Today's class

- Pandas

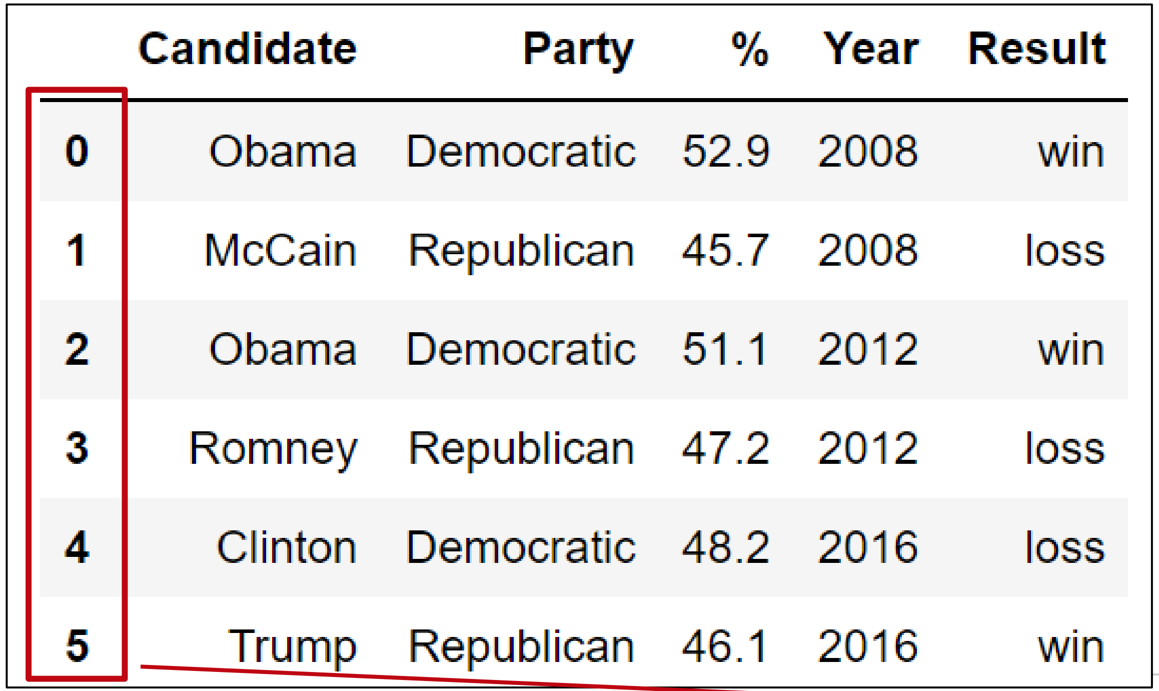
Pandas Data Structures:
Data Frames, Series, and Indices

Pandas Data Structures

There are three fundamental data structures in pandas:

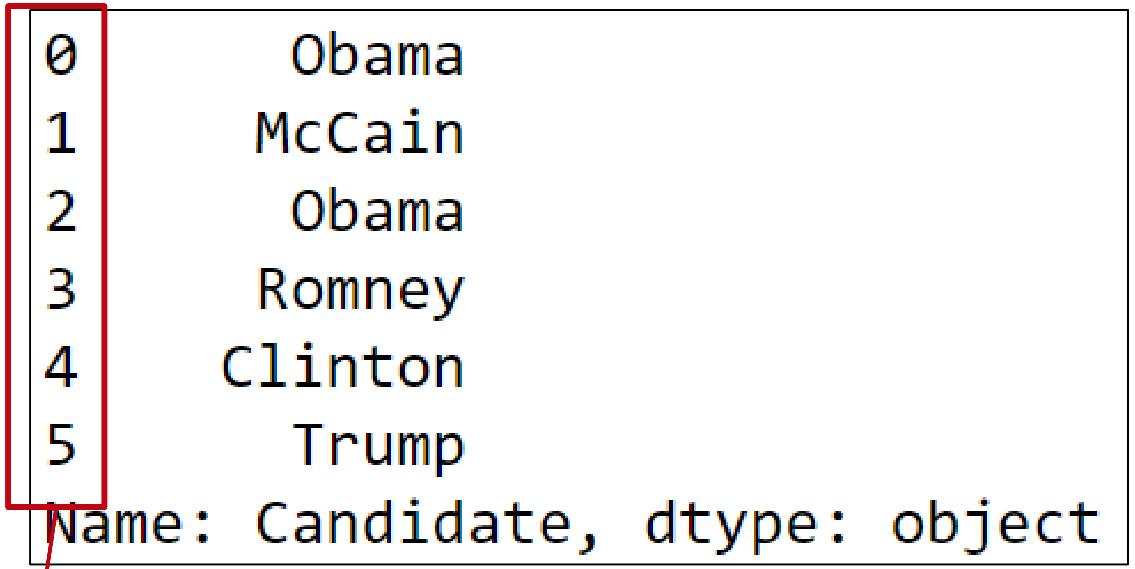
- Data Frame: 2D data tabular data.
- Series: 1D data. I usually think of it as columnar data.
- Index: A sequence of row labels.

Data Frame



	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Series



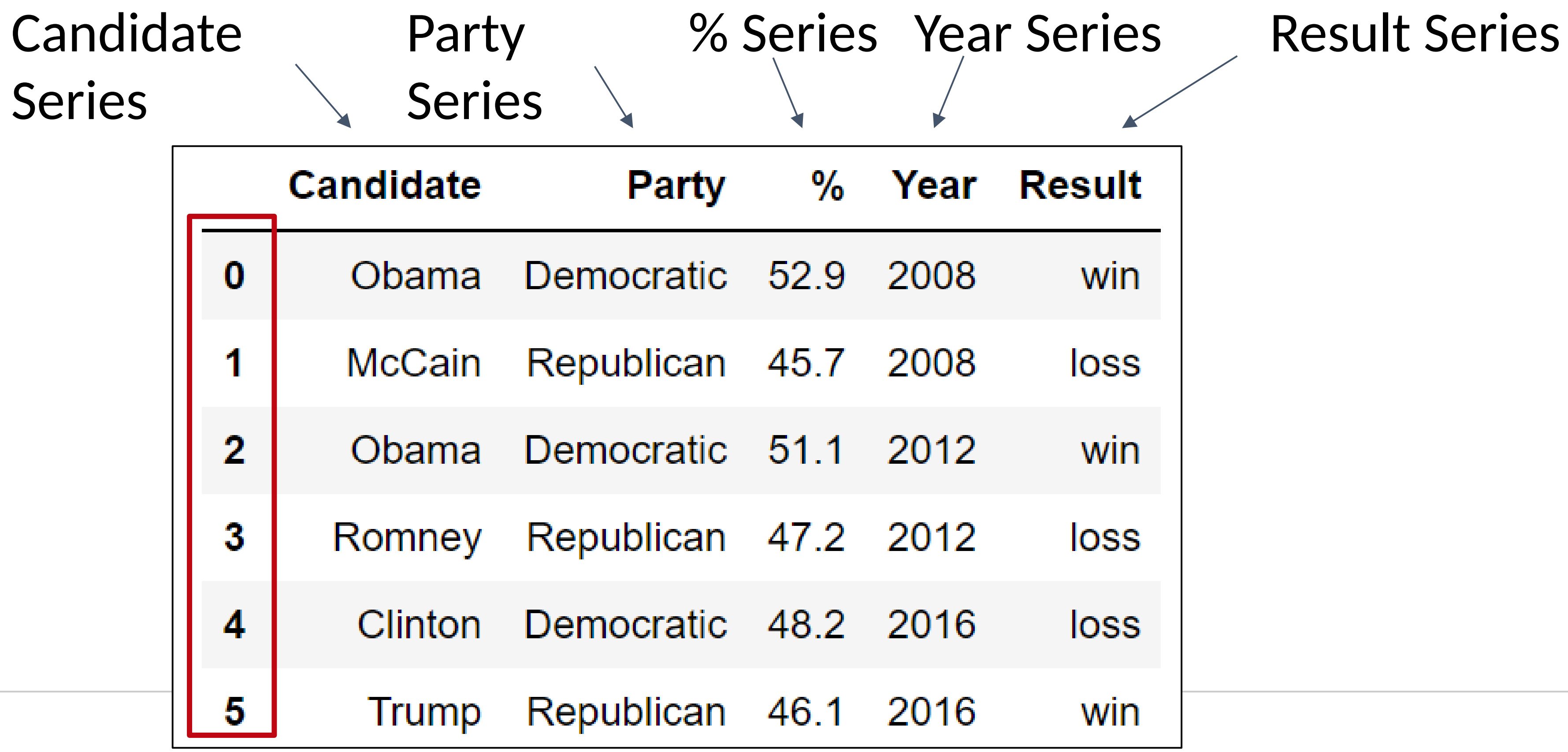
```
0        Obama
1       McCain
2        Obama
3       Romney
4      Clinton
5        Trump
Name: Candidate, dtype: object
```

Index

The Relationship Between Data Frames, Series, and Indices

We can think of a Data Frame as a collection of Series that all share the same Index.

- Candidate, Party, %, Year, and Result Series all share an index from 0 to 5.

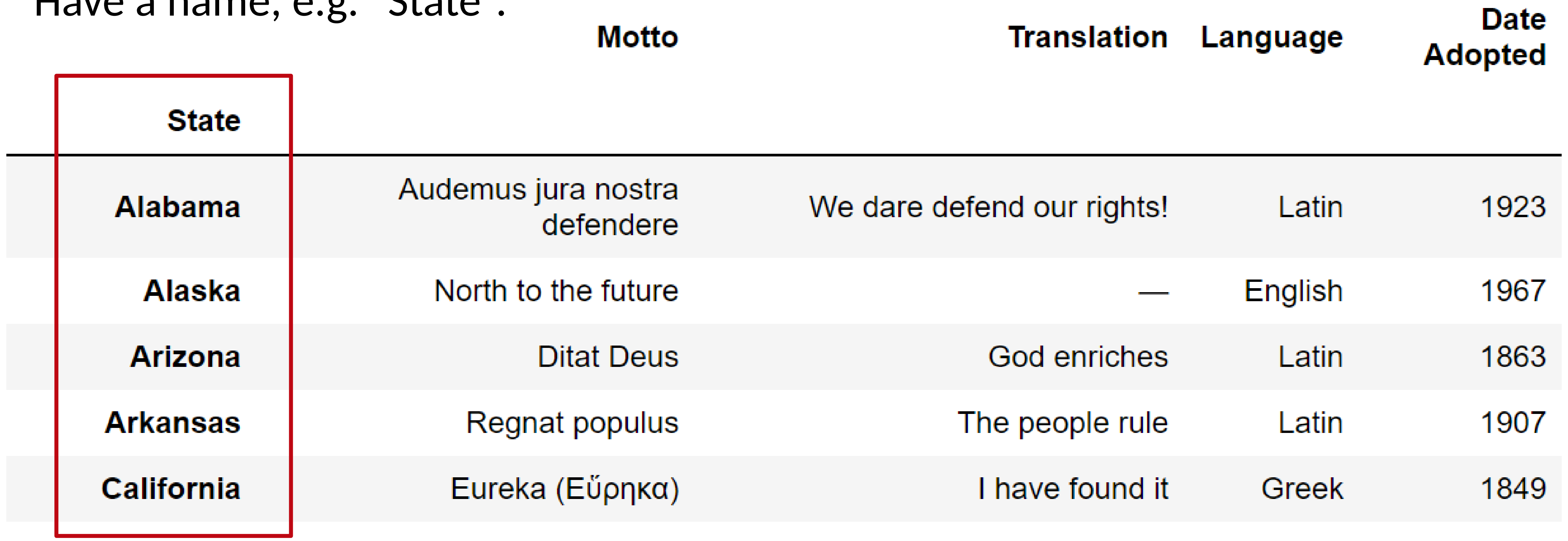


	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Indices Are Not Necessarily Row Numbers

Indices (a.k.a. row labels) can also:

- Be non-numeric.
- Have a name, e.g. “State”.

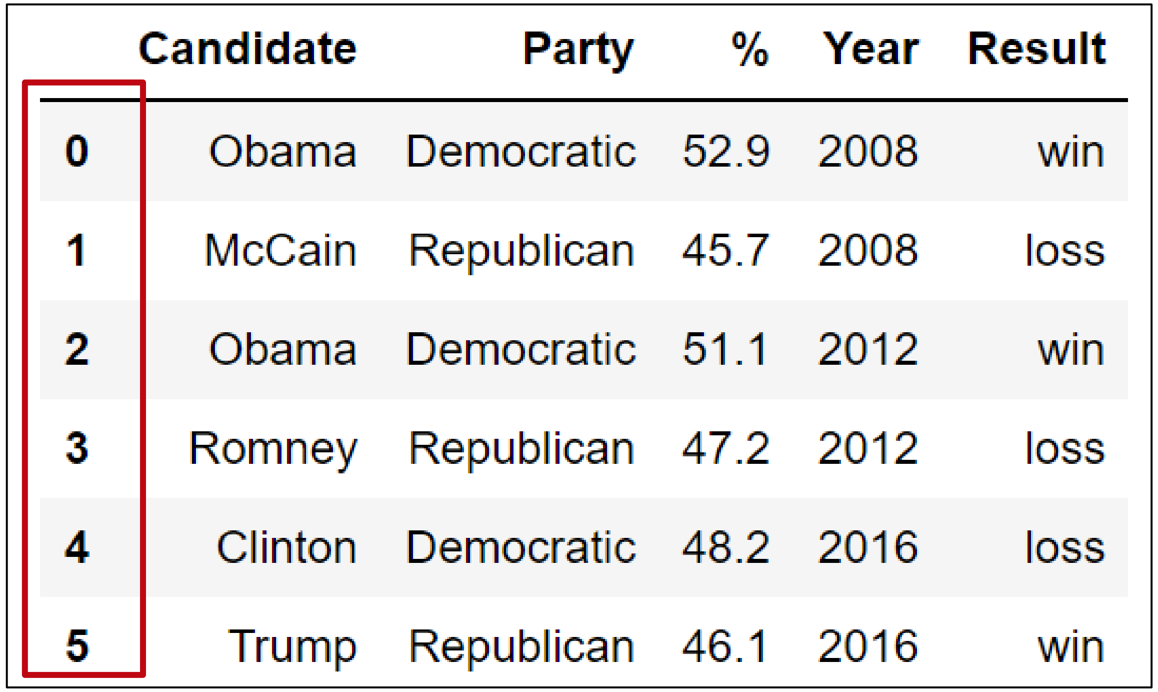


	Motto	Translation	Language	Date Adopted
State				
Alabama	Audemus jura nostra defendere	We dare defend our rights!	Latin	1923
Alaska	North to the future	—	English	1967
Arizona	Ditat Deus	God enriches	Latin	1863
Arkansas	Regnat populus	The people rule	Latin	1907
California	Eureka (Εὕρηκα)	I have found it	Greek	1849

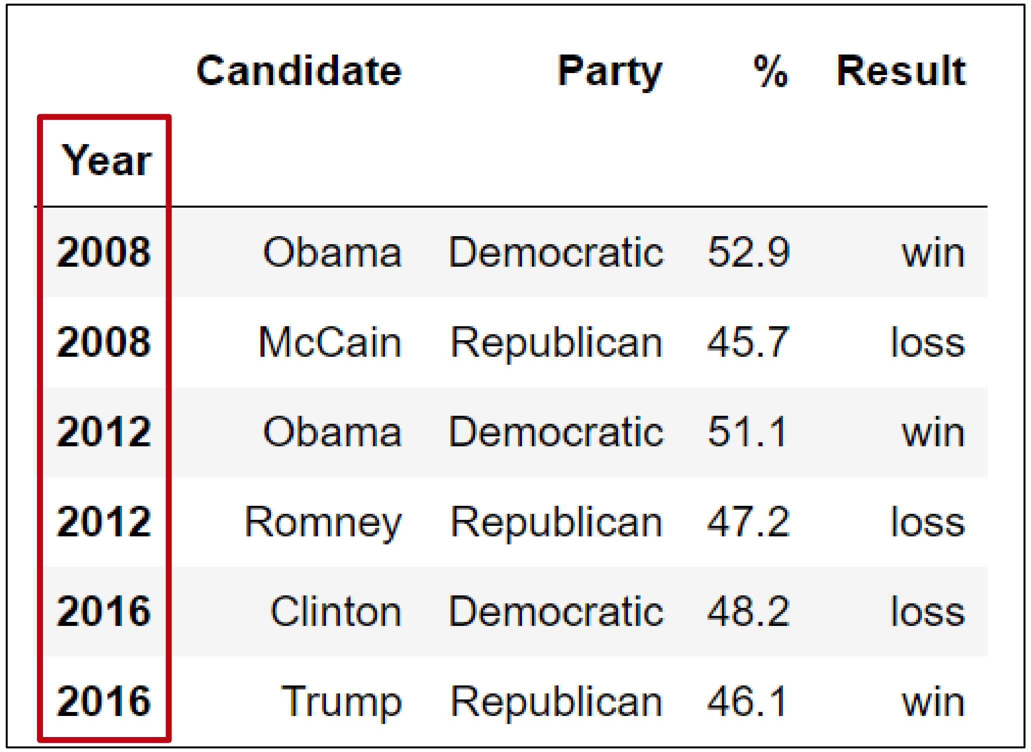
Indices

The row labels that constitute an index do not have to be unique.

- Left: The index values are all unique and numeric, acting as a row number.
- Right: The index values are named and non-unique.



	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

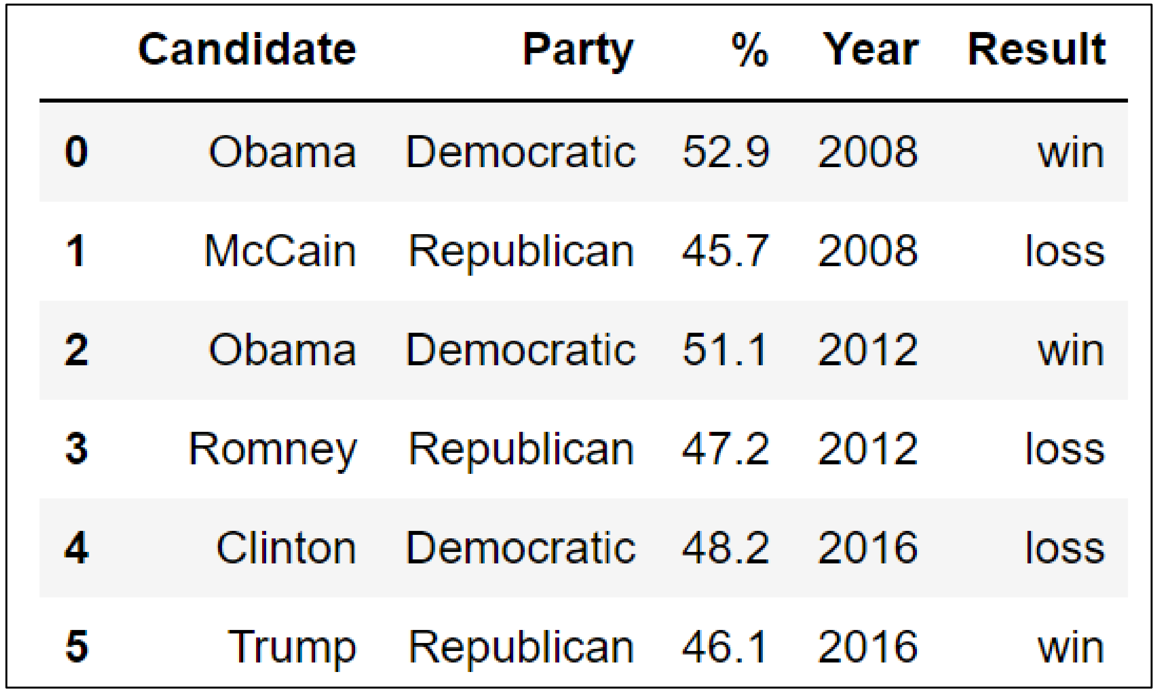


	Candidate	Party	%	Result
Year				
2008	Obama	Democratic	52.9	win
2008	McCain	Republican	45.7	loss
2012	Obama	Democratic	51.1	win
2012	Romney	Republican	47.2	loss
2016	Clinton	Democratic	48.2	loss
2016	Trump	Republican	46.1	win

Column Names Are Usually Unique!

Column names in Pandas are almost always unique!

- Example: Really shouldn’t have two columns named “Candidate”.



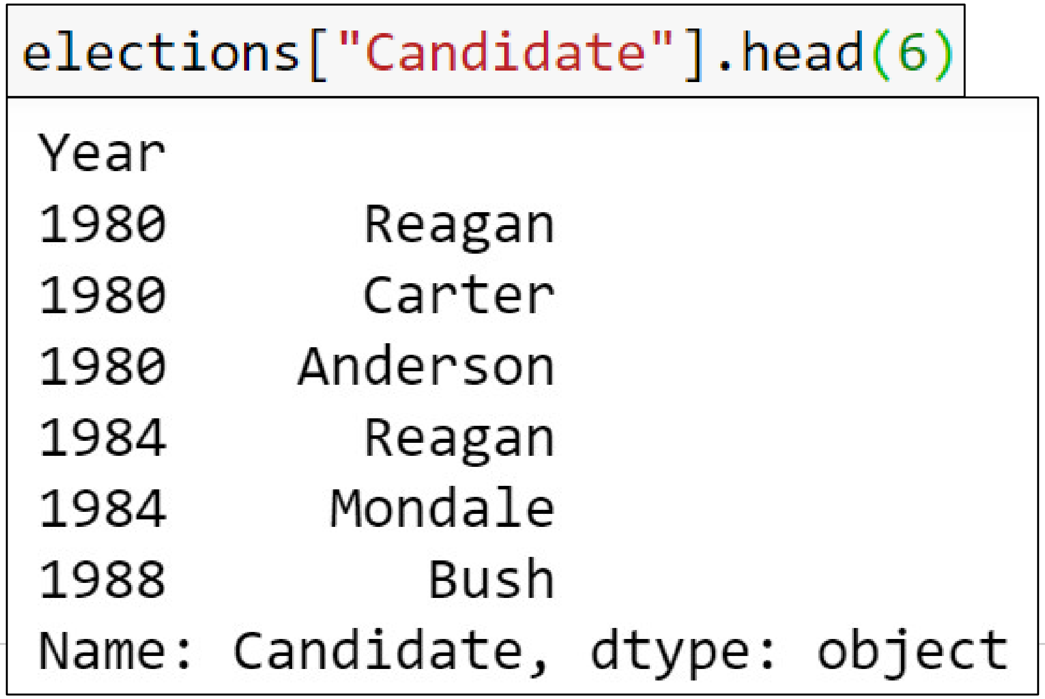
	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Indexing with The [] Operator

Indexing by Column Names Using [] Operator

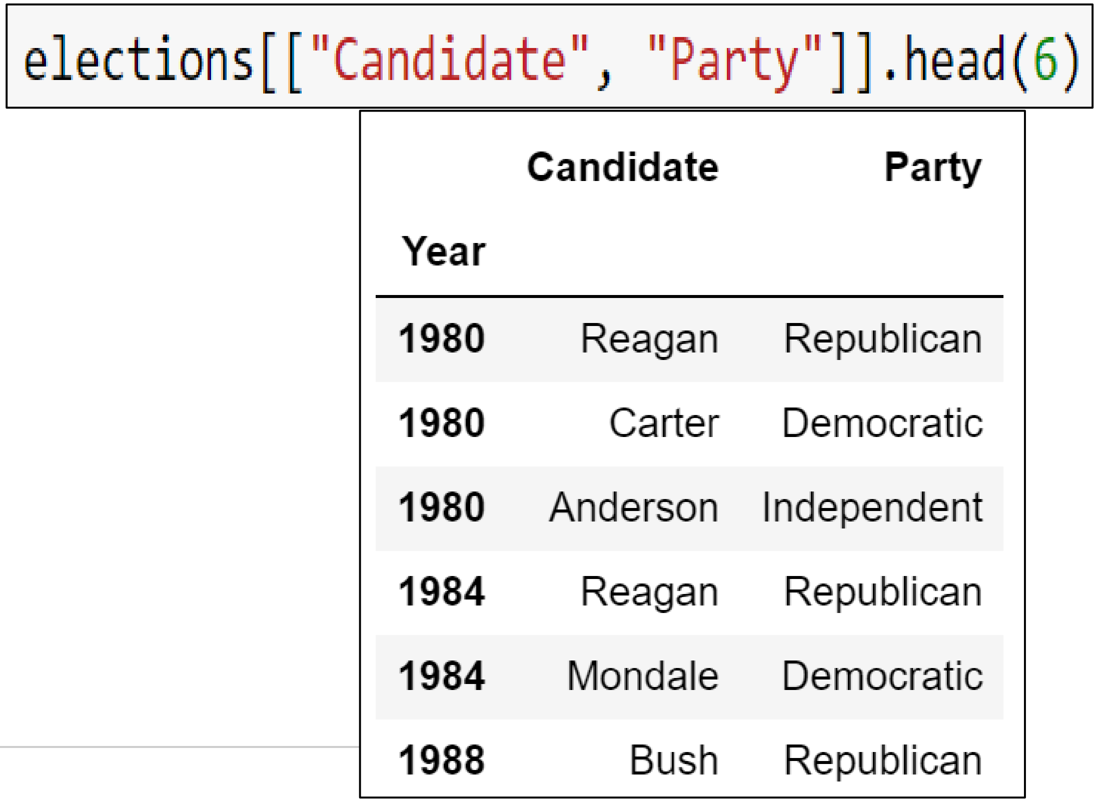
Given a dataframe, it is common to extract a Series or a collection of Series. This process is also known as “Column Selection” or sometimes “indexing by column”.

- Column name argument to [] yields Series.
- List argument to [] yields a Data Frame.



```
elections["Candidate"].head(6)
```

```
Year
1980        Reagan
1980        Carter
1980      Anderson
1984        Reagan
1984       Mondale
1988          Bush
Name: Candidate, dtype: object
```



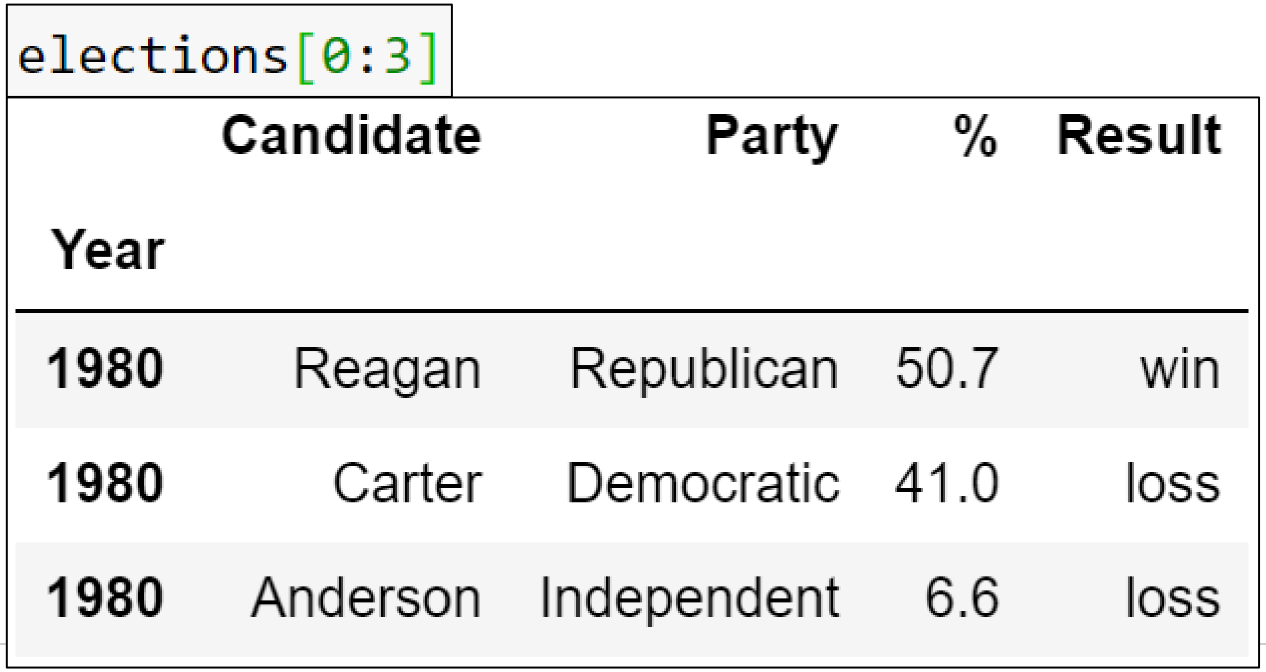
```
elections[["Candidate", "Party"]].head(6)
```

	Candidate	Party
Year		
1980	Reagan	Republican
1980	Carter	Democratic
1980	Anderson	Independent
1984	Reagan	Republican
1984	Mondale	Democratic
1988	Bush	Republican

Indexing by Row Slices Using [] Operator

We can also index by row numbers using the [] operator.

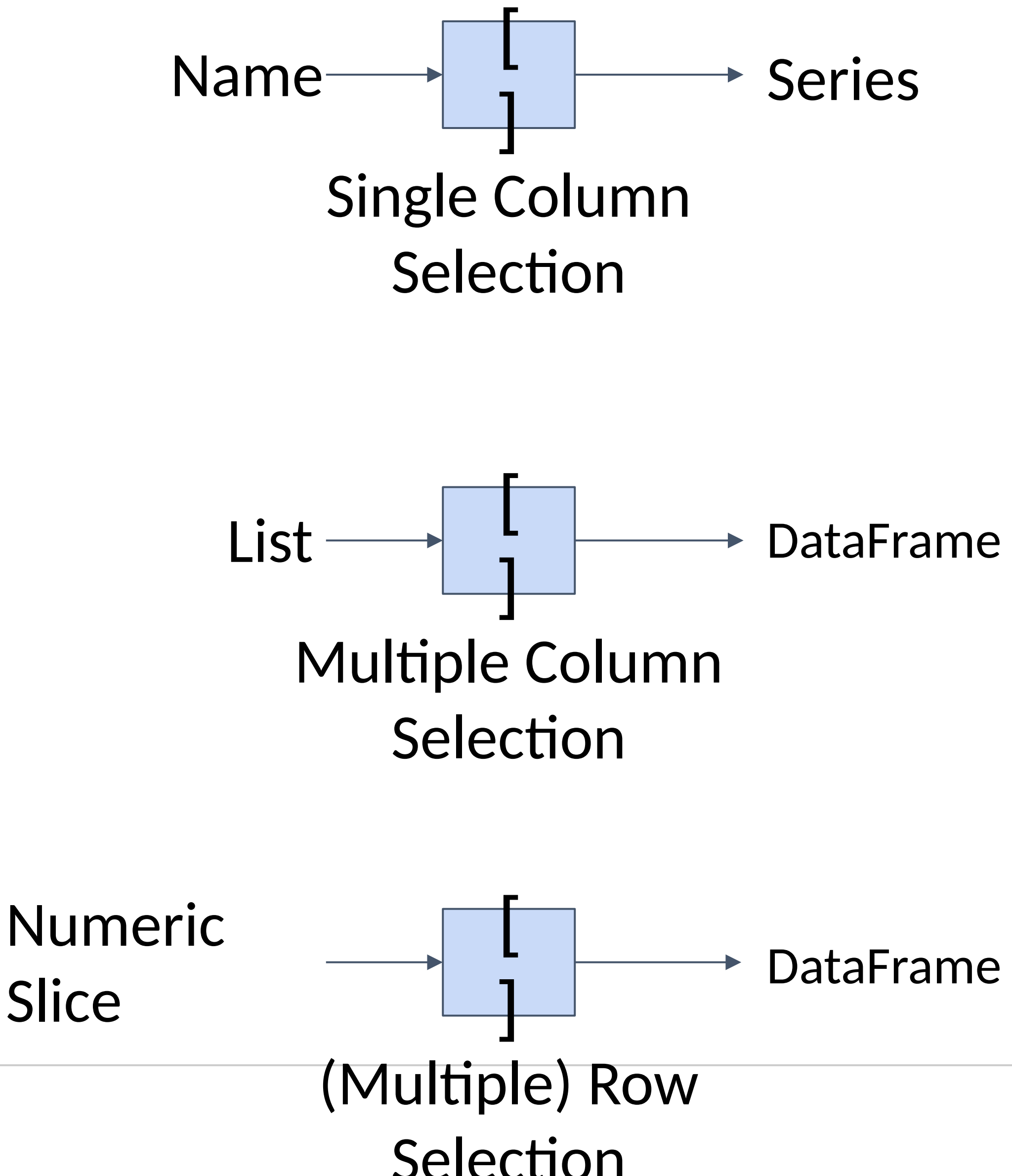
- Numeric slice argument to [] yields rows.
- Example: [0:3] yields rows 0 to 2.

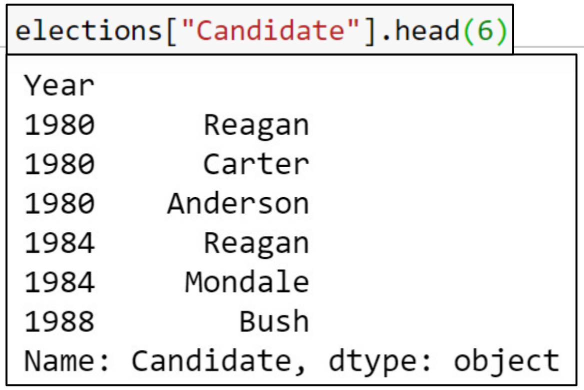


```
elections[0:3]
```

	Candidate	Party	%	Result
Year				
1980	Reagan	Republican	50.7	win
1980	Carter	Democratic	41.0	loss
1980	Anderson	Independent	6.6	loss

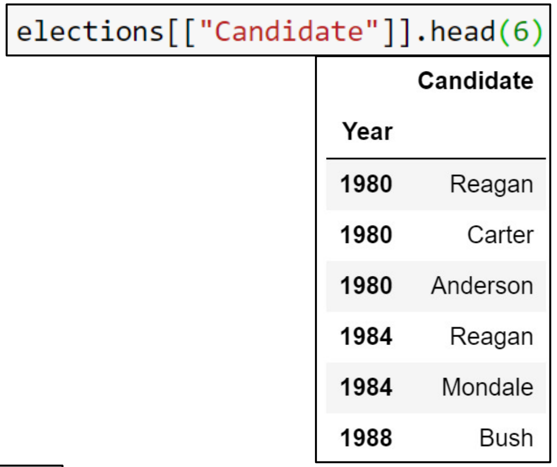
[] Summary





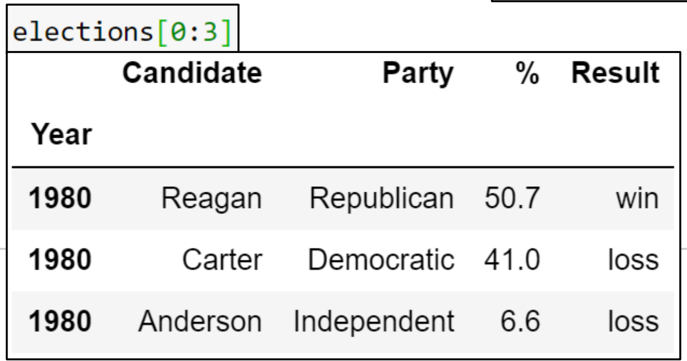
```
elections["Candidate"].head(6)
```

```
Year
1980        Reagan
1980        Carter
1980      Anderson
1984        Reagan
1984       Mondale
1988          Bush
Name: Candidate, dtype: object
```



```
elections[["Candidate"]].head(6)
```

	Candidate
Year	
1980	Reagan
1980	Carter
1980	Anderson
1984	Reagan
1984	Mondale
1988	Bush

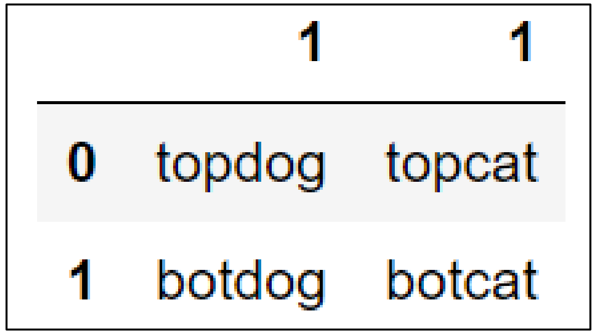


```
elections[0:3]
```

	Candidate	Party	%	Result
Year				
1980	Reagan	Republican	50.7	win
1980	Carter	Democratic	41.0	loss
1980	Anderson	Independent	6.6	loss

Question

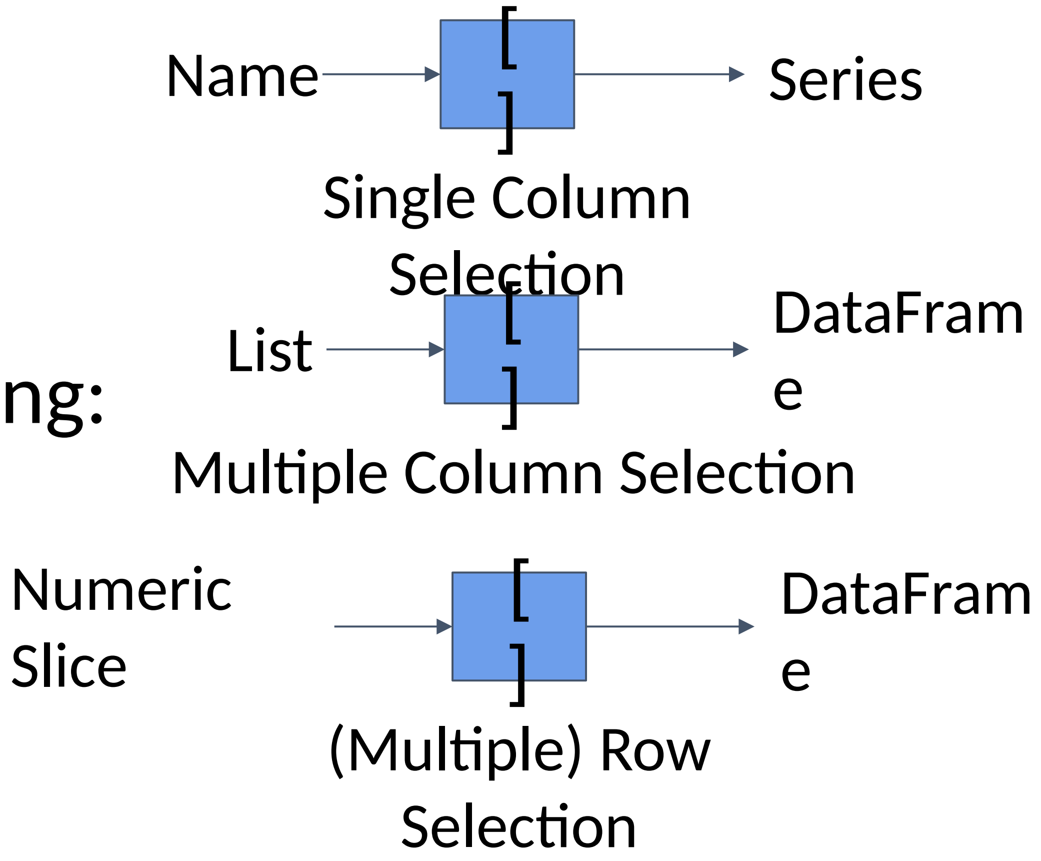
```
weird = pd.DataFrame({1:["topdog","botdog"], "1":["topcat","botcat"]})
weird
```



	1	1
0	topdog	topcat
1	botdog	botcat

Try to predict the output of the following:

- `weird[1]`
- `weird[“1”]`
- `weird[1:]`



Boolean Array Selection and Querying

Boolean Array Input

Yet another input type supported by [] is the boolean array.

Entry number 7

```
elections[[False, False, False, False, False,
            False, False, True, False, False,
            True, False, False, False, True,
            False, False, False, False, False,
            False, False, True]]
```

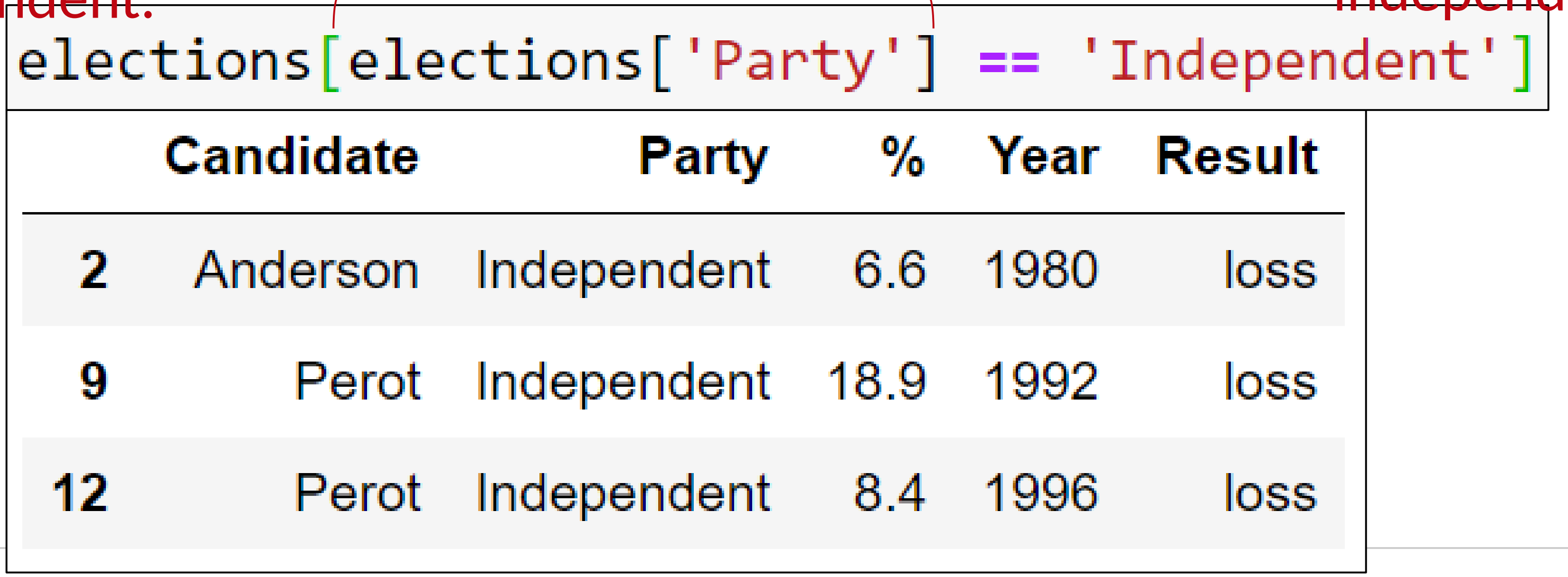
	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win

Boolean Array Input

Yet another input type supported by [] is the boolean array. Useful because boolean arrays can be generated by using logical operators on Series.

Length 23 Series where every entry is "Republican", "Democrat" or "Independent."

Length 23 Series where every entry is either "True" or "False", where "True" occurs for every independent candidate.

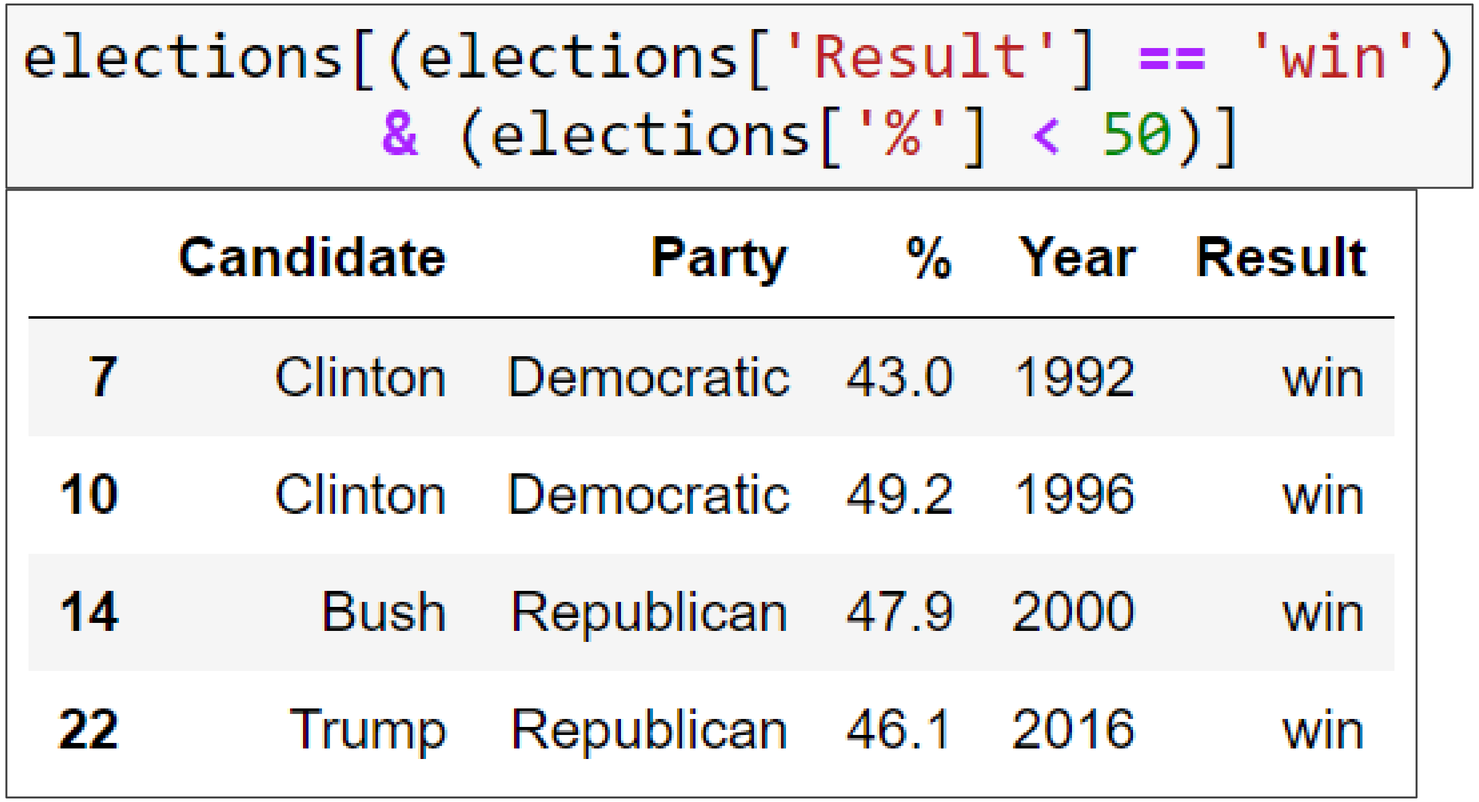


```
elections[elections['Party'] == 'Independent']
```

	Candidate	Party	%	Year	Result
2	Anderson	Independent	6.6	1980	loss
9	Perot	Independent	18.9	1992	loss
12	Perot	Independent	8.4	1996	loss

Boolean Array Input

Boolean Series can be combined using the & operator, allowing filtering of results by multiple criteria.



```
elections[(elections['Result'] == 'win')
          & (elections['%'] < 50)]
```

	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win

`isin`

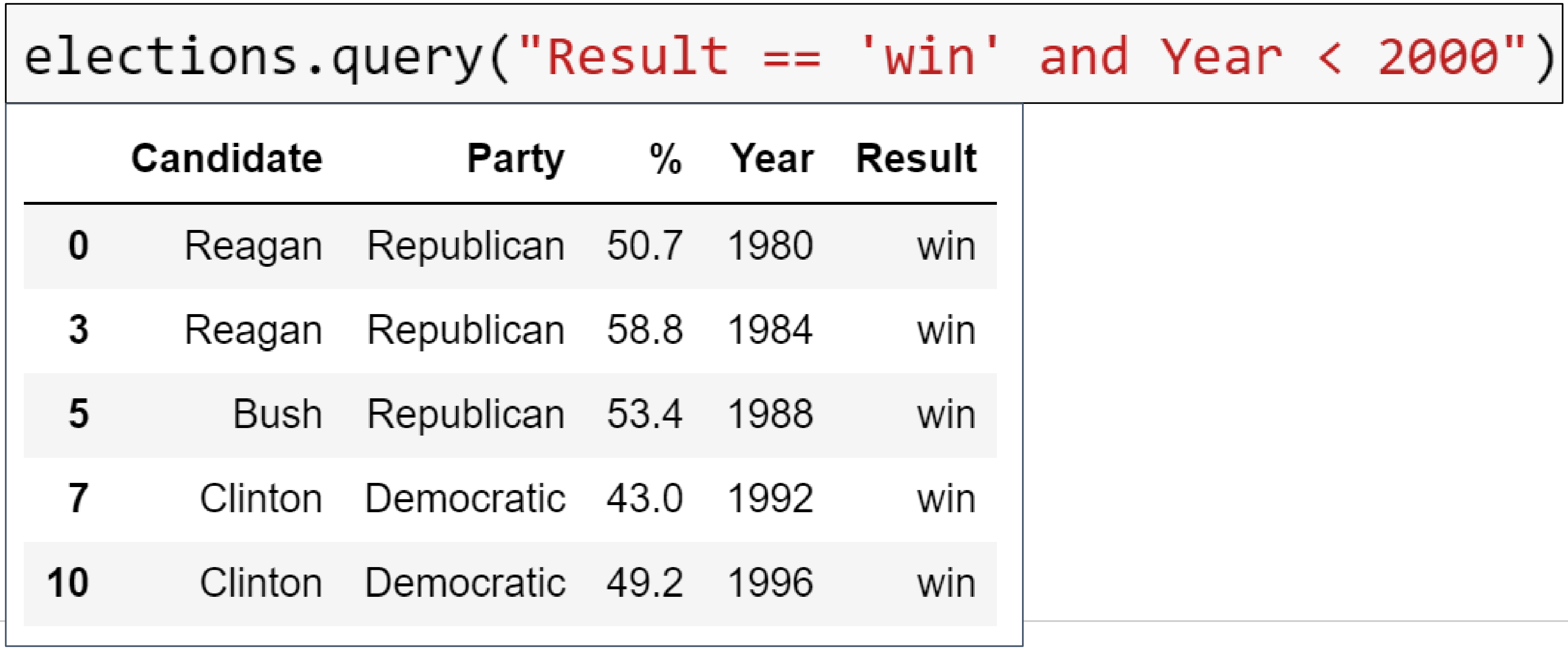
The `isin` function makes it more convenient to find rows that match one of many possible values.

Example: Suppose we want to find “Republican” or “Democratic” candidates. Could use the | operator (| means or), or we can use `isin`.

- Ugly:
```
df[(df["Party"] == "Democratic") | (df["Party"] == "Republican")]
```
- Better:
```
df[df["Party"].isin(["Republican", "Democratic"])]
```

The Query Command

The query command provides an alternate way to combine multiple conditions.



```
elections.query("Result == 'win' and Year < 2000")
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
3	Reagan	Republican	58.8	1984	win
5	Bush	Republican	53.4	1988	win
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win

Indexing with .loc and .iloc
Sampling with .sample

Loc and iloc

Loc and iloc are alternate ways to index into a DataFrame.

- They take a lot of getting used to! Documentation and ideas behind them are quite complex.
- I'll go over common usages (see docs for weirder ones).

Documentation:

- loc: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html
- iloc: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iloc.html
- More general docs on indexing and selecting: Link

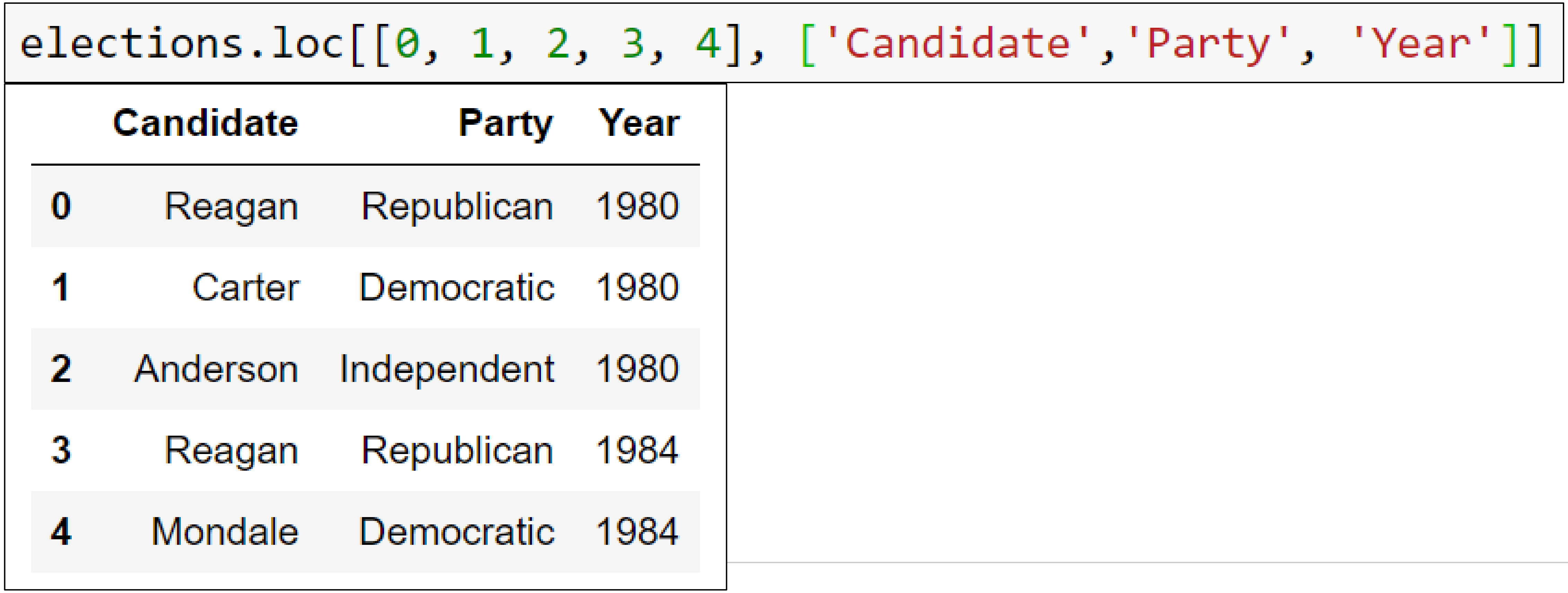
Loc

Loc does two things:

- Access values by labels.
- Access values using a boolean array (a la Boolean Array Selection).

Loc with Lists

The most basic use of loc is to provide a list of row and column labels, which returns a DataFrame.

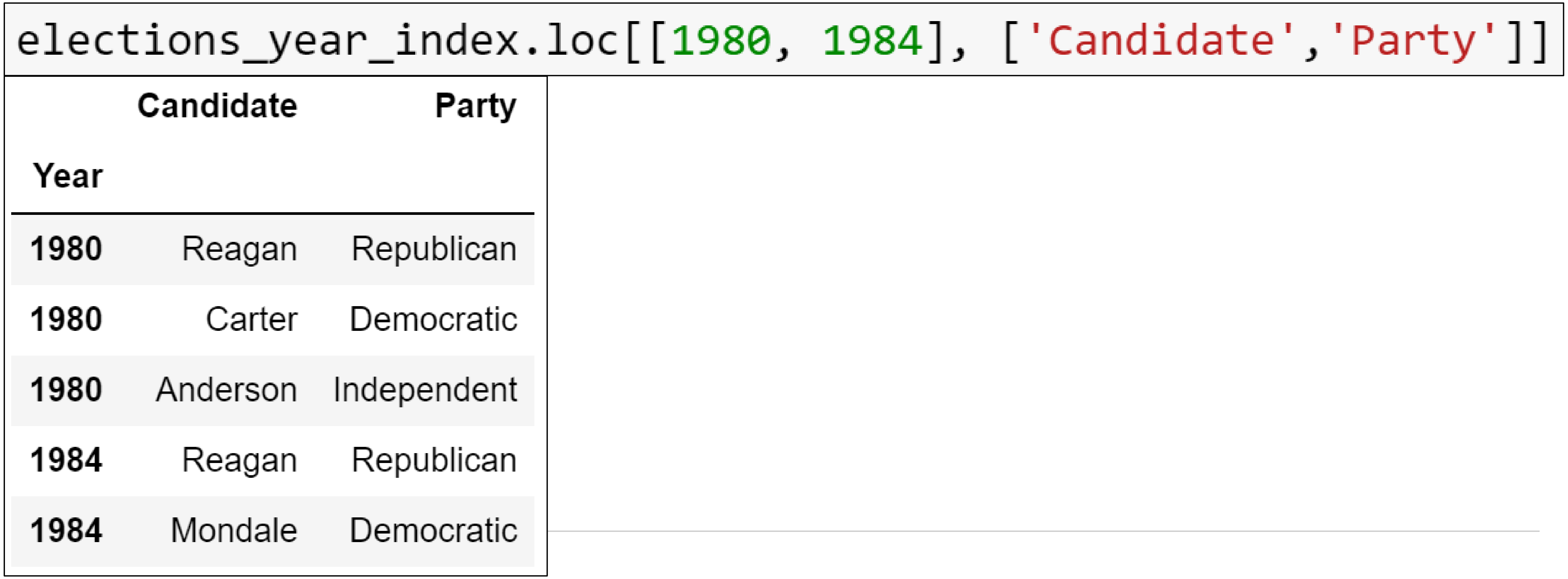


```python
elections.loc[[0, 1, 2, 3, 4], ['Candidate','Party', 'Year']]
```

	Candidate	Party	Year
0	Reagan	Republican	1980
1	Carter	Democratic	1980
2	Anderson	Independent	1980
3	Reagan	Republican	1984
4	Mondale	Democratic	1984

Loc with Lists

The most basic use of loc is to provide a list of row and column labels, which returns a DataFrame.



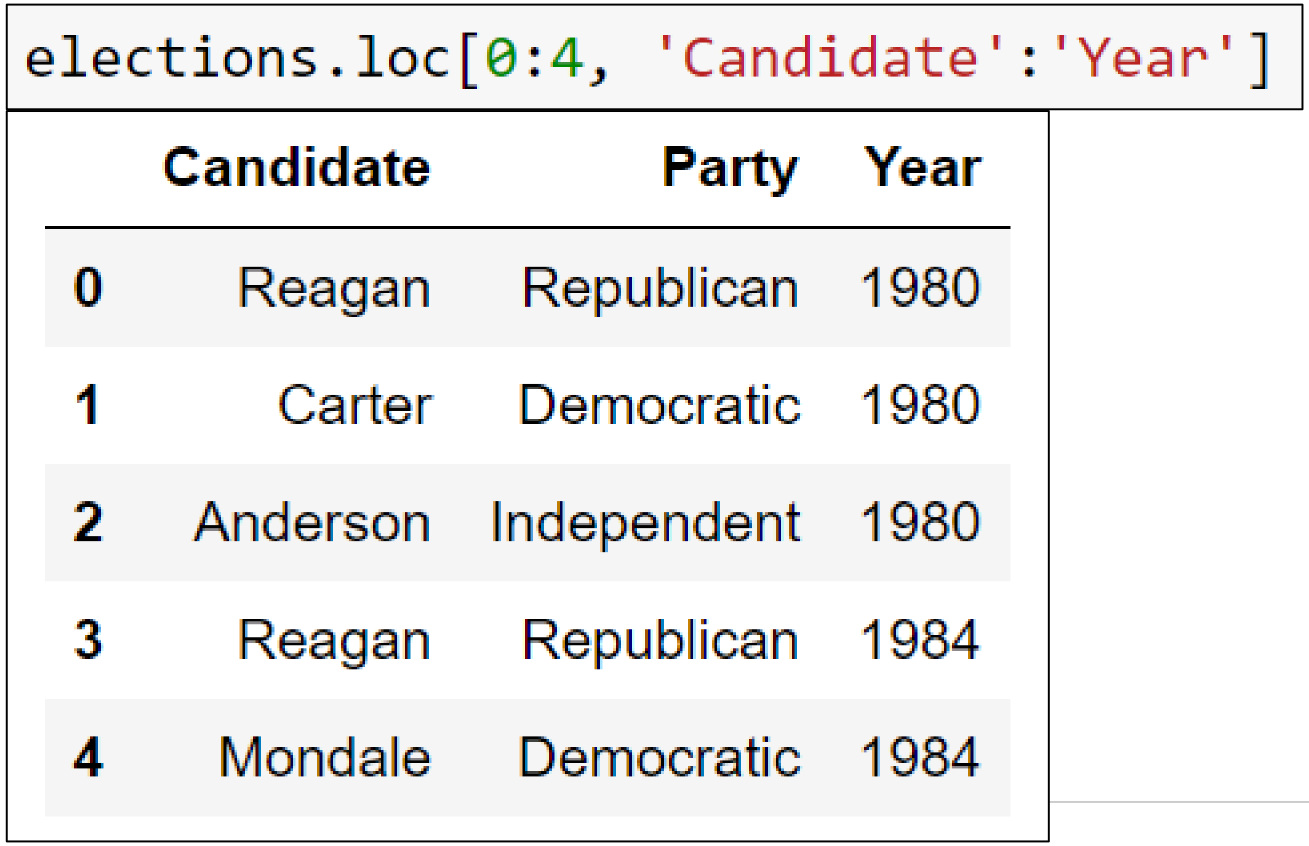
```
elections_year_index.loc[[1980, 1984], ['Candidate','Party']]
```

	Candidate	Party
Year		
1980	Reagan	Republican
1980	Carter	Democratic
1980	Anderson	Independent
1984	Reagan	Republican
1984	Mondale	Democratic

Loc with Slices

Loc is also commonly used with slices.

- Slicing works with all label types, not just numeric labels.
- Slices with loc are **inclusive**, not **exclusive.**



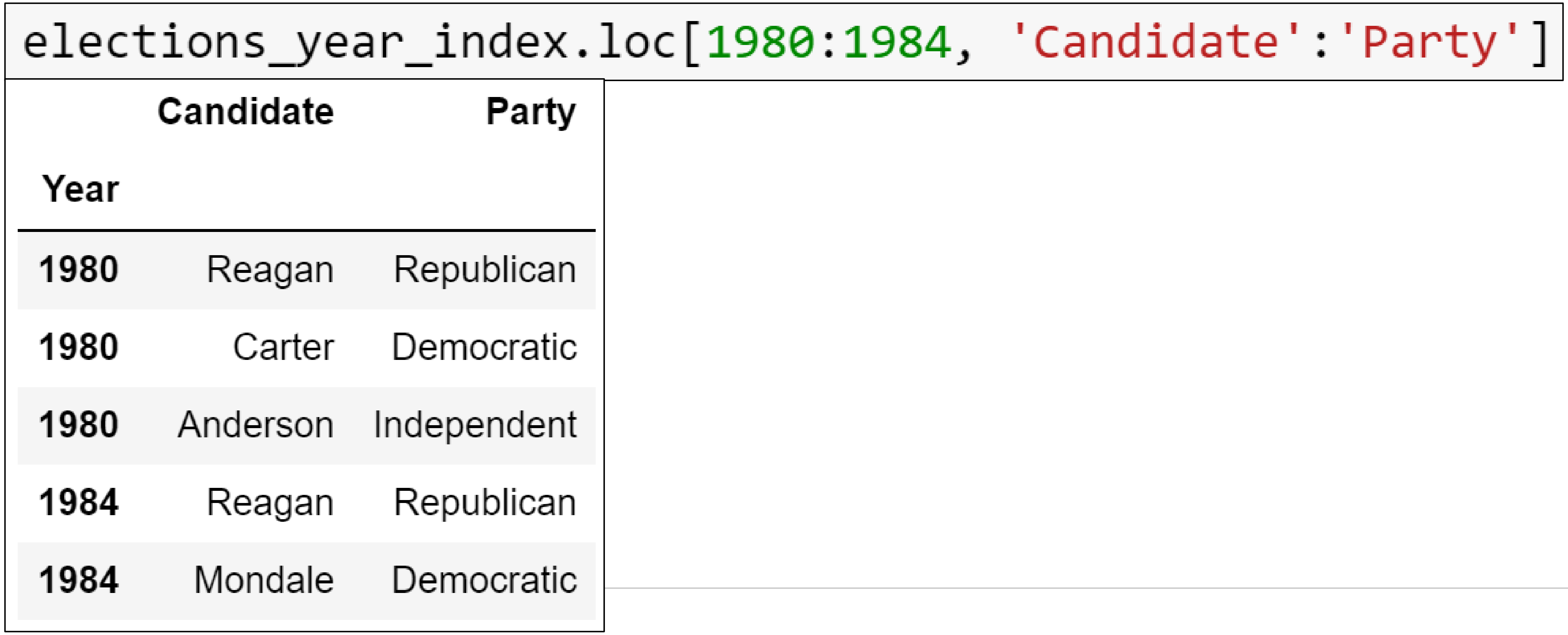
```
elections.loc[0:4, 'Candidate':'Year']
```

	Candidate	Party	Year
0	Reagan	Republican	1980
1	Carter	Democratic	1980
2	Anderson	Independent	1980
3	Reagan	Republican	1984
4	Mondale	Democratic	1984

Loc with Slices

Loc is also commonly used with slices.

- Slicing works with all label types, not just numeric labels.
- Slices with loc are **inclusive**, not **exclusive**.

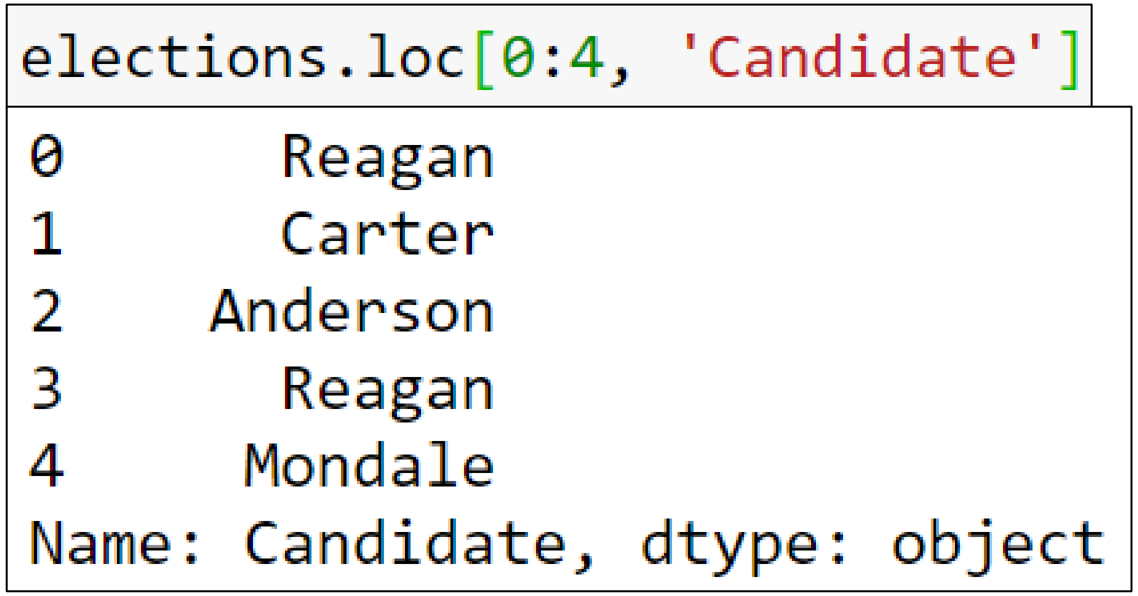


```
elections_year_index.loc[1980:1984, 'Candidate':'Party']
```

	Candidate	Party
Year		
1980	Reagan	Republican
1980	Carter	Democratic
1980	Anderson	Independent
1984	Reagan	Republican
1984	Mondale	Democratic

Loc with Single Values for Column Label

If we provide only a single label as column argument, we get a Series.

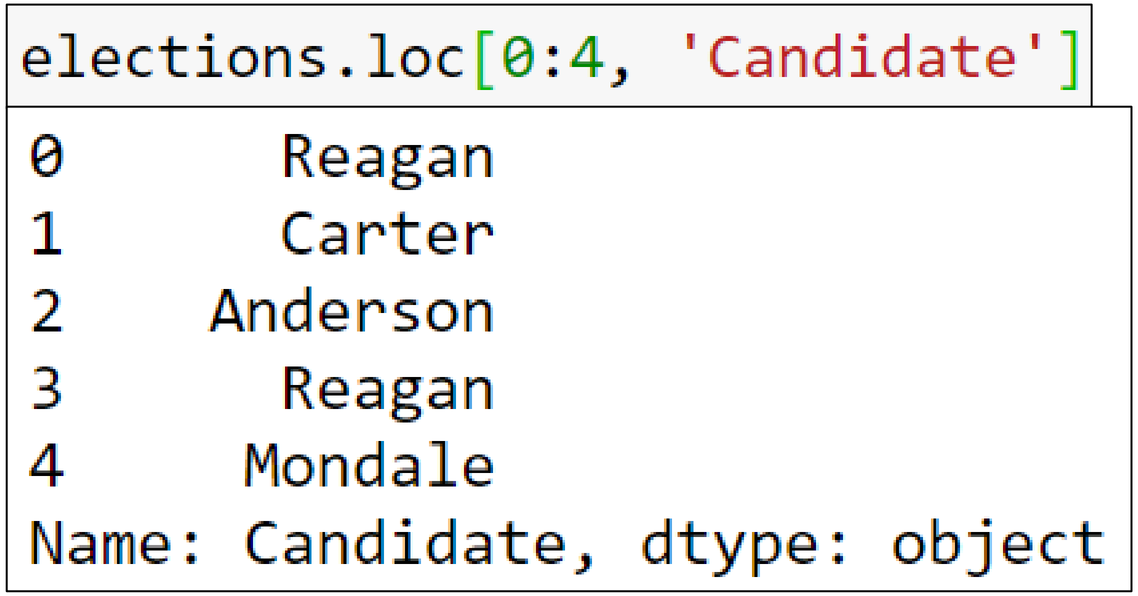


```
elections.loc[0:4, 'Candidate']
```

```
0       Reagan
1       Carter
2     Anderson
3       Reagan
4      Mondale
Name: Candidate, dtype: object
```

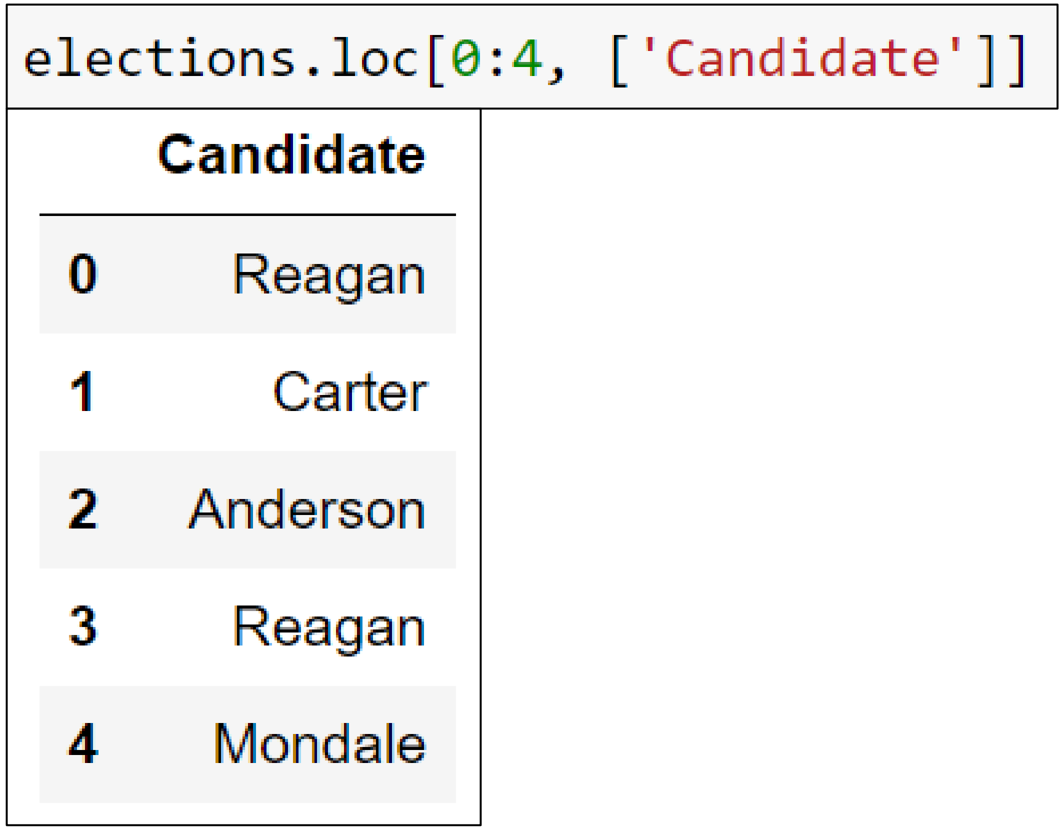
Loc with Single Values for Column Label

As before with the [] operator, if we provide a list of only one label as an argument, we get back a dataframe.



```
elections.loc[0:4, 'Candidate']
```

```
0      Reagan
1      Carter
2    Anderson
3      Reagan
4     Mondale
Name: Candidate, dtype: object
```



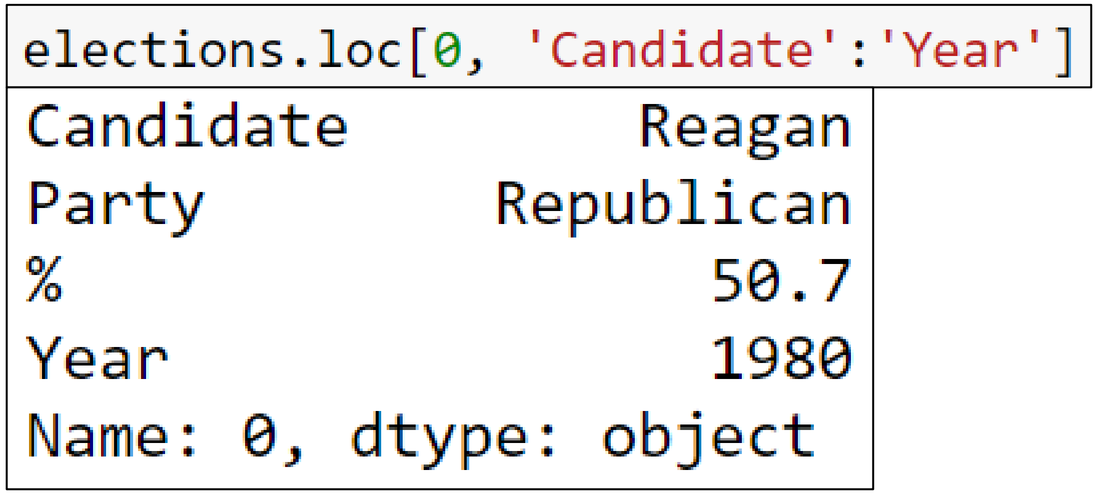
```
elections.loc[0:4, ['Candidate']]
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale

Loc with Single Values for Row Label

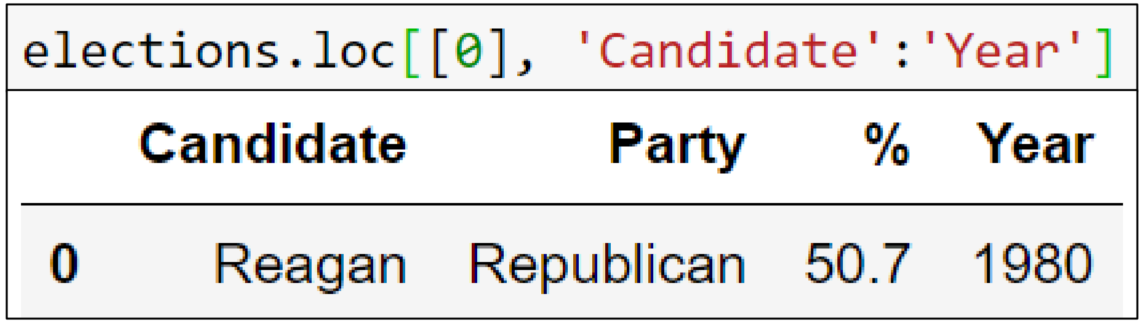
If we provide only a single row label, we get a Series.

- Such a series represents a ROW not a column!
- The index of this Series is the names of the columns from the data frame.
- Putting the single row label in a list yields a dataframe version.



```
elections.loc[0, 'Candidate':'Year']
```

```
Candidate        Reagan
Party        Republican
%                  50.7
Year               1980
Name: 0, dtype: object
```

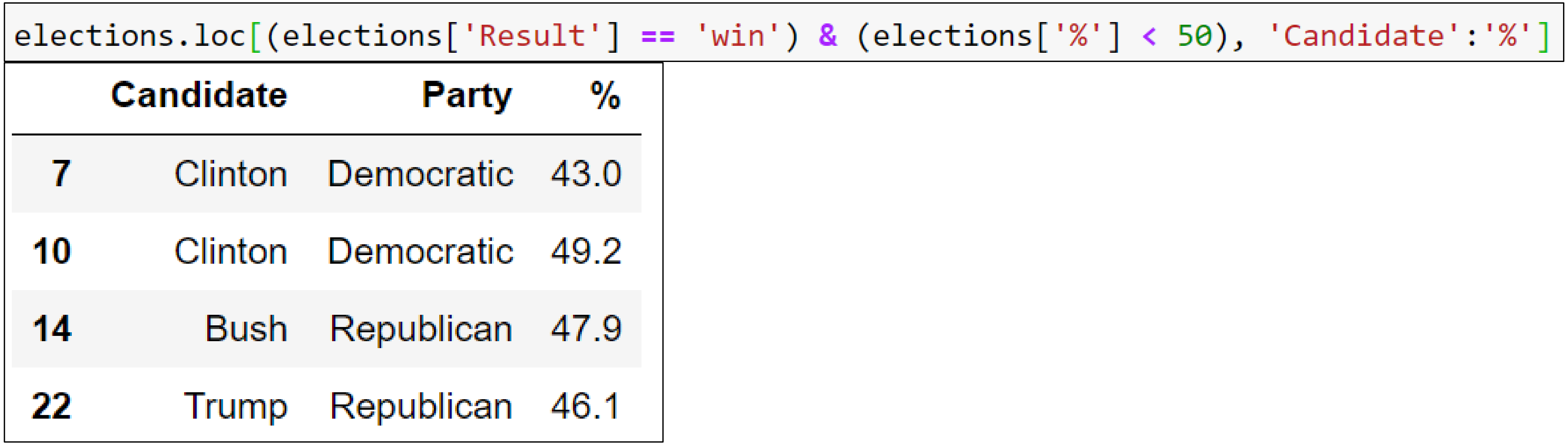


```
elections.loc[[0], 'Candidate':'Year']
```

	Candidate	Party	%	Year
0	Reagan	Republican	50.7	1980

Loc Supports Boolean Arrays

Loc supports Boolean Arrays exactly as you'd expect.

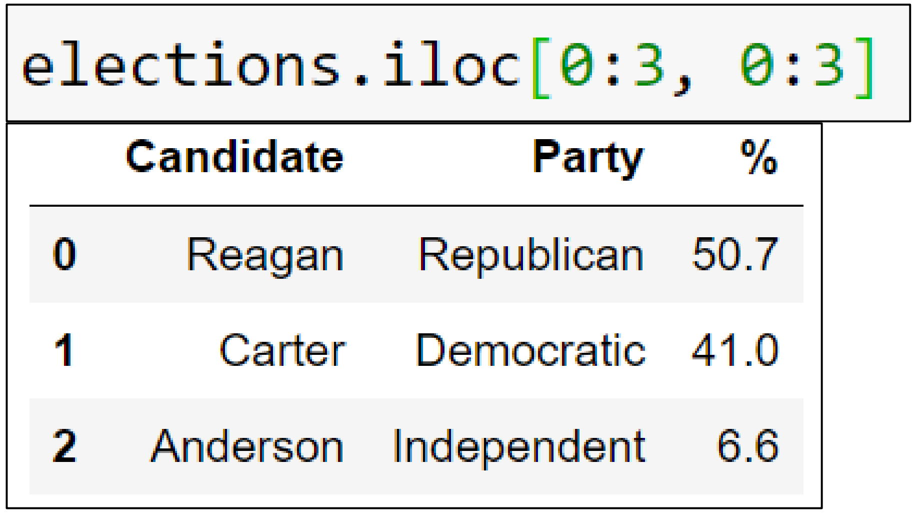


```python
elections.loc[(elections['Result'] == 'win') & (elections['%'] < 50), 'Candidate':'%']
```

	Candidate	Party	%
7	Clinton	Democratic	43.0
10	Clinton	Democratic	49.2
14	Bush	Republican	47.9
22	Trump	Republican	46.1

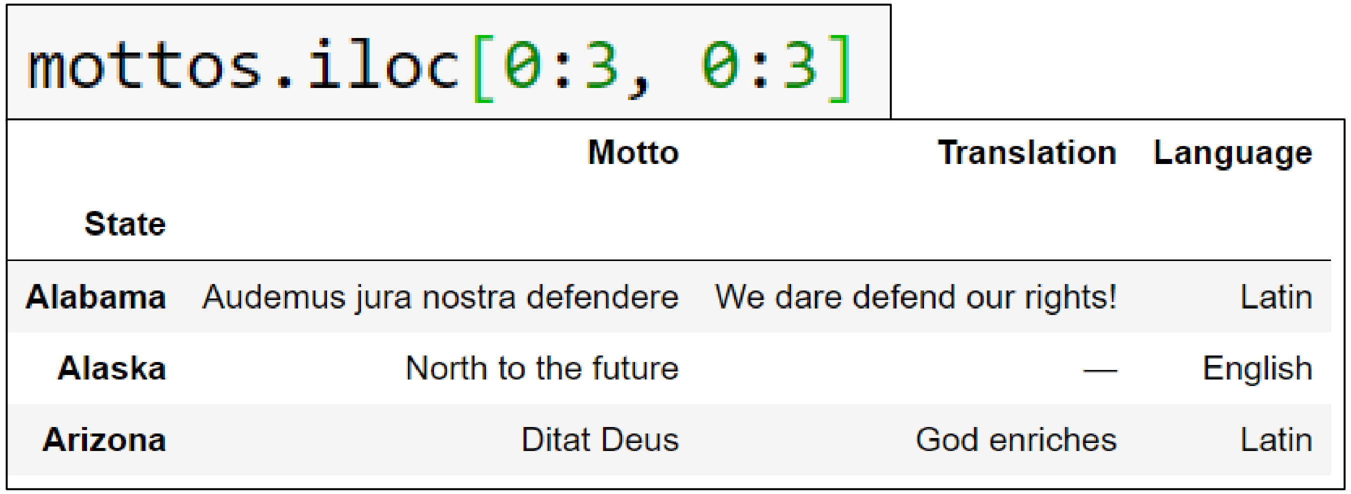
iloc: Integer-Based Indexing for Selection by Position

In contrast to loc, iloc doesn’t think about labels at all. Instead, it returns the items that appear in the numerical positions specified.



```
elections.iloc[0:3, 0:3]
```

	Candidate	Party	%
0	Reagan	Republican	50.7
1	Carter	Democratic	41.0
2	Anderson	Independent	6.6



```
mottos.iloc[0:3, 0:3]
```

	Motto	Translation	Language
State			
Alabama	Audemus jura nostra defendere	We dare defend our rights!	Latin
Alaska	North to the future	—	English
Arizona	Ditat Deus	God enriches	Latin

Advantages of loc:

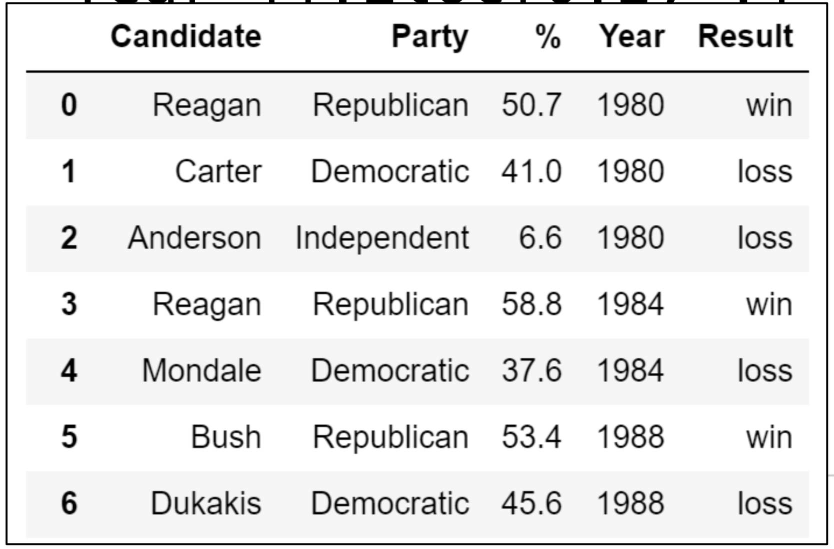
- Harder to make mistakes.
- Easier to read code.
- Not vulnerable to changes to the ordering of rows/cols in raw data files.

Nonetheless, iloc can be more convenient. *Use iloc judiciously*.

Annoying Question Challenge

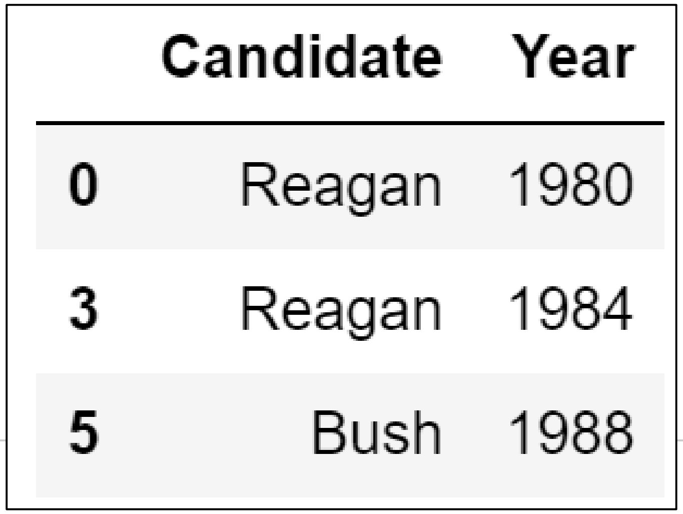
Which of the following pandas statements returns a DataFrame of the first 3 Candidate names only for candidates that won with more than 50% of the vote.

```
elections.iloc[[0, 3, 5], [0, 3]]
elections.loc[[0, 3, 5], ["Candidate":"Year"]
elections.loc[elections["%"] > 50, ["Candidate", "Year"]].head(3)
elections.loc[elections["%"] > 50, ["Candidate",
"Year"]].iloc[0:2, :]
```



	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss

$\longrightarrow$



	Candidate	Year
0	Reagan	1980
3	Reagan	1984
5	Bush	1988

Annoying Question Challenge

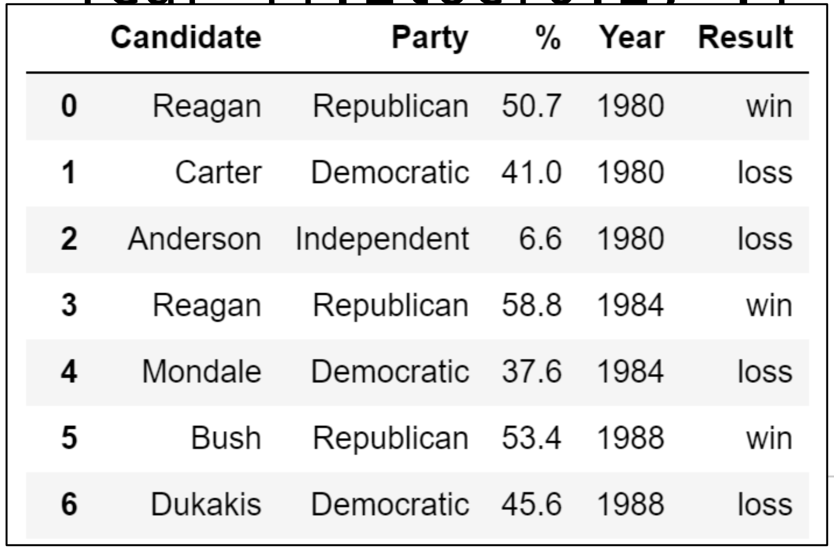
Which of the following pandas statements returns a DataFrame of the first 3 Candidate names only for candidates that won with more than 50% of the vote.

`elections.iloc[[0, 3, 5], [0, 3]]`

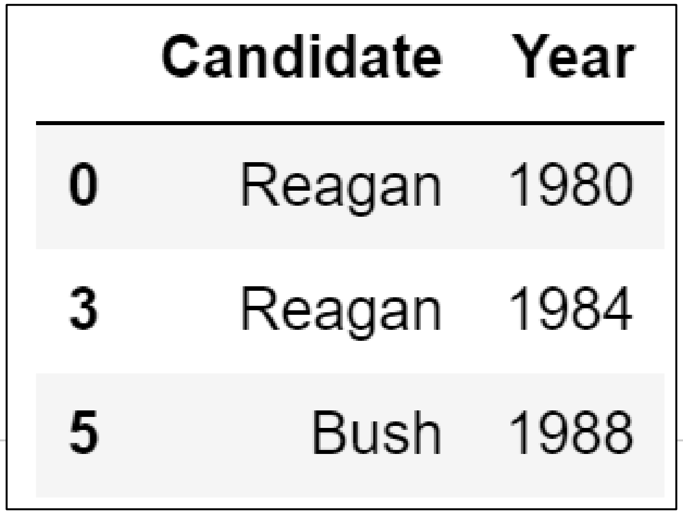
`elections.loc[[0, 3, 5], ["Candidate":"Year"]`

`elections.loc[elections["%"] > 50, ["Candidate", "Year"]].head(3)`

`elections.loc[elections["%"] > 50, ["Candidate", "Year"]].iloc[0:2, :]`



	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss



	Candidate	Year
0	Reagan	1980
3	Reagan	1984
5	Bush	1988

See notebook for why!

Sample

If you want a DataFrame consisting of a random selection of rows, you can use the sample method.

- By default, *it is without replacement*. Use `replace=true` for replacement.
- Naturally, can be chained with our selection operators [], loc, iloc.

```
elections.sample(10)
```

	Candidate	Party	%	Year	Result
15	Kerry	Democratic	48.3	2004	loss
16	Bush	Republican	50.7	2004	win
22	Trump	Republican	46.1	2016	win
9	Perot	Independent	18.9	1992	loss
21	Clinton	Democratic	48.2	2016	loss
11	Dole	Republican	40.7	1996	loss
20	Romney	Republican	47.2	2012	loss
14	Bush	Republican	47.9	2000	win
8	Bush	Republican	37.4	1992	loss
1	Carter	Democratic	41.0	1980	loss

```
elections.query("Year < 1992").sample(4, replace=True)
```

	Candidate	Party	%	Year	Result
1	Carter	Democratic	41.0	1980	loss
4	Mondale	Democratic	37.6	1984	loss
6	Dukakis	Democratic	45.6	1988	loss
1	Carter	Democratic	41.0	1980	loss

Handy Properties and Utility Functions for Series and DataFrames

Numpy Operations

Pandas Series and DataFrames support a large number of operations, including mathematical operations so long as the data is numerical.

```
winners = elections.query("Result == 'win'")["%"]
winners
```

```
0     50.7
3     58.8
5     53.4
7     43.0
10    49.2
14    47.9
16    50.7
17    52.9
19    51.1
22    46.1
Name: %, dtype: float64
```

```
np.mean(winners)
```

```
50.38
```

```
max(winners)
```

```
58.8
```

`head`, `size`, `shape`, and `describe`

`head`: Displays only the top few rows.

`size`: Gives the total number of data points.

`shape`: Gives the size of the data in rows and columns.

`describe`: Provides a summary of the data.

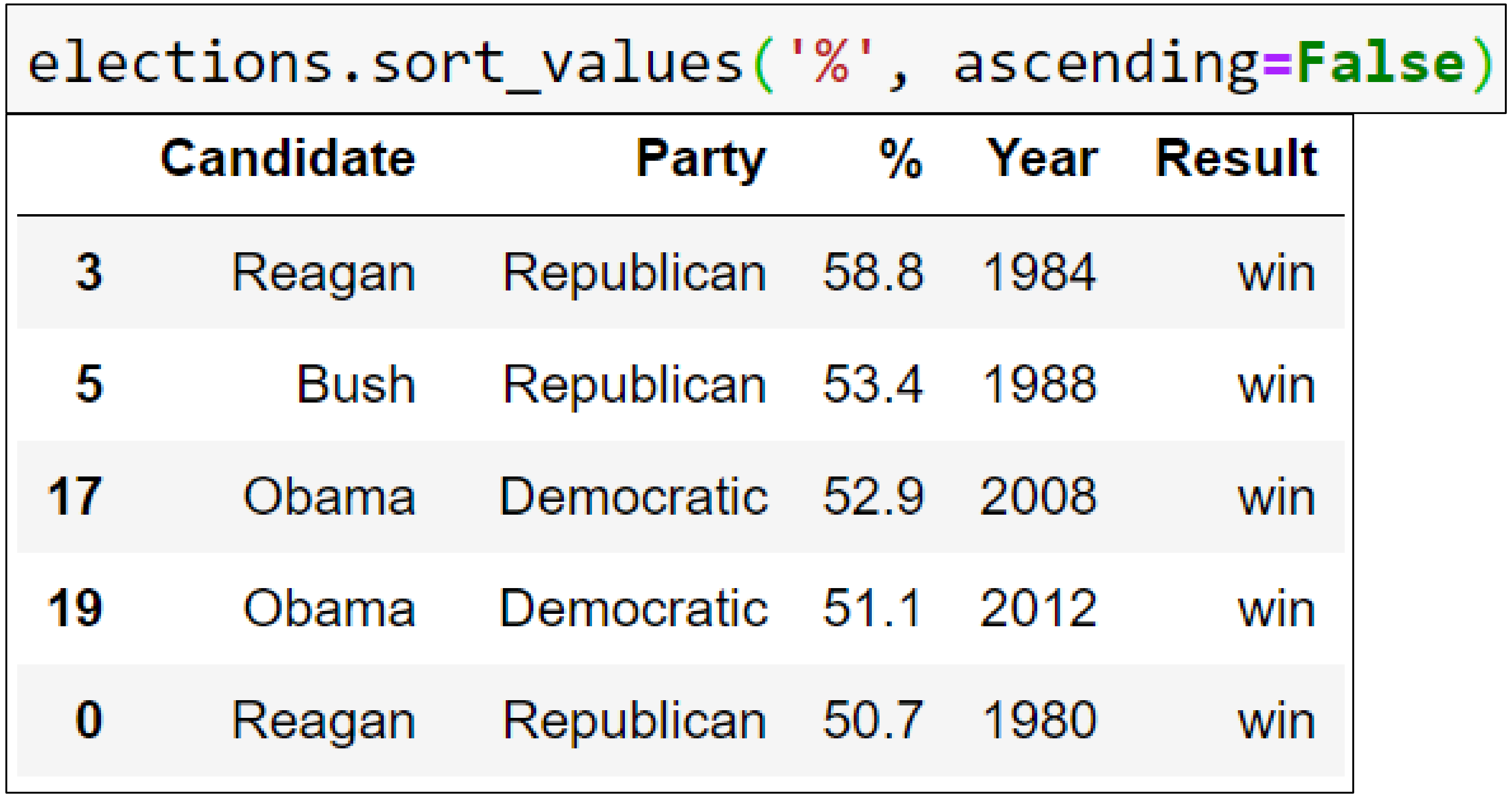
`index` and `columns`

`index`: Returns the index (a.k.a. row labels).

`columns`: Returns the labels for the columns.

The `sort_values` Method

One incredibly useful method for DataFrames is `sort_values`, which creates a copy of a DataFrame sorted by a specific column.

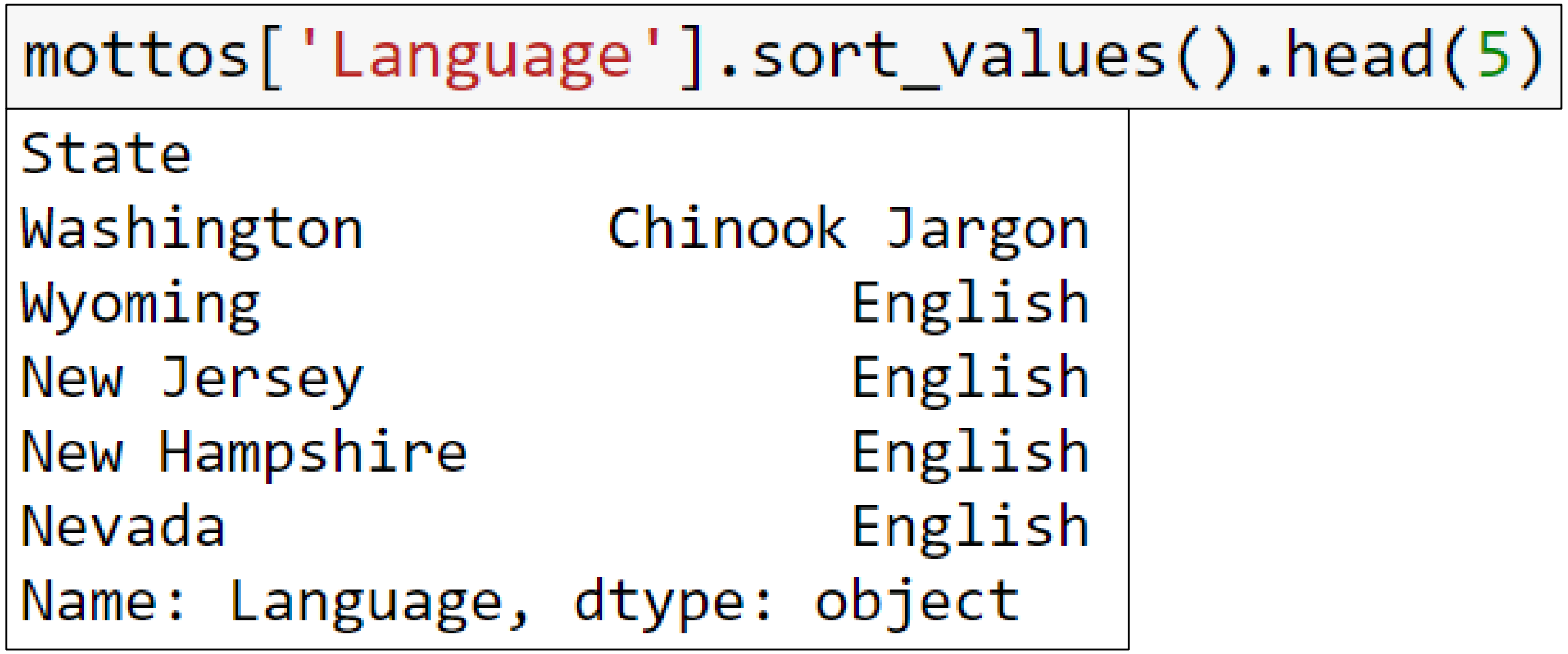


```python
elections.sort_values('%', ascending=False)
```

	Candidate	Party	%	Year	Result
3	Reagan	Republican	58.8	1984	win
5	Bush	Republican	53.4	1988	win
17	Obama	Democratic	52.9	2008	win
19	Obama	Democratic	51.1	2012	win
0	Reagan	Republican	50.7	1980	win

The `sort_values` Method

We can also use `sort_values` on a Series, which returns a copy with with the values in order.

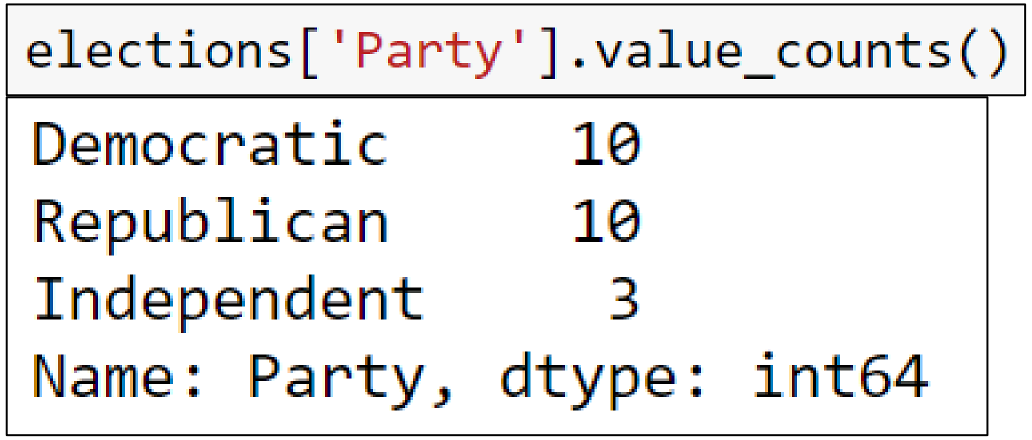


```python
mottos['Language'].sort_values().head(5)
```

```
State
Washington       Chinook Jargon
Wyoming                 English
New Jersey              English
New Hampshire           English
Nevada                  English
Name: Language, dtype: object
```

The `value_counts` Method

Series also has the function `value_counts`, which creates a new Series showing the counts of every value.

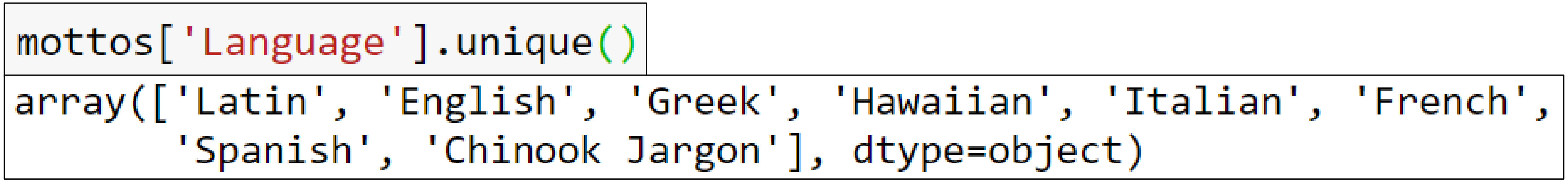


```
elections['Party'].value_counts()
```

```
Democratic     10
Republican     10
Independent     3
Name: Party, dtype: int64
```

The `unique` Method

Another handy method for Series is `unique`, which returns all unique values as an array.



```
mottos['Language'].unique()
```

```
array(['Latin', 'English', 'Greek', 'Hawaiian', 'Italian', 'French',
       'Spanish', 'Chinook Jargon'], dtype=object)
```

The Things We Just Saw

- sort_values
- value_counts
- unique

Summary

- Operations on String series, e.g. babynames[“Name”].str.startswith()
- Creating and dropping columns.
    - Creating temporary columns is often convenient for sorting.
- Passing an index as an argument to loc.
    - Useful as an alternate way to sort a dataframe.