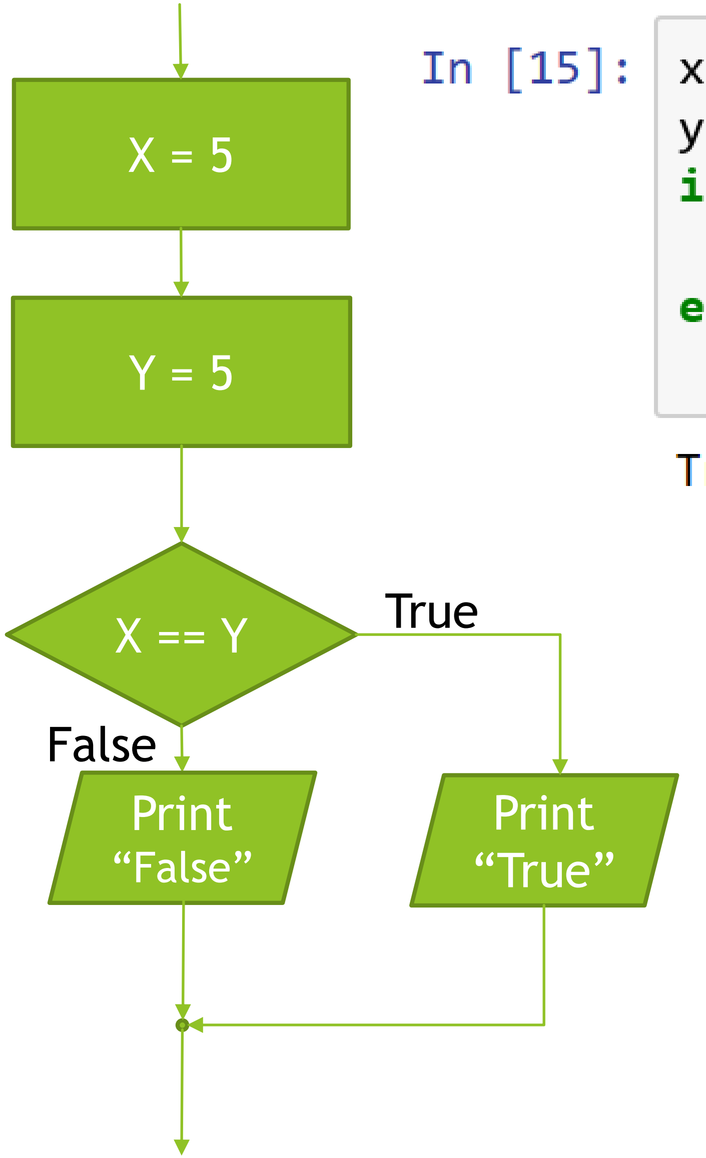
Alternative Execution

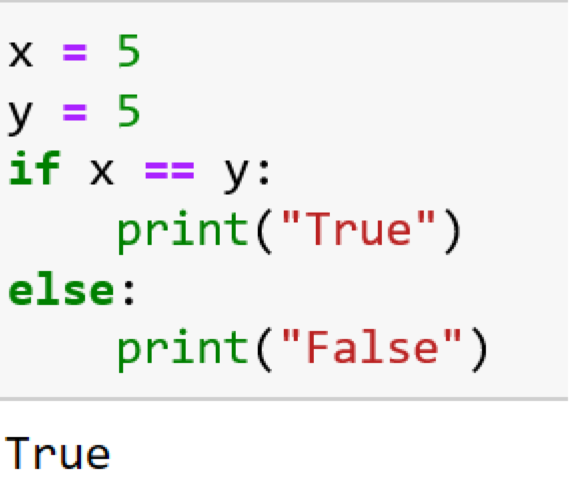
- If a condition is true, run a set of commands, if it is false run a different set of commands
- Uses the if and else keywords



Alternative Execution

- If a condition is true, run a set of commands, if it is false run a different set of commands
- Uses the if and else keywords





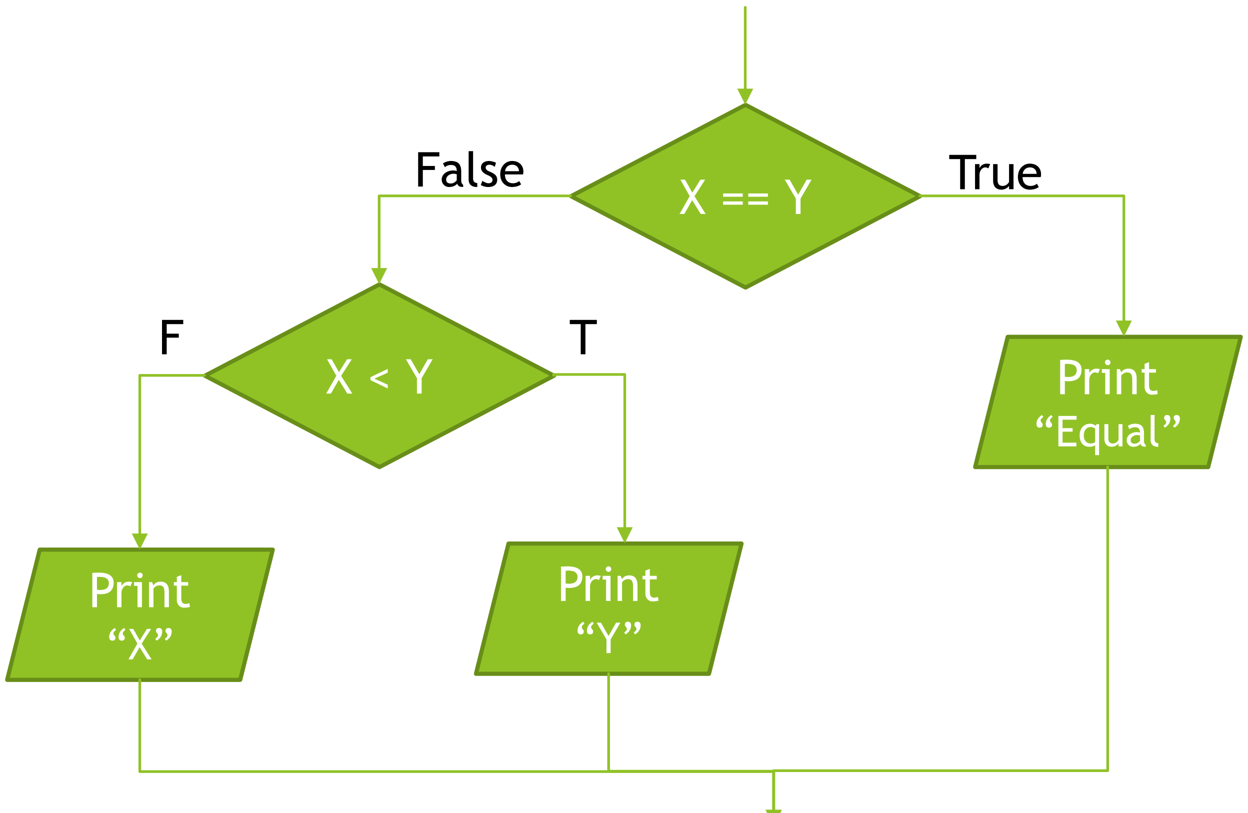
In [15]:

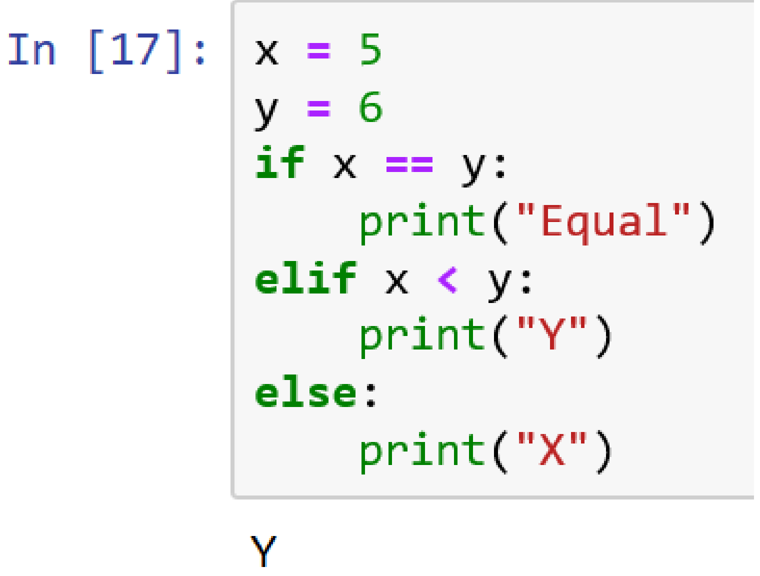
```
x = 5
y = 5
if x == y:
    print("True")
else:
    print("False")
```

```
True
```

Chained Conditionals

- Decisions can be linked in a chain allowing for a choice of many alternatives
- Uses the if, elif and else keywords
- Note: the order matters. The flow is directed down the first true conditional



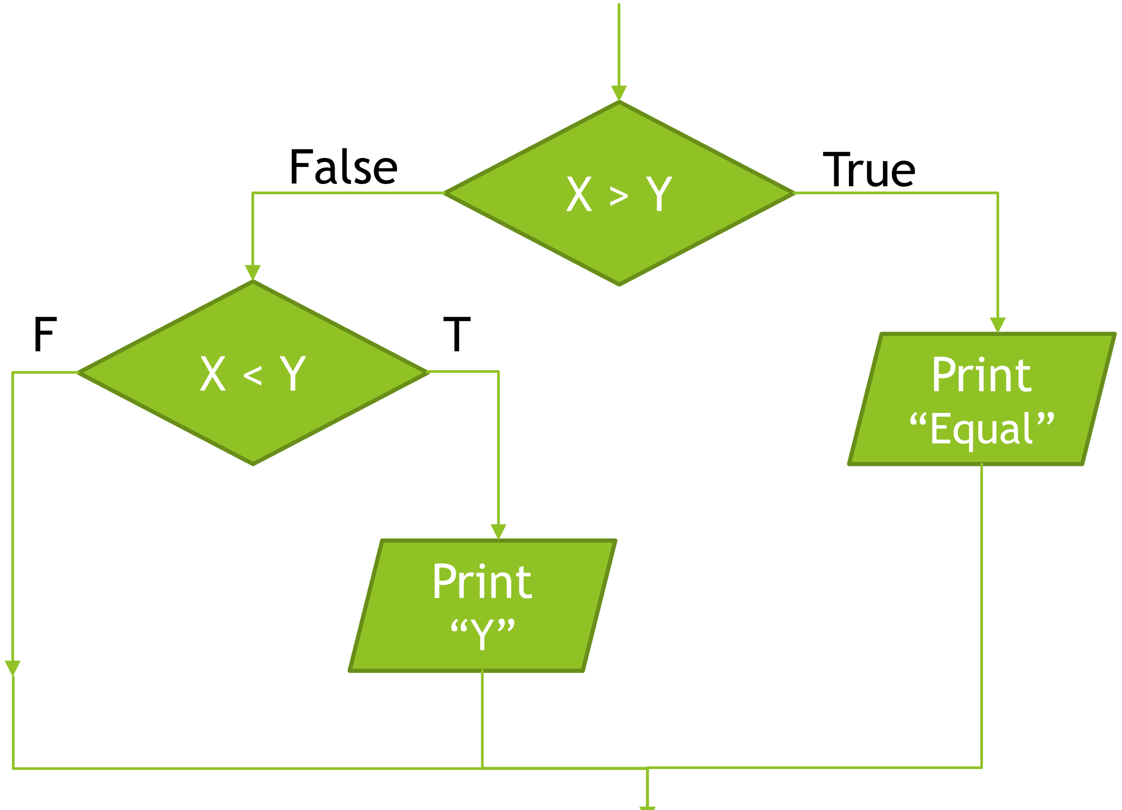


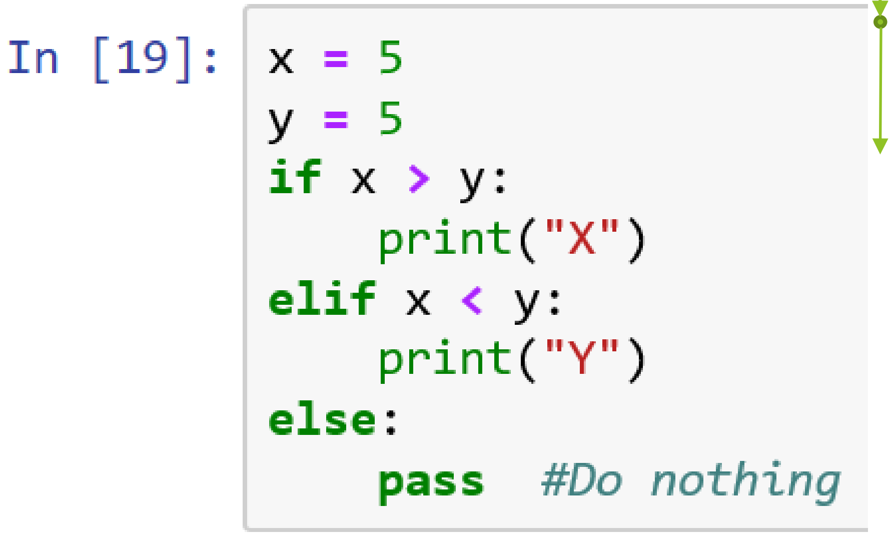
```
In [17]: x = 5
         y = 6
         if x == y:
             print("Equal")
         elif x < y:
             print("Y")
         else:
             print("X")

Y
```

Chained Conditionals

- When testing code, ensure your test cases go down every branch in every conditional
- If there is an elif, there should always be an else
  - We want the decision tree to be complete
  - It is ok if the else case does nothing. To indicate that it does nothing, use the pass keyword

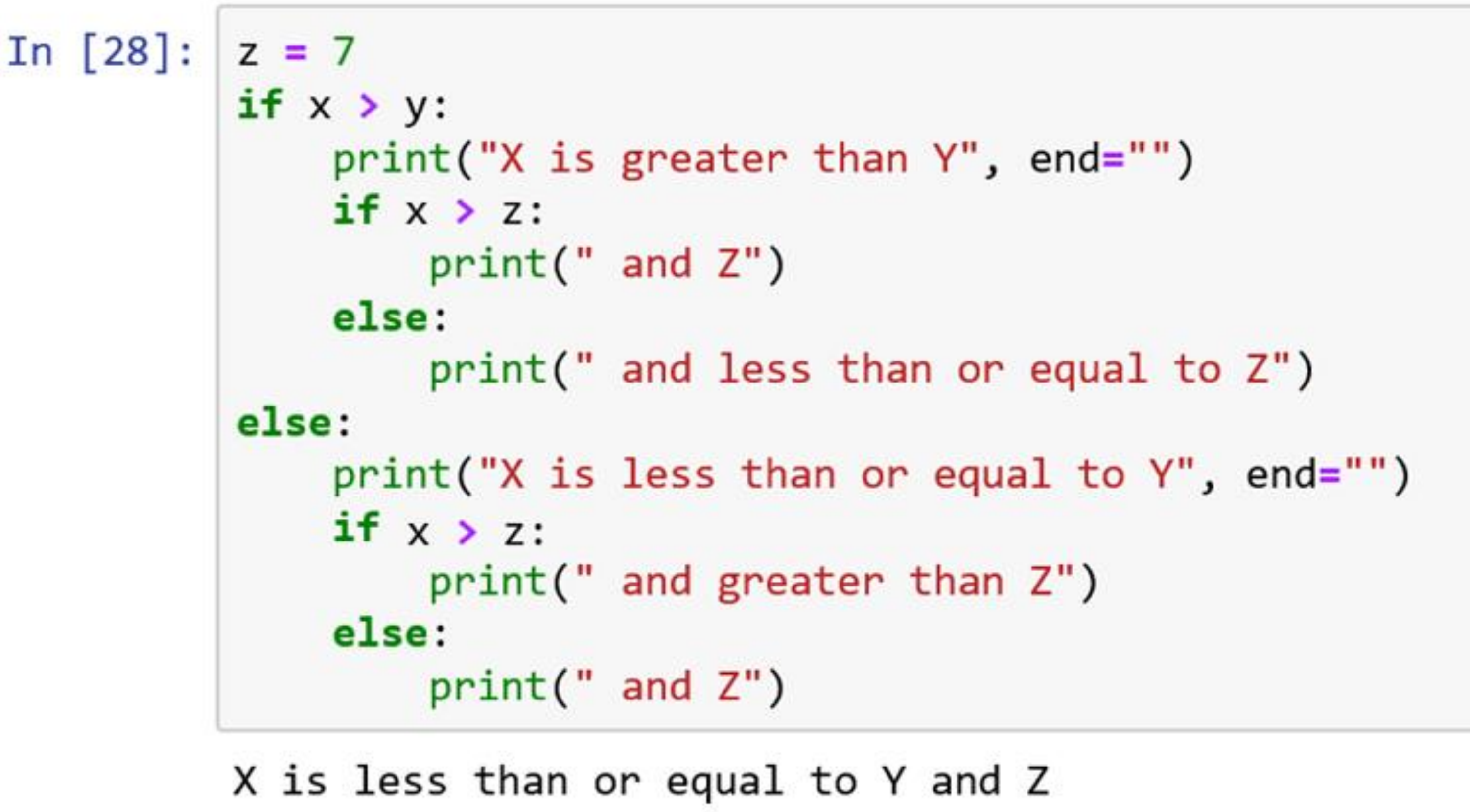




```
In [19]: x = 5
         y = 5
         if x > y:
             print("X")
         elif x < y:
             print("Y")
         else:
             pass  #Do nothing
```

Nesting Logic

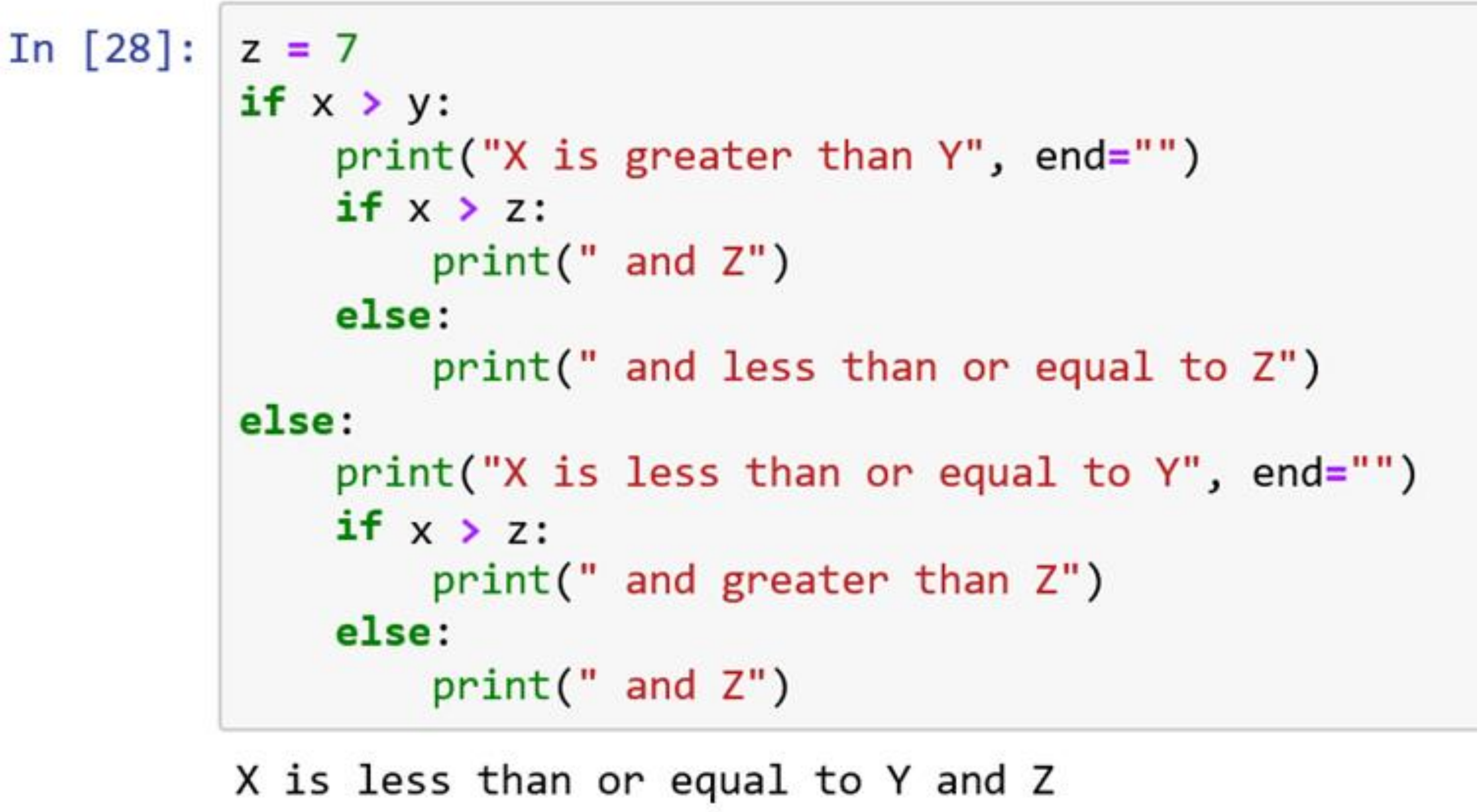
- All conditional operations can be nested
  - Nested is when there is a conditional within the program instructions gated by some other condition
  - Note the indentation. In python that indentation is required and is considered part of the grammar of the language
  - Note the "or equal" clauses in the print statements
  - Generally, if your logic is nested more than a few layers deep, see if there is a more obvious way to write the code



```
z = 7
if x > y:
    print("X is greater than Y", end="")
    if x > z:
        print(" and Z")
    else:
        print(" and less than or equal to Z")
else:
    print("X is less than or equal to Y", end="")
    if x > z:
        print(" and greater than Z")
    else:
        print(" and Z")
```

```
X is less than or equal to Y and Z
```

# end=“” *suppresses the newline/carriage return*



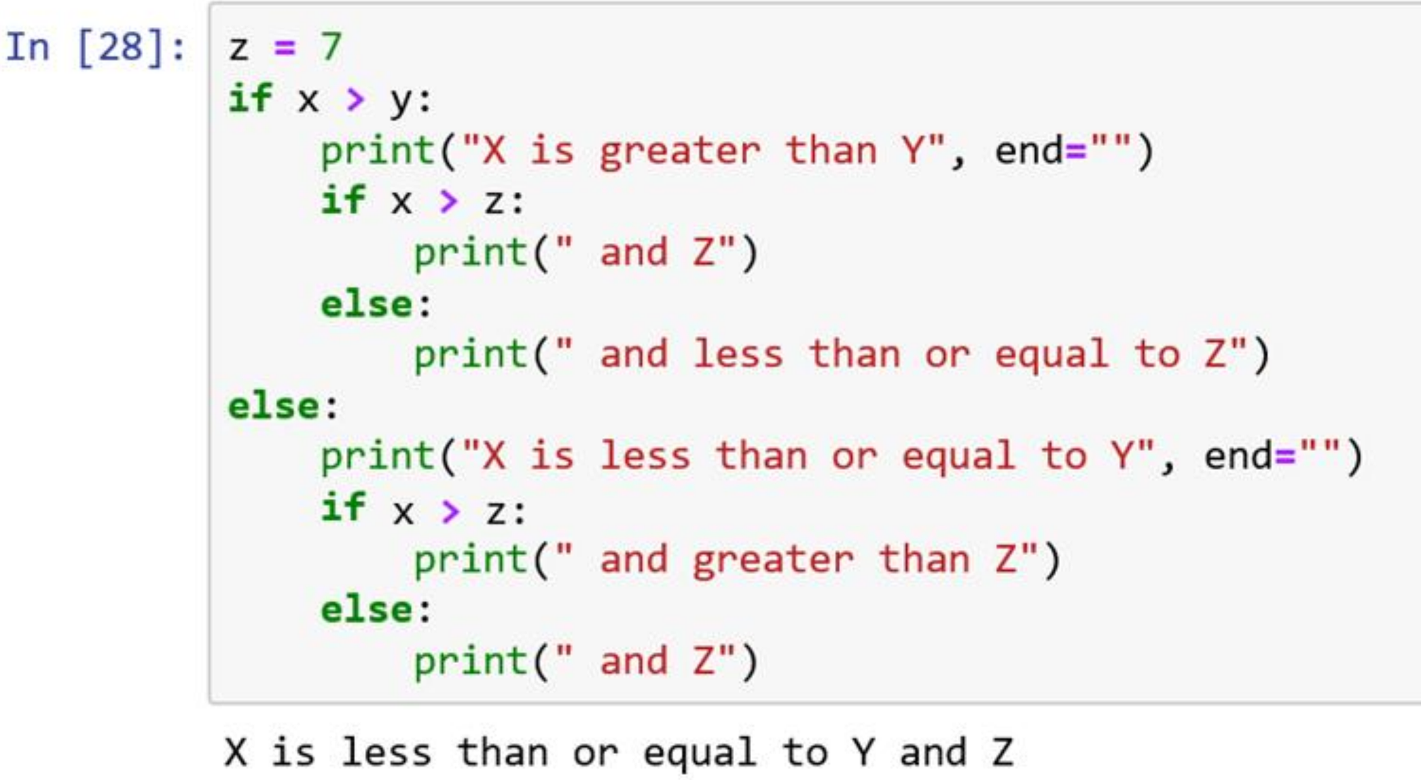
In [28]:

```
z = 7
if x > y:
    print("X is greater than Y", end="")
    if x > z:
        print(" and Z")
    else:
        print(" and less than or equal to Z")
else:
    print("X is less than or equal to Y", end="")
    if x > z:
        print(" and greater than Z")
    else:
        print(" and Z")
```

```
X is less than or equal to Y and Z
```

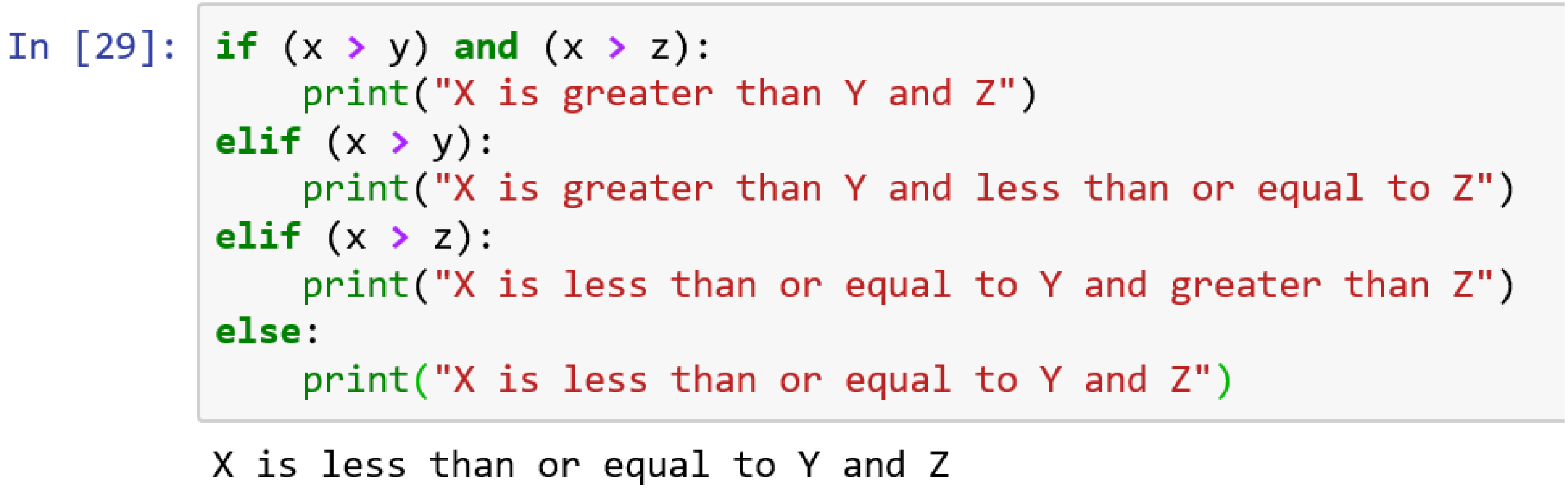
Code Refactoring

The process of restructuring existing code without changing its external meaning



```
In [28]: z = 7
         if x > y:
             print("X is greater than Y", end="")
             if x > z:
                 print(" and Z")
             else:
                 print(" and less than or equal to Z")
         else:
             print("X is less than or equal to Y", end="")
             if x > z:
                 print(" and greater than Z")
             else:
                 print(" and Z")

         X is less than or equal to Y and Z
```

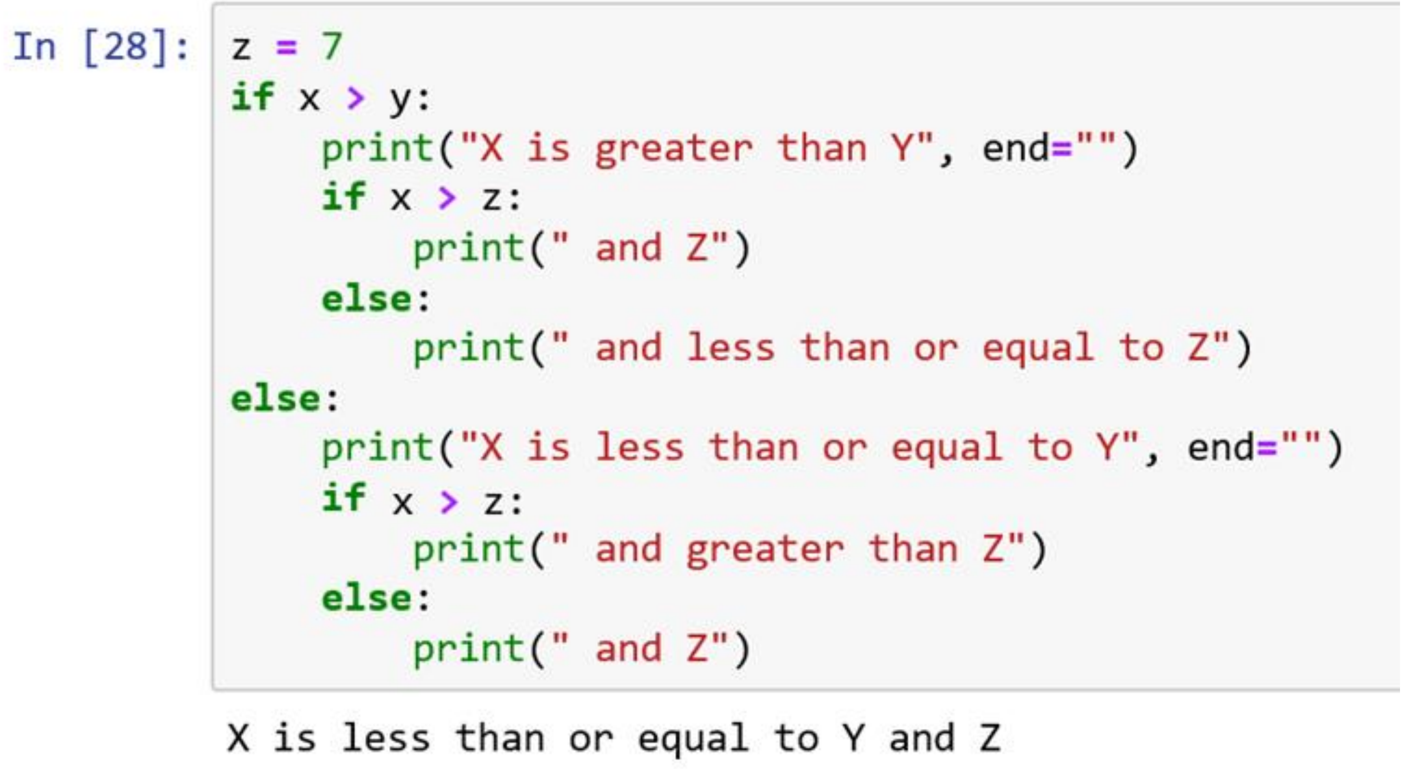


```
In [29]: if (x > y) and (x > z):
             print("X is greater than Y and Z")
         elif (x > y):
             print("X is greater than Y and less than or equal to Z")
         elif (x > z):
             print("X is less than or equal to Y and greater than Z")
         else:
             print("X is less than or equal to Y and Z")

         X is less than or equal to Y and Z
```

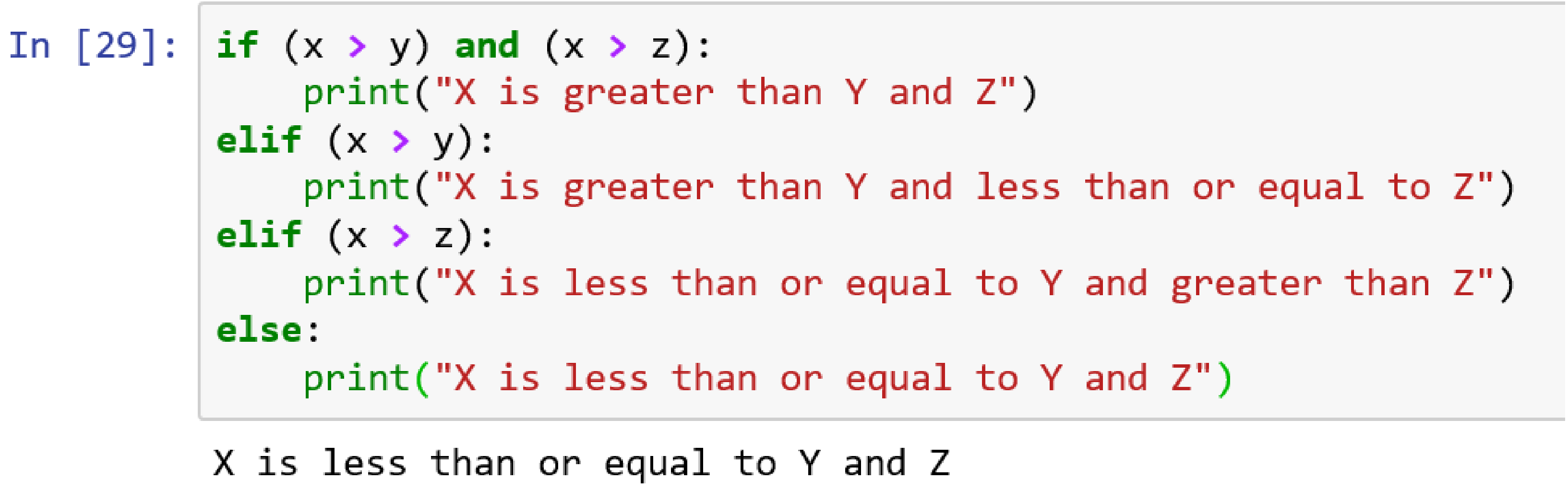
Code Refactoring

The process of restructuring existing code without changing its external meaning



```
In [28]: z = 7
         if x > y:
             print("X is greater than Y", end="")
             if x > z:
                 print(" and Z")
             else:
                 print(" and less than or equal to Z")
         else:
             print("X is less than or equal to Y", end="")
             if x > z:
                 print(" and greater than Z")
             else:
                 print(" and Z")

         X is less than or equal to Y and Z
```



```
In [29]: if (x > y) and (x > z):
             print("X is greater than Y and Z")
         elif (x > y):
             print("X is greater than Y and less than or equal to Z")
         elif (x > z):
             print("X is less than or equal to Y and greater than Z")
         else:
             print("X is less than or equal to Y and Z")

         X is less than or equal to Y and Z
```

Which version is better?

Resources

- Bryan Burlingame's course notes
- Downey, A. (2016) *Think Python, Second Edition* Sebastopol, CA: O'Reilly Media
- (n.d.). 3.7.0 Documentation. *6. Expressions — Python 3.7.0 documentation*. Retrieved September 11, 2018, from http://docs.python.org/3.7/reference/expressions.html