Index Alignment - due tue 09/21 | *by Charlotte*

Exercise 1

Use the code below to get started:

In [166…
```
import pandas as pd
attendees = pd.DataFrame({'names': ["Jill", "Kumar", "Zaira"],
                          'prizes': [0, 0, 0],
                          'arrival_order': [2, 1, 3]})
arrival_prizes = pd.Series([20, 10, 0])
```

Exercise 2

Now let's sort our attendees list by arrival_order so that the first row is the person who arrived first, the second is the person who arrived second, etc. to match how we've organized arrival_prizes.

In [167…
```
attendees = attendees.sort_values('arrival_order')
attendees
```

Out[167…

	names	prizes	arrival_order
1	Kumar	0	1
0	Jill	0	2
2	Zaira	0	3

Exercise 3

Now let's "give" everyone their arrival prizes by adding arrival prizes to people's prize column:

In [168…
```
attendees['prizes'] = arrival_prizes
attendees
```

Out[168…

	names	prizes	arrival_order
1	Kumar	10	1
0	Jill	20	2
2	Zaira	0	3

Exercise 4

Now let's look at the result. Does it look like what you expected? Do you know what went wrong?

- This is now what we expected. Kumar should have received 20 Dollars, Jill 10 Dollars, and Zaira 0 Dollars. The result for Zaira is correct, because her position never changed but Jill and Kumar's prize is switched. What happened, I assume, is that for exercise 2 I created a view and did not change the order in-place. When I added the prizes, they were added, not in the way I had now sorted but in the way the original dataframe was sorted and based on their index value (0-2), i.e. with Jill first and Kumar second (and Zaira third).
- As I learned in the *Discussion* the result was not in fact that we created a view, but because of the phenomenon called **index alignment**.

Exercise 5

So reset prizes to 0.

In [169…

```
attendees['prizes'] = [0, 0, 0]
attendees
```

Out[169…

	names	prizes	arrival_order
1	Kumar	0	1
0	Jill	0	2
2	Zaira	0	3

Reset the index.

In [170…

```
attendees = attendees.reset_index(drop=True)
attendees
```

Out[170…

	names	prizes	arrival_order
0	Kumar	0	1
1	Jill	0	2
2	Zaira	0	3

...and try again.

In [171…

```
attendees['prizes'] = arrival_prizes
attendees
```

Out[171…

	names	prizes	arrival_order
0	Kumar	20	1
1	Jill	10	2
2	Zaira	0	3

Exercise 6

OK, so besides doing automatic alignment, is there a reason to use indices?

Let's find out. Create the following fake dataset of social security numbers and some "names" (random strings). Warning: this will take a little time to run.

In [172…

```
import numpy.random as npr
import string
import random
npr.seed(42)
random.seed(42)

size=1000000 # 1,000,000
people = pd.DataFrame({'social_security_numbers': npr.randint(low=10000000,
                       'names': [''.join(random.choices(string.ascii_upperc
                                 for i in range(size)]})
```

Exercise 7

Now subset your data to get the social security number associated with the name of "TPKSMSLREI". (Yes, there are ways to get real random names, but they take a while to run because they query websites that generate fake names, so we're just doing this!).

In [173…
```
subs = people.loc[people['names'] == "TPKSMSLREI"]
subs
```

Out[173…

	social_security_numbers	names
299029	84423764	TPKSMSLREI

Exercise 8

Now time your operation using the %timeit ipython magic function.

In [174…
```
print("Time needed to run the operation is:")
%timeit set(subs)
```

```
Time needed to run the operation is:
713 ns ± 7.3 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Exercise 9

Now make *names* your index for this data. Then try subsetting using loc[] to get all the observations with the name of "TPKSMSLREI" and time the operation.

In [175…
```
people.set_index('names')
print("Time needed to run the operation after making names the index is:")
%timeit people.loc[people.names == "TPKSMSLREI"]
```

```
Time needed to run the operation after making names the index is:
32.7 ms ± 283 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```