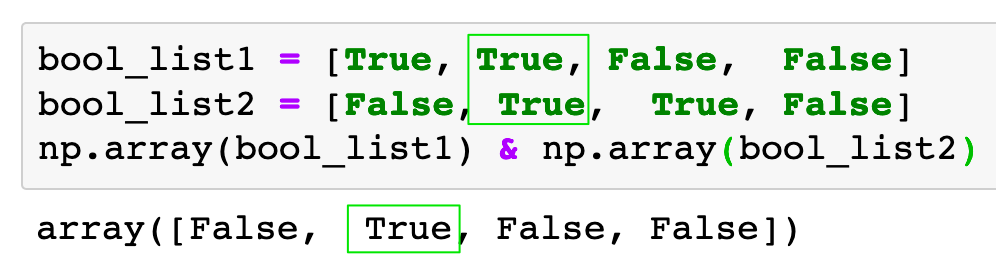
**Filter on Multiple Conditions: International Films**

So far, we've created subsets for our DataFrames based on a single condition. But what if we want to select our data based on more than one condition? For example, which films made money internationally (i.e., data.USD\_Worldwide\_Gross != 0), but had zero box office revenue in the United States (i.e., data.USD\_Domestic\_Gross == 0)?

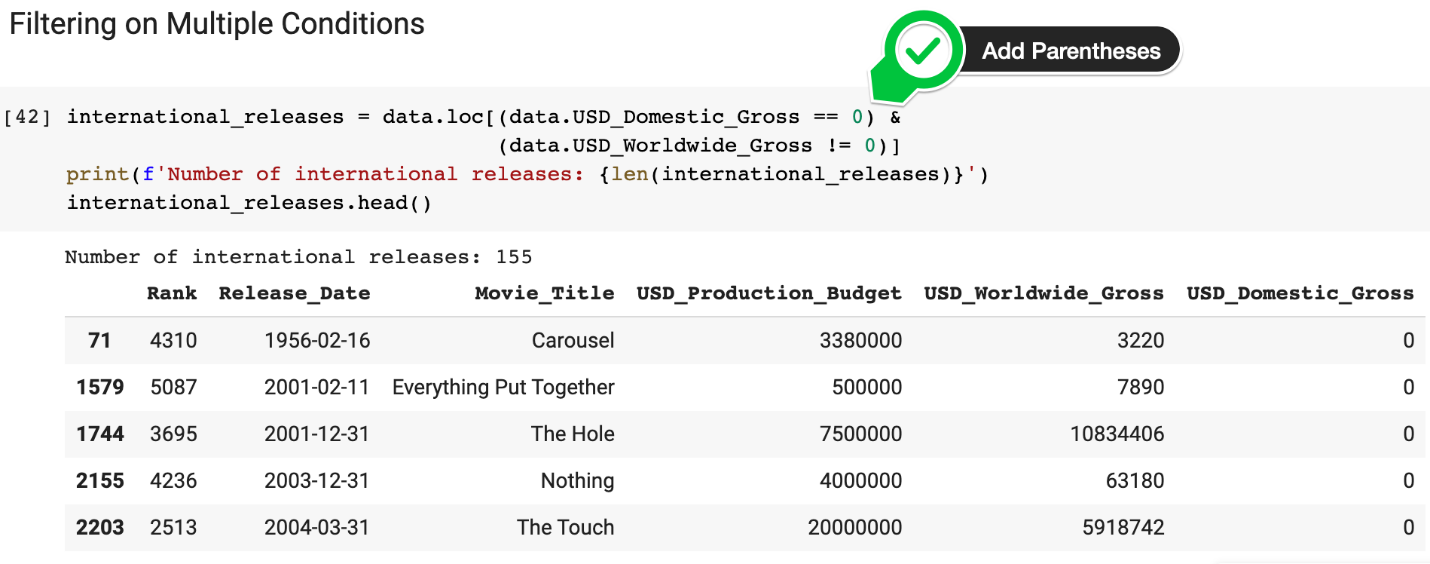
How would we create a filter for these two conditions? One approach is to use the .loc[] property combined with the [bitwise and](https://docs.python.org/3.4/library/operator.html#mapping-operators-to-functions) & operator.

1. international\_releases = data.loc[(data.USD\_Domestic\_Gross == 0) &
2. (data.USD\_Worldwide\_Gross != 0)]

Why does this work? Pandas is built on top of NumPy, which uses Python's bitwise operators. And these bitwise operators allow us to do comparisons on an element by element basis in both NumPy and Pandas! Here's an example:



However, we're also checking if the domestic revenue was zero and the worldwide revenue was not zero. Because the bitwise operator takes precedence, we need to include parentheses () around the comparisons we'd like to prioritise.



However, this is not the only technique we can use to make multiple comparisons.

Challenge

Use the Pandas [.query() function](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html) to accomplish the same thing. Create a subset for international releases that had some worldwide gross revenue, but made zero revenue in the United States.

Hint: This time you'll have to use the and keyword.

.

.

..

...

..

.

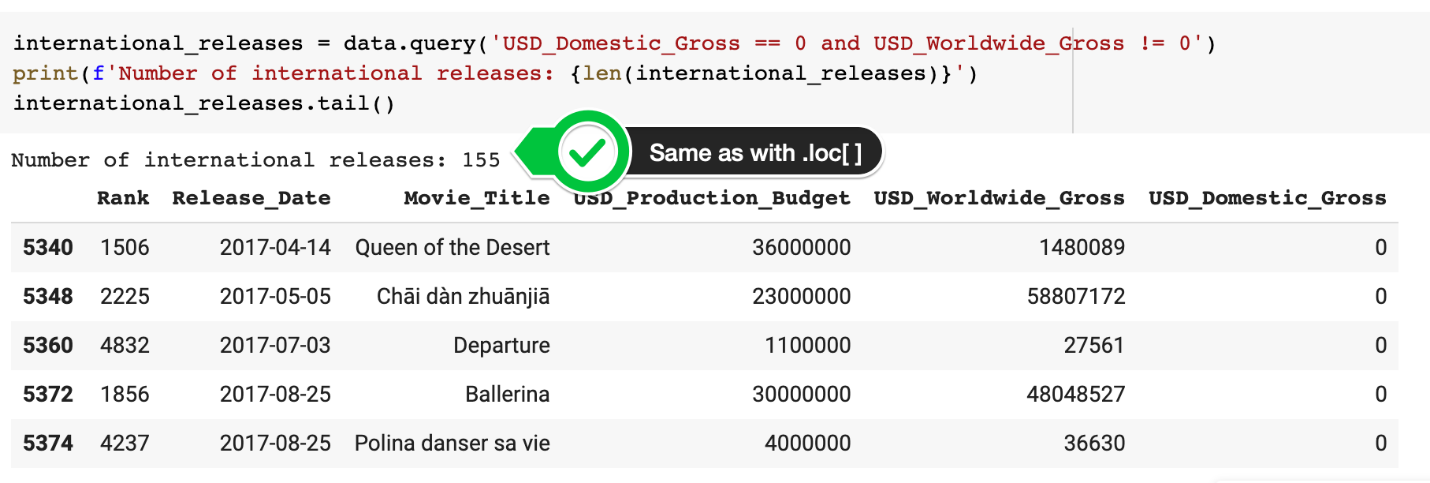
.

**Solution: Using the .query() function to filter on multiple conditions**

In this case, we enclose the entire query inside a string.

1. international\_releases = data.query('USD\_Domestic\_Gross == 0 and USD\_Worldwide\_Gross != 0')
2. print(f'Number of international releases: {len(international\_releases)}')
3. international\_releases.tail()

The column names are recognised and we see the following:



Unreleased Films

Now we can turn our attention to films in the dataset that were not released at the time the data was collected. This is why films like Singularity and Aquaman had zero revenue.

Challenge

* Identify which films were not released yet as of the time of data collection (May 1st, 2018).
* How many films are included in the dataset that have not yet had a chance to be screened in the box office?
* Create another DataFrame called data\_clean that does not include these films.

.

.

..

...

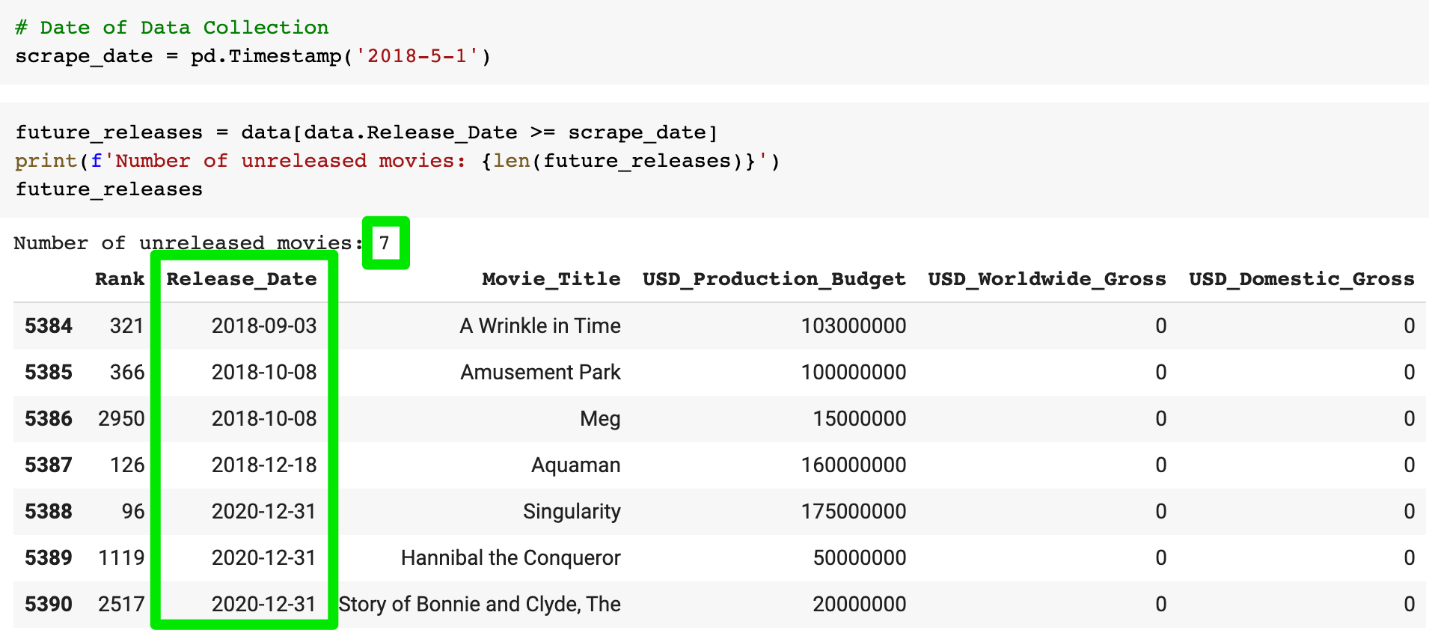
..

.

.

**Solution: Removing the unreleased films**

There are a total of 7 unreleased films at the time of data collection included in the dataset.



From this point on, we'll work with another DataFrame called data\_clean that does not include these films.

1. data\_clean = data.drop(future\_releases.index)

Bonus Challenge: Films that Lost Money

Having removed the unreleased films entirely can you calculate the percentage of films that did not break even at the box office? We already saw that more than the bottom quartile of movies appears to lose money when we ran .describe(). However, what is the true percentage of films where the costs exceed the worldwide gross revenue?

.

.

..

...

..

.

.

**Solution: Budget greater than revenue**

Again, there are different ways you could have calculated this. For example, using the .loc[] property,

1. money\_losing = data\_clean.loc[data\_clean.USD\_Production\_Budget > data\_clean.USD\_Worldwide\_Gross]
2. len(money\_losing)/len(data\_clean)

or the .query() function

1. money\_losing = data\_clean.query('USD\_Production\_Budget > USD\_Worldwide\_Gross')
2. money\_losing.shape[0]/data\_clean.shape[0]

In both cases, we see that a whopping 37.2% 😮 of films do not recoup their production budget at the box office. 💸💸💸 Who knew that film finance could be so risky! 😬