Pandas brain teasers

EXERCISE YOUR MIND

```python
from io import StringIO
import pandas as pd

csv_data = '''\
day,hits
2020-01-01,400
2020-02-02,800
2020-02-03,600
'''

df = pd.read_csv(StringIO(csv_data))
print(df['day'].dt.month.unique())
```

WILL THIS CODE RUN? WHAT WILL IT PRINT?

25 MIND BENDING TEASERS & SOLUTIONS

MIKI TEBEKA

Copyright

Copyright © 2020, 353solutions LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Rectified

relu.py

```
import pandas as pd


def relu(n):
    if n < 0:
        return 0
    return n


arr = pd.Series([-1, 0, 1])
print(relu(arr))
```

Try to guess what the output is before moving to the next page.

This code will raise a `ValueError`.

The problematic line is `if n < 0:`, `n` is the result of `arr < 0` which is a `pandas.Series`.

```
In [1]: import pandas as pd
In [2]: arr = pd.Series([-1, 0, 1])
In [3]: arr < 0
Out[3]:
0     True
1    False
2    False
dtype: bool
```

Once `arr < 0` is computed, we use it in an `if` statement. Which brings us to how boolean values work in Python.

Every Python object, not only `True` and `False` has a boolean value. The documentation state the rules:

Everything is `True` except:

- 0 numbers: `0`, `0.0`, `0+0j` ...
- Empty collections: `[]`, `{}`, `''`, ...
- `None`
- `False`

You can test the truth value of a Python object using the built-in bool function.

On top of the above, any object can state its own boolean value using the `__bool__` special method. The boolean logic for `pandas.Series` is different than the one for a list or a tuple - it raises an exception.

```
In [4]: bool(arr < 0)
...
ValueError: The truth value of a Series is ambiguous.
Use a.empty, a.bool(), a.item(), a.any() or a.all().
```

The exception tells you the reasoning - it follows The Zen of Python which states:

> In the face of ambiguity, refuse the temptation to guess.

So, what are your options? You can use `all` or `any` but then you'll need to check the type of `n` to see if it's a plain number of a `pandas.Series`.

A function that works both on scalar and a `pandas.Series` (or a numpy array) is called a "ufunc",

short for "universal function". Most of the function from numpy or Pandas, such as `min`, `to_datetime`..., are ufuncs.

numpy has a vectorize decorator for these cases.

relu_vec.py

```
1  import numpy as np
2  import pandas as pd
3
4
5  @np.vectorize
6  def relu(n):
7      if n < 0:
8          return 0
9      return n
10
11
12 arr = pd.Series([-1, 0, 1])
13 print(relu(arr))
```

Now `relu` will work both on scalars (e.g. 7, 2.18 ...) and vectors (e.g. numpy array, `pandas.Series` ...)

The output of `relu` now is `numpy.ndarray`, **not** `pandas.Series`. You might want to have a look at numba.vectorize as well.

Further Reading

- Truth value testing in the Python documentation
- PEP 285 - Adding a bool type
- __bool__ documentation
- Universal functions on the numpy docs