

COMS BC1016

Introduction to Computational Thinking and Data Science

Lecture 8: Functions, Groups, Pivots, and Joins

Sep 3, 2025

Copyright © 2026 Barnard College

February 16, 2026

Logistics

- HW 2 is due Wednesday at 11:59pm
 - Please remember to submit it as a **.ipynb** file
 - If you are having trouble, please come to one of the office hours (mine, TA, or computing fellow)

Last time: Charts

Charts Summary

Type	Syntax	Description
Line graph	<code>.plot(x_axis, y_axis)</code>	Sequential numerical data
Scatter Plot	<code>.scatter(x_axis, y_axis)</code>	Relation between two numerical values
Bar Chart	<code>.barh(column_label)</code>	Distribution of one categorical variable (already grouped)
Histogram	<code>.hist(column_label, unit, bins)</code>	Distribution of one numerical variable

Chart Selection Exercise

We have NYC weather data from 2019 as shown below (from [Kaggle](#))

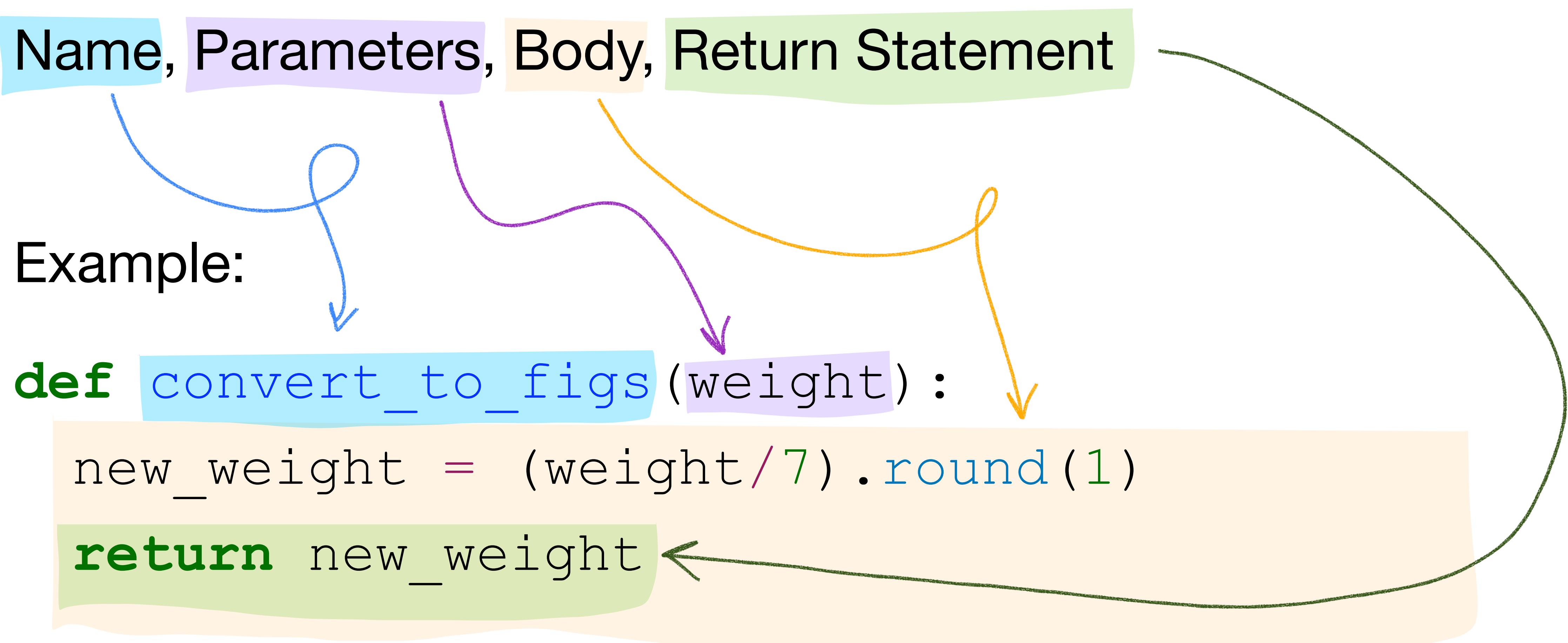
Which type of chart (line, scatter, bar, histogram) would best help you answer to each question?

- Do days with hotter highs also tend to have hotter lows?
- How do the number of rainy days compare with the number of snowy days?
- What percent of days have a high of at least 75 degrees?

date	tmax	tmin	tavg	condition
1/1/19	60	40	50	rainy
2/1/19	41	35	38	
3/1/19	45	39	42	
4/1/19	47	37	42	
5/1/19	47	42	44.5	rainy
6/1/19	49	32	40.5	
7/1/19	35	26	30.5	
8/1/19	47	35	41	rainy
9/1/19	46	35	40.5	rainy
10/1/19	35	30	32.5	

Functions

Recall: Anatomy of a Function



Functions with Multiple Arguments/Parameters

Functions can take in multiple inputs

- Each argument is given a unique name and separated by commas

```
def convert_to_figs(weight, decimal_places):  
    '''Divides the input by 7 (Figs weight) and then rounds to  
    the given number of decimal places'''  
    new_weight = (weight/7).round(decimal_places)  
    return new_weight
```

Functions with Multiple Arguments/Parameters

Functions can take in multiple inputs

- Each argument is given a unique name and separated by commas
- Specifying default values for particular inputs to makes them optional

```
def convert_to_figs(weight, decimal_places=1):  
    '''Divides the input by 7 (Figs weight) and then rounds to  
    the given number of decimal places'''  
    new_weight = (weight/7).round(decimal_places)  
  
    return new_weight
```

Function Demo

Recall: apply

Use **apply** to call a function on each element in a column

```
def convert_to_figs(weight):  
    new_weight = (weight/7).round(1)  
    return new_weight
```

```
cat_tbl.apply(convert_to_figs, 'Weight')
```

>Returns an array with convert_to_figs called on each element in the 'Weight' column

apply with Multiple Inputs

For functions with multiple inputs, `apply` can take multiple columns

```
def convert_to_figs(weight, decimal_places=1):  
    new_weight = (weight/7).round(decimal_places)  
  
    return new_weight  
  
cat_tbl.apply(convert_to_figs, 'Weight', 'Precision')
```

Apply Demo

Groups and Pivot Tables

Groups and Pivots

Two ways of summarizing table data by **categorical** variables

Name	Age	Weight	Coloring	Sex	Owner
Ruby	14	8	tuxedo	F	Alice
Gertrude	15	12	tuxedo	F	Alice
Hamby	8	16	tabby	M	Bob
Fig	3	7	tabby	F	Bob
Corina	6	10	tortie	F	Carol
Frito	2	8.5	tabby	M	Carol

Grouping by a Single Column

The `group` method aggregates all rows with the same value in column `c`

- `tbl.group(c)`
- `tbl.group(c, func)`

`group` can optionally apply `func` to grouped values, for example:

- `len`: count of grouped values (default)
- `list`: list of all grouped values
- `sum`: total of all grouped values

cat_tbl.group('Owner')	
Owner	count
Alice	2
Bob	2
Carol	2

cat_tbl.group('Owner', np.average)						
Owner	Name average	Age average	Weight average	Coloring average	Sex average	
Alice			14.5		10	
Bob			5.5		11.5	
Carol			4		9.25	

Grouping by Multiple Columns

The `group` method can also aggregate all rows that *share the combination of values* from multiple columns

cat_tbl.groupby(['Owner', 'Sex'])		
Owner	Sex	count
Alice	F	2
Bob	F	1
Bob	M	1
Carol	F	1
Carol	M	1

cat_tbl.groupby(['Sex', 'Coloring'], sum)					
Sex	Coloring	Name sum	Age sum	Weight sum	Owner sum
F	tabby		3	7	
F	tortie		6	10	
F	tuxedo		29	20	
M	tabby		10	24.5	

Pivot Tables

Cross-classifies according to *two* categorical variables

- Produces a grid of **all possible combinations** of the two categorical variables
- Grid entries are either counts or aggregated values

Create a pivot table where entries are counts:

```
tbl.pivot(col_var, row_var)
```

Create a pivot table where entries are aggregated according function
collect on values in column values

```
tbl.pivot(col_var, row_var, values, collect)
```

Pivot Tables

```
tbl.pivot(col_var, row_var)
```

- col_var: Variable that forms column labels of grid
- row_var: Variable that forms row labels of grid

```
cat_tbl.pivot('Owner', 'Sex')
```

Sex	Alice	Bob	Carol
F	2	1	1
M	0	1	1

Pivot Tables

```
tbl.pivot(col_var, row_var, values, collect)
```

- values: Table column to aggregate
- collect: Function to aggregate with

Either include **both** values and collect or **neither**

```
cat_tbl.pivot('Owner', 'Sex', 'Age', np.average)
```

Sex	Alice	Bob	Carol
F	14.5	3	6
M	0	8	2

Group vs Pivot

Group

- One combo of grouping variables **per row**
- **Any number** of grouping variables
- Aggregate values of **all other columns** in the table
- Missing combos are **absent**

Pivot

- One combo of grouping variables **per entry**
- **Two** grouping variables: columns and rows
- Aggregate values of **values column**
- Missing combos = **0 (or empty string)**

```
cat_tbl.groupby(['Sex', 'Coloring'], np.average)
```

Sex	Coloring	Name average	Age average	Weight average	Owner average
F	tabby		3	7	
F	tortie		6	10	
F	tuxedo		14.5	10	
M	tabby		5	12.25	

```
cat_tbl.pivot('Sex', 'Coloring', 'Weight', np.average)
```

Coloring	F	M
tabby	7	12.25
tortie	10	0
tuxedo	10	0

Joining Two Tables

Sometimes data about the same individuals are in different tables

- `join` combines the two datasets together
- Entries that do not appear in both tables are not included in the new table

To combine entries from `table1` and `table2` based on columns `c1` and `c2`

- `table1.join(c1, table2, c2)`

join Example

bubble_teas

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

discounts

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

join Example

bubble_teas

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

discounts

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

```
bubble_teas.join('cafe', discounts, 'location')
```

Match rows in
this table...

...using values in
this column ...

...with rows in this
second table...

...using values in
this column.

join Example

bubble_teas

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

discounts

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

```
bubble_teas.join('cafe', discounts, 'location')
```

Match rows in this table...

...using values in this column ...

...with rows in this second table...

...using values in this column.

output:

cafe	drinks	prices	% off
Gong Cha	Matcha Tea Latte	5.75	10
Hey Tea	Coconut Mango Boom	6.49	25
Moge Tee	Taro Milk Tea	7.45	5

join Example

bubble_teas

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

discounts

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

```
bubble_teas.join('cafe', discounts, 'location')
```

Match rows in this table...

...using values in this column ...

...with rows in this second table...

...using values in this column.

output:

cafe	drinks	prices	% off
Gong Cha	Matcha Tea Latte	5.75	10
Hey Tea	Coconut Mango Boom	6.49	25
Moge Tee	Taro Milk Tea	7.45	5

Next Class

- Today
 - Functions and Apply
 - Group and Pivot
- *Wednesday*
 - Conditionals and iteration