**Intermediate Python for Data Science**

**#3 Comparsion and Operators**

**Equality**

To check if two Python values, or variables, are equal you can use ==. To check for inequality, you need !=. As a refresher, have a look at the following examples that all result in True. Feel free to try them out in the IPython Shell.

2 == (1 + 1)

"intermediate" != "python"

True != False

"Python" != "python"

When you write these comparisons in a script, you will need to wrap a [**print()**](https://docs.python.org/3/library/functions.html#print) function around them to see the output.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* In the editor on the right, write code to see if Trueequals False.
* Write Python code to check if -5 \* 15 is *not* equal to 75.
* Ask Python whether the strings "pyscript" and "PyScript" are equal.
* What happens if you compare booleans and integers? Write code to see if True and 1 are equal.

Script.py  
01 # Comparison of booleans

02 print(True == False)

03

04 # Comparison of integers

05 print(-5 \* 15 != 75)

06

07 # Comparison of strings

08 print("pyscript" == "PyScript")

09

10 # Compare a boolean with an integer

11 print(True == 1)

IPython Shell  
In [1]: # Comparison of booleans

print(True == False)

# Comparison of integers

print(-5 \* 15 != 75)

# Comparison of strings

print("pyscript" == "PyScript")

# Compare a boolean with an integer

print(True == 1)

False

True

False

True

<script.py> output:

False

True

False

True

In [2]:

**Greater and less than**

In the video, Filip also talked about the less than and greater than signs, < and > in Python. You can combine them with an equals sign: <= and >=. Pay attention: <= is valid syntax, but =< is not.

All Python expressions in the following code chunk evaluate to True:

3 < 4

3 <= 4

"alpha" <= "beta"

Remember that for string comparison, Python determines the relationship based on alphabetical order.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Write Python expressions, wrapped in a [**print()**](https://docs.python.org/3/library/functions.html#print)function, to check whether:
  + x is greater than or equal to -10. x has already been defined for you.
  + "test" is less than or equal to y. y has already been defined for you.
  + True is greater than False.

Script.py  
01 # Comparison of integers

02 x = -3 \* 6

03 print(x >= -10)

04

05 # Comparison of strings

06 y = "test"

07 print(y >= "test")

08

09 # Comparison of booleans

10 print(True > False)

IPython Shell  
In [1]: # Comparison of integers

x = -3 \* 6

print(x >= -10)

# Comparison of strings

y = "test"

print(y >= "test")

# Comparison of booleans

print(True > False)

False

True

True

In [2]:

**Compare arrays**

Out of the box, you can also use comparison operators with Numpy arrays.

Remember areas, the list of area measurements for different rooms in your house from the previous course? This time there's two Numpy arrays: my\_house and your\_house. They both contain the areas for the kitchen, living room, bedroom and bathroom in the same order, so you can compare them.

**Instructions**

**100 XP**

* Using comparison operators, generate boolean arrays that answer the following questions:
  + Which areas in my\_house are greater than or equal to 18?
  + You can also compare two Numpy arrays element-wise. Which areas in my\_house are smaller than the ones in your\_house?

Make sure to wrap both commands in a [**print()**](https://docs.python.org/3/library/functions.html#print)statement, so that you can inspect the output.

Script.py  
01 # Create arrays

02 import numpy as np

03 my\_house = np.array([18.0, 20.0, 10.75, 9.50])

04 your\_house = np.array([14.0, 24.0, 14.25, 9.0])

05

06 # my\_house greater than or equal to 18

07 print(my\_house >= 18)

08

09 # my\_house less than your\_house

10 print(my\_house < your\_house)

IPython Shell  
In [1]: # Create arrays

import numpy as np

my\_house = np.array([18.0, 20.0, 10.75, 9.50])

your\_house = np.array([14.0, 24.0, 14.25, 9.0])

# my\_house greater than or equal to 18

print(my\_house >= 18)

# my\_house less than your\_house

print(my\_house < your\_house)

[ True True False False]

[False True True False]

<script.py> output:

[ True True False False]

[False True True False]

In [2]:

# and, or, not (1)

A boolean is either 1 or 0, True or False. With boolean operators such as and, or and not, you can combine these booleans to perform more advanced queries on your data.

In the sample code on the right, two variables are defined: my\_kitchen and your\_kitchen, representing areas.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Write Python expressions, wrapped in a [**print()**](https://docs.python.org/3/library/functions.html#print)function, to check whether:
  + my\_kitchen is bigger than 10 and smaller than 18.
  + my\_kitchen is smaller than 14 or bigger than 17.
  + double the area of my\_kitchen is smaller than triple the area of your\_kitchen.

Script.py  
01 # Define variables

02 my\_kitchen = 18.0

03 your\_kitchen = 14.0

04

05 # my\_kitchen bigger than 10 and smaller than 18?

06 print(my\_kitchen > 10 and my\_kitchen < 18)

07

08 # my\_kitchen smaller than 14 or bigger than 17?

09 print(my\_kitchen < 14 or my\_kitchen > 17)

10

11 # Double my\_kitchen smaller than triple your\_kitchen?

12 print(my\_kitchen \* 2 < your\_kitchen \* 3)

IPython Shell  
In [1]: # Define variables

my\_kitchen = 18.0

your\_kitchen = 14.0

# my\_kitchen bigger than 10 and smaller than 18?

print(my\_kitchen > 10 and my\_kitchen < 18)

# my\_kitchen smaller than 14 or bigger than 17?

print(my\_kitchen < 14 or my\_kitchen > 17)

# Double my\_kitchen smaller than triple your\_kitchen?

print(my\_kitchen \* 2 < your\_kitchen \* 3)

False

True

True

<script.py> output:

False

True

True

In [2]:

**and, or, not (2)**

To see if you completely understood the boolean operators, have a look at the following piece of Python code:

x = 8

y = 9

not(not(x < 3) and not(y > 14 or y > 10))

What will the result be if you execute these three commands in the IPython Shell?

*NB: Notice that not has a higher priority than and and or, it is executed first.*

**Instructions**

**50 XP**

**Instructions**

**50 XP**

**Possible Answers**

* 

True

* 

False

* 

Running these commands will result in an error.

**Boolean operators with Numpy**

Before, the operational operators like < and >=worked with Numpy arrays out of the box. Unfortunately, this is not true for the boolean operators and, or, and not.

To use these operators with Numpy, you will need **[np.logical\_and()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logical_and.html" \t "_blank)**, **[np.logical\_or()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logical_or.html" \t "_blank)** and **[np.logical\_not()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logical_not.html" \t "_blank)**. Here's an example on the my\_house and your\_house arrays from before to give you an idea:

np.logical\_and(your\_house > 13,

your\_house < 15)

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Generate boolean arrays that answer the following questions:
* Which areas in my\_house are greater than 18.5 or smaller than 10?
* Which areas are smaller than 11 in both my\_houseand your\_house? Make sure to wrap both commands in [**print()**](https://docs.python.org/3/library/functions.html#print) statement, so that you can inspect the output.

Script.py  
01 # Create arrays

02 import numpy as np

03 my\_house = np.array([18.0, 20.0, 10.75, 9.50])

04 your\_house = np.array([14.0, 24.0, 14.25, 9.0])

05

06 # my\_house greater than 18.5 or smaller than 10

07 print(np.logical\_or(my\_house > 18.5, my\_house < 10) )

08

09 # Both my\_house and your\_house smaller than 11

10 print(np.logical\_and(my\_house < 11, your\_house < 11))

IPython Shell  
In [1]: # Create arrays

import numpy as np

my\_house = np.array([18.0, 20.0, 10.75, 9.50])

your\_house = np.array([14.0, 24.0, 14.25, 9.0])

# my\_house greater than 18.5 or smaller than 10

print(np.logical\_or(my\_house > 18.5, my\_house < 10) )

# Both my\_house and your\_house smaller than 11

print(np.logical\_and(my\_house < 11, your\_house < 11))

[False True False True]

[False False False True]

<script.py> output:

[False True False True]

[False False False True]

In [2]:

**if**

It's time to take a closer look around in your house.

Two variables are defined in the sample code: room, a string that tells you which room of the house we're looking at, and area, the area of that room.

**Instructions**

**100 XP**

* Examine the if statement that prints out "Looking around in the kitchen." if room equals "kit".
* Write another if statement that prints out "big place!" if area is greater than 15.

Script.py  
01 room = "kit"

02 area = 14.0

03

04 # if statement for room

05 if room == "kit" :

06 print("looking around in the kitchen.")

07

08 # if statement for area

09 if area > 15:

10 print("big place!")

IPython Shell  
In [1]: # Define variables

room = "kit"

area = 14.0

# if statement for room

if room == "kit" :

print("looking around in the kitchen.")

# if statement for area

if area > 15:

print("big place!")

looking around in the kitchen.

<script.py> output:

looking around in the kitchen.

In [2]:

# Add else

On the right, the if construct for room has been extended with an else statement so that "looking around elsewhere." is printed if the condition room == "kit" evaluates to False.

Can you do a similar thing to add more functionality to the ifconstruct for area?

##### Instructions

**100 XP**

Add an else statement to the second control structure so that "pretty small." is printed out if area > 15 evaluates to False.

Script.py  
01 # Define variables

02 room = "kit"

03 area = 14.0

04

05 # if-else construct for room

06 if room == "kit" :

07 print("looking around in the kitchen.")

08 else :

09 print("looking around elsewhere.")

10

11 # if-else construct for area

12 if area > 15 :

13 print("big place!")

14 else :

15 print("pretty small.")

IPython Shell  
In [1]: # Define variables

room = "kit"

area = 14.0

# if-else construct for room

if room == "kit" :

print("looking around in the kitchen.")

else :

print("looking around elsewhere.")

# if-else construct for area

if area > 15 :

print("big place!")

else :

print("pretty small.")

looking around in the kitchen.

pretty small

In [2]:

# Customize further: elif

It's also possible to have a look around in the bedroom. The sample code contains an elif part that checks if room equals "bed". In that case, "looking around in the bedroom." is printed out.

It's up to you now! Make a similar addition to the second control structure to further customize the messages for different values of area.

##### Instructions

**100 XP**

Add an elif to the second control structure such that "medium size, nice!" is printed out if area is greater than 10.

Script.py  
01 # Define variables

02 room = "kit"

03 area = 14.0

04

05 # if-else construct for room

06 if room == "kit" :

07 print("looking around in the kitchen.")

08 else :

09 print("looking around elsewhere.")

10

11 # if-else construct for area

12 if area > 15 :

13 print("big place!")

14 elif area > 10 ;

15 print("medium size, nice!")

16 else :

17 print("pretty small.")

IPython Shell  
In [1]: # Define variables

room = "bed"

area = 14.0

# if-elif-else construct for room

if room == "kit" :

print("looking around in the kitchen.")

elif room == "bed":

print("looking around in the bedroom.")

else :

print("looking around elsewhere.")

# if-elif-else construct for area

if area > 15 :

print("big place!")

elif area > 10 :

print("medium size, nice!")

else :

print("pretty small.")

looking around in the bedroom.

medium size, nice!

In [2]:

**Driving right (1)**

Remember that cars dataset, containing the cars per 1000 people (cars\_per\_cap) and whether people drive right (drives\_right) for different countries (country)? The code that imports this data in CSV format into Python as a DataFrame is available on the right.

In the video, you saw a step-by-step approach to filter observations from a DataFrame based on boolean arrays. Let's start simple and try to find all observations in cars where drives\_right is True.

drives\_right is a boolean column, so you'll have to extract it as a Series and then use this boolean Series to select observations from cars.

**Instructions**

**100 XP**

* Extract the drives\_right column *as a Pandas Series* and store it as dr.
* Use dr, a boolean Series, to subset the cars DataFrame. Store the resulting selection in sel.
* Print sel, and assert that drives\_right is True for all observations.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Extract drives\_right column as Series: dr

06 dr = cars['drives\_right']

07

08 # Use dr to subset cars: sel

09 sel = cars[dr]

10

11 # Print sel

12 print(sel)

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Extract drives\_right column as Series: dr

dr = cars['drives\_right']

# Use dr to subset cars: sel

sel = cars[dr]

# Print sel

print(sel)

cars\_per\_cap country drives\_right

US 809 United States True

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

<script.py> output:

cars\_per\_cap country drives\_right

US 809 United States True

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [2]:

# Driving right (2)

The code in the previous example worked fine, but you actually unnecessarily created a new variable dr. You can achieve the same result without this intermediate variable. Put the code that computes dr straight into the square brackets that select observations from cars.

##### Instructions

**100 XP**

Convert the code on the right to a one-liner that calculates the variable sel as before.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Convert code to a one-liner

06 sel = cars[cars['drives\_right']]

07

08 # Print sel

09 print(sel)

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Convert code to a one-liner

sel = cars[cars['drives\_right']]

# Print sel

print(sel)

cars\_per\_cap country drives\_right

US 809 United States True

RU 200 Russia True

MOR 70 Morocco True

EG 45 Egypt True

In [2]:

**Cars per capita (1)**

Let's stick to the cars data some more. This time you want to find out which countries have a high *cars per capita* figure. In other words, in which countries do many people have a car, or maybe multiple cars.

Similar to the previous example, you'll want to build up a boolean Series, that you can then use to subset the cars DataFrame to select certain observations. If you want to do this in a one-liner, that's perfectly fine!

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Select the cars\_per\_cap column from cars as a Pandas Series and store it as cpc.
* Use cpc in combination with a comparison operator and 500. You want to end up with a boolean Series that's True if the corresponding country has a cars\_per\_cap of more than 500and False otherwise. Store this boolean Series as many\_cars.
* Use many\_cars to subset cars, similar to what you did before. Store the result as car\_maniac.
* Print out car\_maniac to see if you got it right.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Create car\_maniac: observations that have a cars\_per\_cap over 500

06 cpc = cars['cars\_per\_cap']

07 many\_cars = cpc > 500

08 car\_maniac = cars[many\_cars]

09

10 # Print car\_maniac

11 print(car\_maniac)

IPython Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Create car\_maniac: observations that have a cars\_per\_cap over 500

cpc = cars['cars\_per\_cap']

many\_cars = cpc > 500

car\_maniac = cars[many\_cars]

# Print car\_maniac

print(car\_maniac)

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

<script.py> output:

cars\_per\_cap country drives\_right

US 809 United States True

AUS 731 Australia False

JAP 588 Japan False

In [2]:

**Cars per capita (2)**

Remember about **[np.logical\_and()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logical_and.html" \t "_blank)**, **[np.logical\_or()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logical_or.html" \t "_blank)** and **[np.logical\_not()](http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logical_not.html" \t "_blank)**, the Numpy variants of the and, or and not operators? You can also use them on Pandas Series to do more advanced filtering operations.

Take this example that selects the observations that have a cars\_per\_cap between 10 and 80. Try out these lines of code step by step to see what's happening.

cpc = cars['cars\_per\_cap']

between = np.logical\_and(cpc > 10, cpc < 80)

medium = cars[between]

**Instructions**

**100 XP**

* Use the code sample above to create a DataFrame medium, that includes all the observations of cars that have a cars\_per\_cap between 100 and 500.
* Print out medium.

Script.py  
01 # Import cars data

02 import pandas as pd

03 cars = pd.read\_csv('cars.csv', index\_col = 0)

04

05 # Import numpy, you'll need this

06 import numpy as np

07

08 # Create medium: observations with cars\_per\_cap between 100 and 500

09 cpc = cars['cars\_per\_cap']

10 between = np.logical\_and(cpc > 100, cpc <500)

11 medium = cars[between]

12

13 # Print medium

14 print(medium)

IPytnron Shell  
In [1]: # Import cars data

import pandas as pd

cars = pd.read\_csv('cars.csv', index\_col = 0)

# Import numpy, you'll need this

import numpy as np

# Create medium: observations with cars\_per\_cap between 100 and 500

cpc = cars['cars\_per\_cap']

between = np.logical\_and(cpc > 100, cpc <500)

medium = cars[between]

# Print medium

print(medium)

cars\_per\_cap country drives\_right

RU 200 Russia True

<script.py> output:

cars\_per\_cap country drives\_right

RU 200 Russia True

In [2]: