Pandas I

10.2.24

What is Pandas?

- Environment for structured data
- Builtin methods for data cleaning, summarization

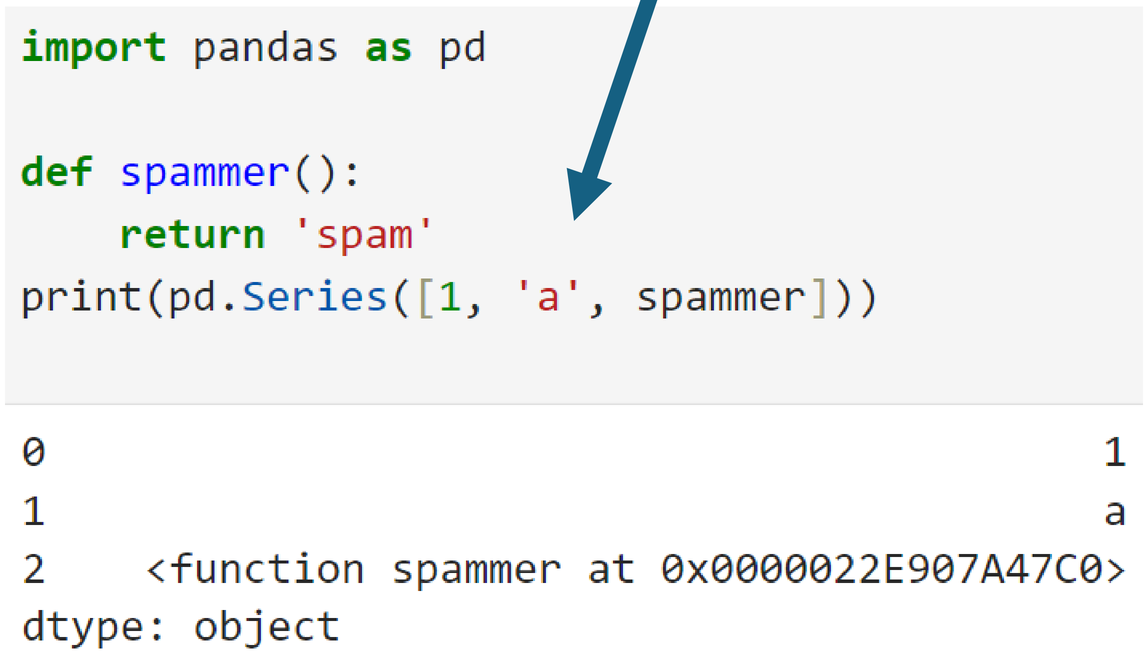
2 new classes:

- Series: 1d, untyped like a list
- DataFrame: table, untyped

Series = list with style

- Like lists, series can hold anything
- Series come with attached methods, new indexing, and more efficient storage
- Literally one column of a dataFrame, hence we'll just talk about dataFrames for simplicity

Putting an int, str, and fct in a series container



```python
import pandas as pd

def spammer():
    return 'spam'
print(pd.Series([1, 'a', spammer]))
```

```
0                                          1
1                                          a
2    <function spammer at 0x0000022E907A47C0>
dtype: object
```

DataFrames = table with style

Building dataframes:

pd.DataFrame(.....)

- Dictionaries: elements must be lists
- NumPy arrays
- Series
- Anything that can become a NumPy array

pd.read_csv(....)
pd.read_excel(....)
pd.read_json(....)
pd.read_clipboard(...)

This converts whatever you've copied (as in ctrl+c) into a DataFrame

Structuring DataFrames from arrays

By default **pd.DataFrame** labels the columns/rows 0, 1, 2 etc.

Assign values in call

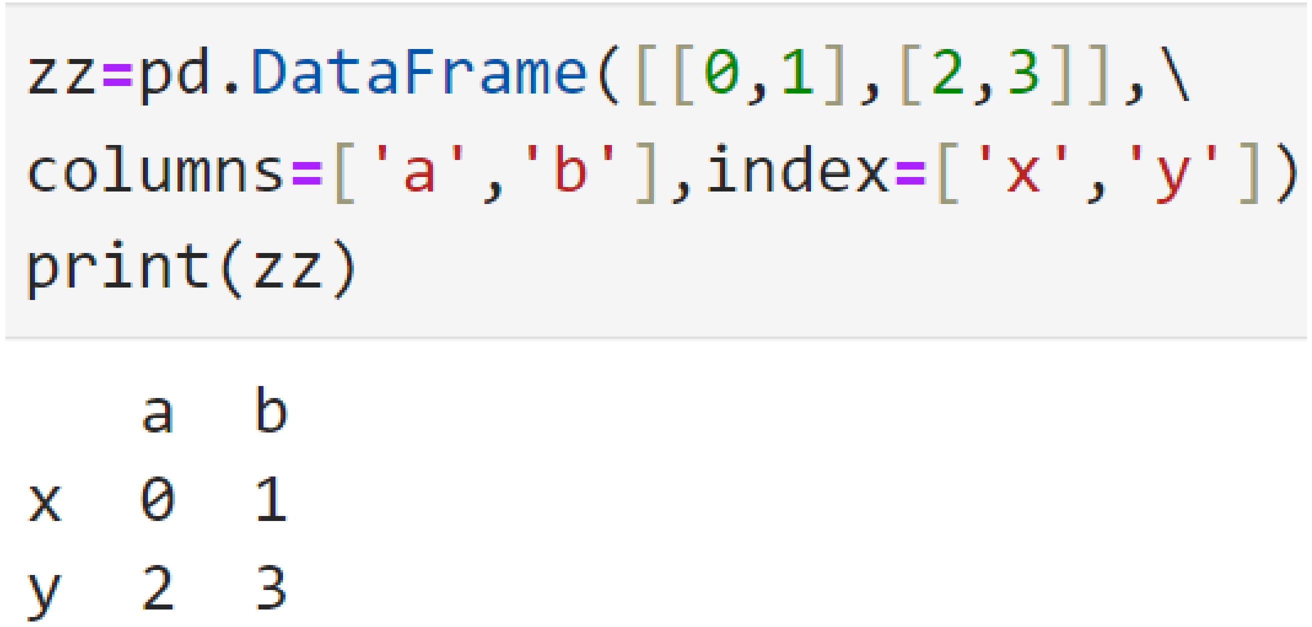
    columns=

    index= (rows)

Change later:

    data.columns

    data.index



```
zz=pd.DataFrame([[0,1],[2,3]],\
columns=['a','b'],index=['x','y'])
print(zz)
```

```
   a  b
x  0  1
y  2  3
```

Indexing DataFrames

Indexing columns:

data['A']=[1,4,7]

data[['A', 'B']]=

1	2
4	5
7	8

data=

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

BUT, slice indices return **rows**:

data[:1]=[1,2]

iloc: integer/Boolean indexing

- **Behaves like numpy indexing**
- Primary dimension=row
- Combined: [row,column]
- end-exclusive

data=

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

loc: label-based or Boolean

data=

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

Label-Based:

```
data.loc[['X','Y'],['A','B']]
```

	A	B
X	1	2
Y	4	5

Slicing is always inclusive within **loc:**

```
data.loc[:,'A':'B']
```

	A	B
X	1	2
Y	4	5
Z	7	8

```
data2.loc[:2,:2]
```

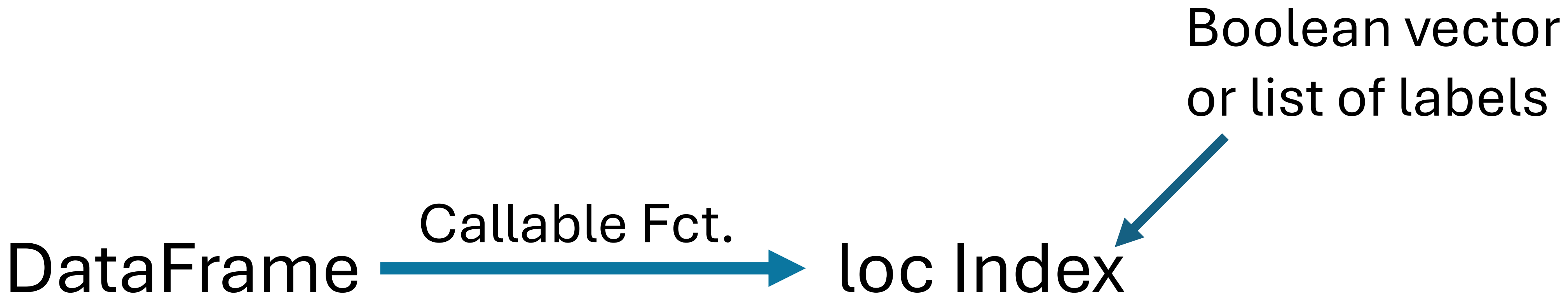
In this case labels are numeric (default)

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Callable-indexing in loc

- You can directly apply conditions to index using **loc** and a callable fct. e.g. as a lamda expression.
- The Callable will act on the DataFrame itself and should produce a valid indexer (Boolean or label vector)

data.loc[lambda x:..., lambda x:...**]**



Example: Select rows with A odd

data=

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

```
data.loc[lambda x: x['A']%2==1]
```

True

False

True

	A	B	C
X	1	2	3
Z	7	8	9

Example: Select Columns with X odd

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

```
data.loc[:,lambda x: x.loc['X',:]%2==1]
```

$\rightarrow$ [True, False, True] $\rightarrow$

	A	C
X	1	3
Y	4	6
Z	7	9

Pandas Operations

- By default, Pandas performs operations by label so:

dat1

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

+

dat2

	B	D	A
Y	1	2	3
P	4	5	6
X	7	8	9

=

dat1+dat2

	A	B	C	D
P	NaN	NaN	NaN	NaN
X	10	9	NaN	NaN
Y	7	6	NaN	NaN
Z	NaN	NaN	NaN	NaN

Pandas Operations

- Using values returns a NumPy array

dat1.values + dat2.values =

	A	B	C
X	1	2	3
Y	4	5	6
Z	7	8	9

	B	D	A
Y	1	2	3
P	4	5	6
X	7	8	9

	B	D	A
Y	2	4	6
P	8	10	12
X	14	16	18

Practice Together: Spike Data

- Idea Space: How to isolate the spikes?

Remainder in Jupyter….

Fin