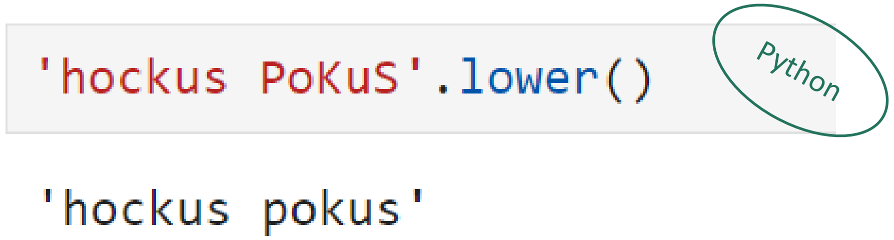
Text manipulations

- .str
- Replace()
- Index methods

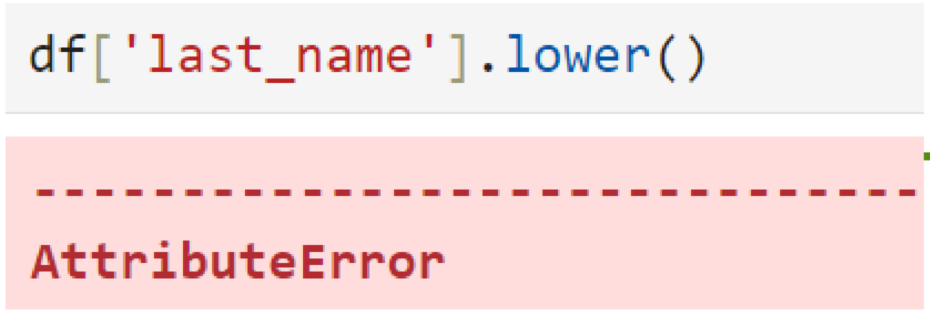
String methods in pandas vs python

Why .str?
In Python, string methods (e.g., lower(), replace()) can be applied directly to strings.
In pandas, columns in a DataFrame are series objects, not plain Python lists.
To apply string methods on series, use .str.



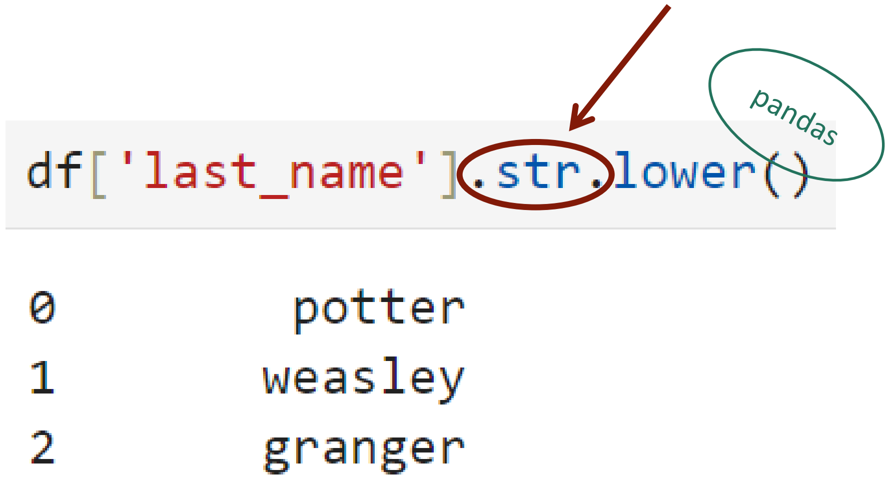
```
'hockus PoKuS'.lower()
```

```
'hockus pokus'
```



```
df['last_name'].lower()
```

```
------------------------------
AttributeError
```



```
df['last_name'].str.lower()
```

```
0          potter
1         weasley
2         granger
```

.str bridges the gap between Python string methods and pandas series.

Combining Methods

Convert names to lowercase and check if they start with "a"

```
df['names'].str.lower().str.startswith('a')
```

Chain .str methods to perform multiple operations in one line.

Replace()

General Syntax:

```python
df['column_name'] = df['column_name'].str.replace('old', 'new')
```

Replacing $ signs with spaces

```python
df['price'] = df['price'].str.replace('$', '')
```

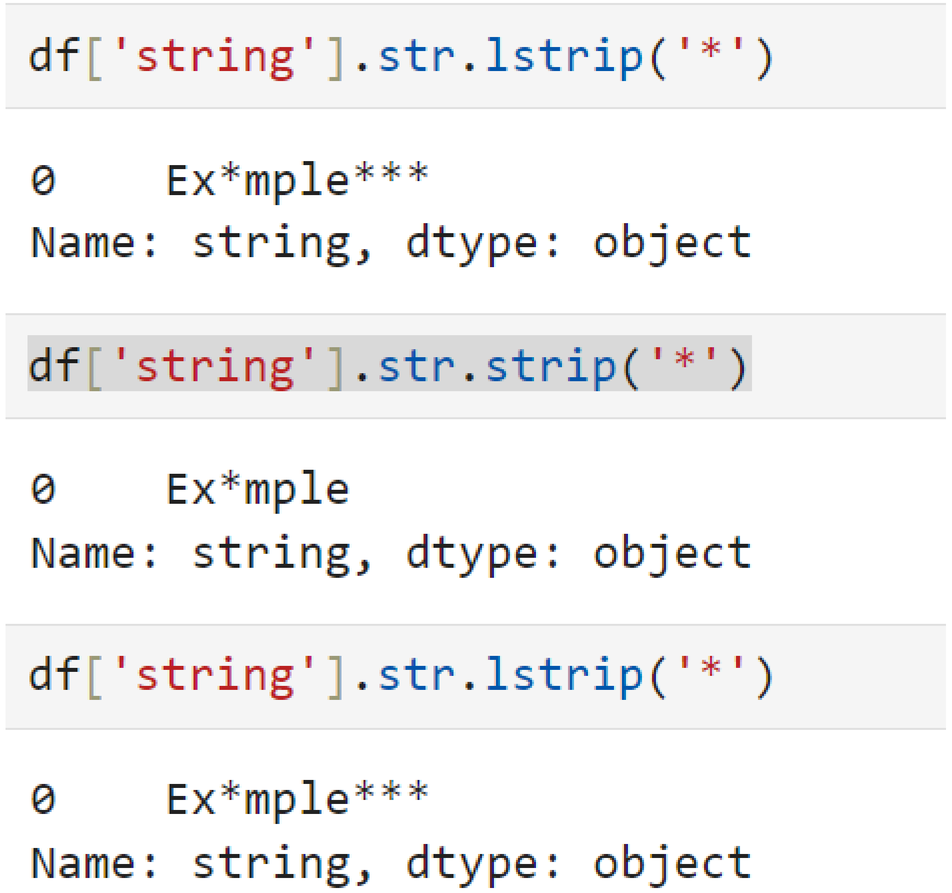
Correcting typos or formatting issues:

```python
df['names'] = df['names'].str.replace('Jon', 'John')
```

rstrip(), lstrip(), and strip()

Trimming Whitespace or Characters:

- strip(): Removes leading and trailing spaces (or characters)
- lstrip(): Removes leading spaces (or characters).
- rstrip(): Removes trailing spaces (or characters)



```
df['string'].str.lstrip('*')

0     Ex*mple***
Name: string, dtype: object

df['string'].str.strip('*')

0     Ex*mple
Name: string, dtype: object

df['string'].str.lstrip('*')

0     Ex*mple***
Name: string, dtype: object
```

Text based index

Text-based index values can be transformed just like column values.

```
cereal.index[0:3].str.upper()
```

```
Index(['100% BRAN', '100% NATURAL BRAN', 'ALL-BRAN'],
```

```
cereal.index[0:3].str.len()
```

```
Index([9, 17, 8], dtype='int64', name='name')
```

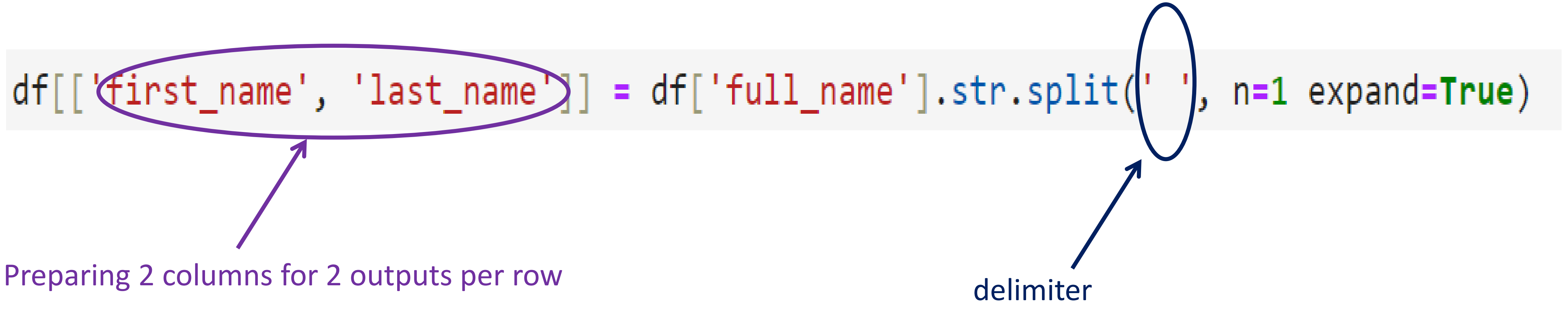
Split()

Why Split Strings?
•Separate values within a column into multiple columns or lists.

Options:

1. **n=**: Limit the number of splits
2. **expand=True**: Create multiple new columns

In this example the full_name column will be split once at the space into first_name and last_name



```
df[['first_name', 'last_name']] = df['full_name'].str.split(' ', n=1 expand=True)
```

Summary and Best Practices

- Use .str for string operations in pandas.
- Chain methods for concise transformations.
- Clean your data with replace, strip, and split.Leverage options like n= and expand=True for splitting.
- Don’t forget text-based indices!

TECHNION
Azrieli Continuing Education and External Studies Division