Comparison Operators

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

Numpy recap

```
# Code from Intro to Python for Data Science, Chapter 4
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
bmi = np_weight / np_height ** 2
bmi
```

```
array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
bmi > 23
```

```
array([False, False, False,  True, False], dtype=bool)
```

```
bmi[bmi > 23]
```

```
array([ 24.747])
```

- Comparison operators: how Python values relate

Numeric comparisons

```
2 < 3
```

```
True
```

```
2 == 3
```

```
False
```

```
2 <= 3
```

```
True
```

```
3 <= 3
```

```
True
```

```
x = 2
y = 3
x < y
```

```
True
```

Other comparisons

```
"carl" < "chris"
```

```
True
```

```
3 < "chris"
```

```
TypeError: unorderable types: int() < str()
```

```
3 < 4.1
```

```
True
```

Other comparisons

```
bmi
```

```
array([21.852, 20.975, 21.75 , 24.747, 21.441])
```

```
bmi > 23
```

```
array([False, False, False, True, False], dtype=bool)
```

Comparators

Comparator	Meaning
`<`	Strictly less than
`<=`	Less than or equal
`>`	Strictly greater than
`>=`	Greater than or equal
`==`	Equal
`!=`	Not equal

Let's practice!

INTERMEDIATE PYTHON

Boolean Operators

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

Boolean Operators

- `and`
- `or`
- `not`

and

```
True and True
```

```
True
```

```
x = 12
x > 5 and x < 15
# True     True
```

```
True
```

```
False and True
```

```
False
```

```
True and False
```

```
False
```

```
False and False
```

```
False
```

or

```
True or True
```

```
True
```

```
False or True
```

```
True
```

```
True or False
```

```
True
```

```
False or False
```

```
False
```

```
y = 5
y < 7 or y > 13
```

```
True
```

not

```
not True
```

```
False
```

```
not False
```

```
True
```

NumPy

```
bmi     # calculation of bmi left out
```

```
array([21.852, 20.975, 21.75 , 24.747, 21.441])
```

```
bmi > 21
```

```
array([True, False, True, True, True], dtype=bool)
```

```
bmi < 22
```

```
array([True, True, True, False, True], dtype=bool)
```

```
bmi > 21 and bmi < 22
```

```
ValueError: The truth value of an array with more than one element is
ambiguous. Use a.any() or a.all()
```

Let's now try to combine those with the and operator I just introduced. Oops, an error. The truth value of an array with more than one element is ambiguous. and clearly doesn't like an array of booleans to work on.

NumPy

- `logical_and()`
- `logical_or()`
- `logical_not()`

7. NumPy
After some digging in the numpy documentation, you can find the functions logical_and, logical_or and logical_not, the "array equivalents" of and or and not. To find out which bmis are between 21 and 22, we thus need this call. Again, as we expect from Numpy, the and operation is performed element-wise: True and True give True, like these ones, but False and True or True and False give False, like for these elements. To actually select only these bmis from the bmi array, we can use the resulting array of booleans in square brackets.

```
np.logical_and(bmi > 21, bmi < 22)
```

```
array([True, False, True, False, True], dtype=bool)
```

```
bmi[np.logical_and(bmi > 21, bmi < 22)]
```

```
array([21.852, 21.75, 21.441])
```

Let's practice!

INTERMEDIATE PYTHON

if, elif, else

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

Overview

- Comparison Operators
  - `<`, `>`, `>=`, `<=`, `==`, `!=`
- Boolean Operators
  - `and`, `or`, `not`
- Conditional Statements
  - `if`, `else`, `elif`

Things get really interesting when you can actually use these concepts to change how your program behaves. Depending on the outcome of your comparisons, you might want your Python code to behave differently. You can do this with conditional statements in Python: if, else and elif.

if

```
if condition :
    expression
```

control.py

```
z = 4
if z % 2 == 0 :     # True
    print("z is even")
```

3. if
Let's start working in a script, control.py. Suppose you have a variable z, equal to 4. If the value is even, you want to print out: "z is even". This code does the trick. modulo operator 2 will return 0 if z is even. If you run this, Python checks if the condition holds. It's true, so the corresponding code is executed: "z is even" gets printed out. Let's compare this to the general recipe for an if statement. It reads as follows: if condition, execute expression. Notice the colon at the end, and the fact that you simply have to indent the Python code with four spaces (or a tab) to tell Python what to do in the case the condition succeeds.

```
z is even
```

if

```
if condition :
    expression
```

- `expression` not part of if

`control.py`

```
z = 4
if z % 2 == 0 :    # True
    print("z is even")
```

```
z is even
```

if

```
if condition :
    expression
```

`control.py`

```
z = 4
if z % 2 == 0 :
    print("checking "  + str(z))
    print("z is even")
```

```
checking 4
z is even
```

if

```
if condition :
    expression
```

`control.py`

```
z = 5
if z % 2 == 0 :     # False
    print("checking "  + str(z))
    print("z is even")
```

else

```
if condition :
    expression
else :
    expression
```

`control.py`

```
z = 5
if z % 2 == 0 :    # False
    print("z is even")
else :
    print("z is odd")
```

```
z is odd
```

elif

```
if condition :
    expression
elif condition :
    expression
else :
    expression
```

`control.py`

```
z = 3
if z % 2 == 0 :
    print("z is divisible by 2")    # False
elif z % 3 == 0 :
    print("z is divisible by 3")    # True
else :
    print("z is neither divisible by 2 nor by 3")
```

```
z is divisible by 3
```

elif

```
if condition :
    expression
elif condition :
    expression
else :
    expression
```

`control.py`

```
z = 6
if z % 2 == 0 :
    print("z is divisible by 2")    # True
elif z % 3 == 0 :
    print("z is divisible by 3")    # Never reached
else :
    print("z is neither divisible by 2 nor by 3")
```

```
z is divisible by 2
```

Let's practice!

INTERMEDIATE PYTHON

Filtering pandas DataFrames

INTERMEDIATE PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

datacamp

brics

```
import pandas as pd
brics = pd.read_csv("path/to/brics.csv", index_col = 0)
brics
```

```
         country    capital    area  population
BR        Brazil   Brasilia   8.516      200.40
RU        Russia     Moscow  17.100      143.50
IN         India  New Delhi   3.286     1252.00
CH         China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.98
```

Goal

```
         country    capital    area  population
BR        Brazil   Brasilia   8.516      200.40
RU        Russia     Moscow  17.100      143.50
IN         India  New Delhi   3.286     1252.00
CH         China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.98
```

- Select countries with area over 8 million km2
- 3 steps
  - Select the area column
  - Do comparison on area column
  - Use result to select countries

Step 1: Get column

```
         country    capital    area  population
BR        Brazil   Brasilia   8.516      200.40
RU        Russia     Moscow  17.100      143.50
IN         India  New Delhi   3.286     1252.00
CH         China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.98
```

```
brics["area"]
```

```
BR     8.516
RU    17.100
IN     3.286
CH     9.597
SA     1.221
Name: area, dtype: float64    # - Need Pandas Series
```

- Alternatives:

```
brics.loc[:,"area"]
brics.iloc[:,2]
```

4. Step 1: Get column
So the first step, getting the area column from brics. There are many different ways to do this. What's important here, is that we ideally get a Pandas Series, not a Pandas DataFrame. Let's do this with square brackets, like this. This loc alternative, and this iloc version, would also work perfectly fine.

Step 2: Compare

```
brics["area"]
```

```
BR     8.516
RU    17.100
IN     3.286
CH     9.597
SA     1.221
Name: area, dtype: float64
```

```
brics["area"] > 8
```

```
BR     True
RU     True
IN    False
CH     True
SA    False
Name: area, dtype: bool
```

```
is_huge = brics["area"] > 8
```

5. Step 2: Compare
Next, we actually perform the comparison. To see which rows have an area greater than 8, we simply append greater than 8 to the code from before, like this. Now we get a Series containing booleans. If you compare it to the actual area values, you can see that the areas with a value over 8 correspond to True, and the ones with a value under 8 correspond to False now. Let me store this Boolean Series as is_huge.

Step 3: Subset DF

```
is_huge
```

```
BR     True
RU     True
IN    False
CH     True
SA    False
Name: area, dtype: bool
```

```
brics[is_huge]
```

```
   country   capital    area  population
BR  Brazil  Brasilia   8.516       200.4
RU  Russia    Moscow  17.100       143.5
CH   China   Beijing   9.597      1357.0
```

6. Step 3: Subset DF
The final step is using this boolean Series to subset the Pandas DataFrame. This is something I haven't shown you yet. To do this, you put is_huge inside square brackets. The result is exactly what we want: only the countries with an area greater than 8, namely Brazil, Russia and China.

Summary

```
         country    capital    area  population
BR        Brazil   Brasilia   8.516      200.40
RU        Russia     Moscow  17.100      143.50
IN         India  New Delhi   3.286     1252.00
CH         China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.988
```

```
is_huge = brics["area"] > 8
brics[is_huge]
```

```
   country   capital    area  population
BR  Brazil  Brasilia   8.516       200.4
RU  Russia    Moscow  17.100       143.5
CH   China   Beijing   9.597      1357.0
```

```
brics[brics["area"] > 8]
```

```
   country   capital    area  population
BR  Brazil  Brasilia   8.516       200.4
RU  Russia    Moscow  17.100       143.5
CH   China   Beijing   9.597      1357.0
```

7. Summary
So let's summarize this: I selected the area column, performed a comparison on this column and the stored it as is_huge so that I can use it to index the brics dataframe. These different commands do the trick. However, we can also write this in a one-liner: simply put the code that defines is_huge directly in the square brackets. Great!

Boolean operators

8. Boolean operators
Now we haven't used boolean operators yet. Remember that we used this logical_and function from the Numpy package to do an element wise boolean operation on Numpy arrays? Because Pandas is built on Numpy, you can also use that function here. Suppose you only want to keep the observations that have an area between 8 and 10 million square kilometers. After importing numpy as np, we can use the logical_and() function to create a Boolean Series. The only thing left to do is placing this code inside square brackets to subset brics appropriately. This time, only Brazil and China are included. Russia has an area of 17 million square kilometers, which doesn't meet the conditions. I hope these examples have shown you how easy it is to filter dataframes to get interesting results.

```
country    capital    area  population
BR         Brazil   Brasilia   8.516      200.40
RU         Russia     Moscow  17.100      143.50
IN          India  New Delhi   3.286     1252.00
CH          China    Beijing   9.597     1357.00
SA  South Africa   Pretoria   1.221       52.98
```

```
import numpy as np
np.logical_and(brics["area"] > 8, brics["area"] < 10)
```

```
BR     True
RU    False
IN    False
CH     True
SA    False
Name: area, dtype: bool
```

```
brics[np.logical_and(brics["area"] > 8, brics["area"] < 10)]
```

```
   country   capital   area  population
BR  Brazil  Brasilia  8.516       200.4
CH   China   Beijing  9.597      1357.0
```

Let's practice!

INTERMEDIATE PYTHON