COMS BC1016

Introduction to Computational Thinking and Data Science

Lecture 9: Conditionals and Iteration

Oct 1, 2025

Upcoming Schedule

Date	Topic	Lab	Assignment
10/1	9 - Conditionals and Iteration	Lab 4 - Functions and Visualizations (Due 10/3) Courseworks	HW2 Due
10/6	10 - Probability and Sampling		HW4 - Probability, Simulation, Estimation (Due 10/15) Courseworks
10/8	11 - Models and Empirical Simulations	Lab 5 - Simulations (Due 10/10) Courseworks	HW3 Due
10/13	Programming/Python Review		
10/15	Midterm Review	*No Lab*	HW4 Due
10/20	**Midterm Exam**		
10/22	Special Topics - Bias in AI	*No Lab*	

Lecture Outline

- Comparison Operators
- Control Statements
    - If statements
    - For loops
- Randomness

Groups, Pivot Tables, `join`

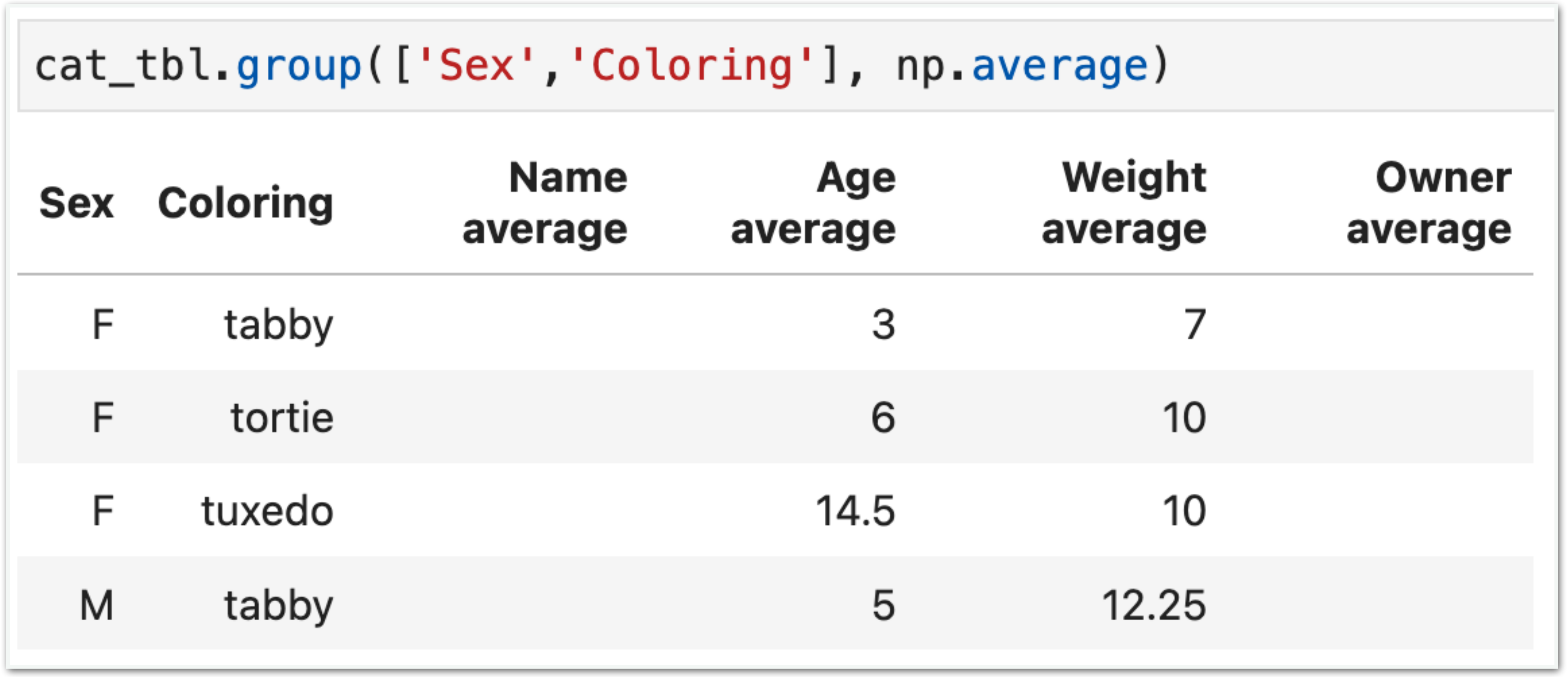
Group vs Pivot

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**

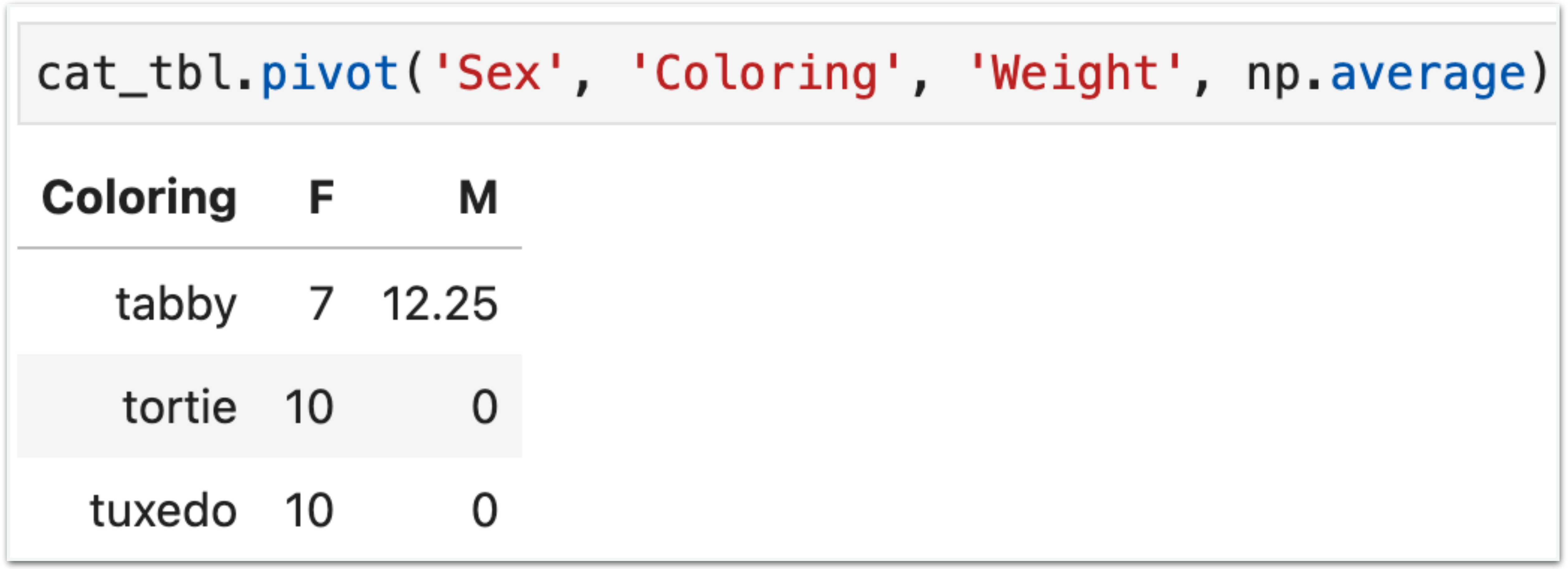
Pivot

- 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.group(['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

Booleans and Comparisons

Boolean Data Type

- Booleans are data types for truth values: **`True`** or **`False`**
    - **`True`** is equivalent to `1`
    - **`False`** is equivalent to `0`
- `bool(x)` turns `x` into a boolean
    - e.g., `bool(1)` evaluates to **`True`** and `bool(0)` evaluates to **`False`**

Comparison Operators

Operation	Meaning
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`==`	equal to
`!=`	not equal to

Comparison Operators

Example	Result	Explanation
`3 > 2`	`True`	3 is greater than 2
`3 > 3`	`False`	3 is not (*strictly*) *greater than* 3
`4 <= 4`	`True`	4 is less than or equal to 4

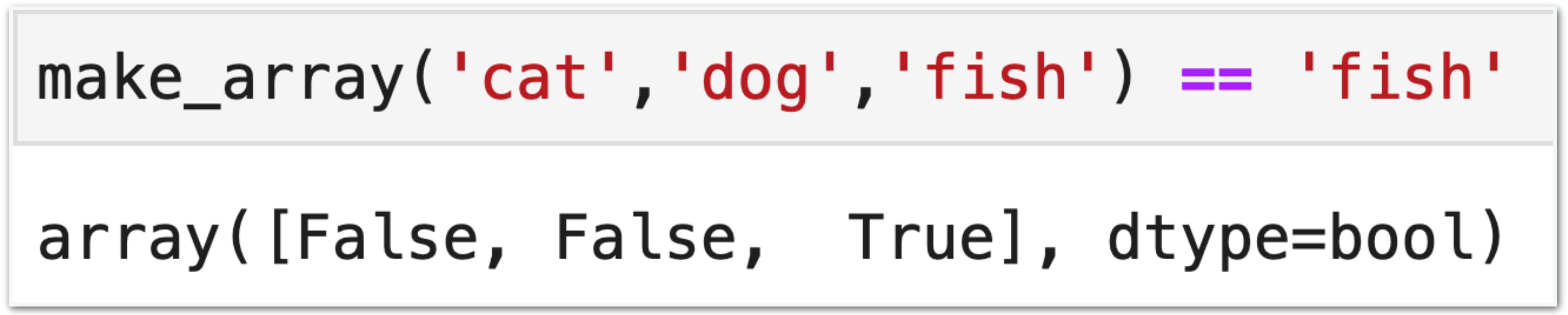
Comparison Operators

Example	Result	Explanation
`'4' == 4`	**`False`**	`'4'` is a string and `4` is an int
`3 - 2 == 4 - 3`	**`True`**	`3-2` equals `1` and `4-3` equals `1`; `1` equals `1`
`2 != 2`	**`False`**	`2` is not *not* equal to `2`

Comparisons with Arrays

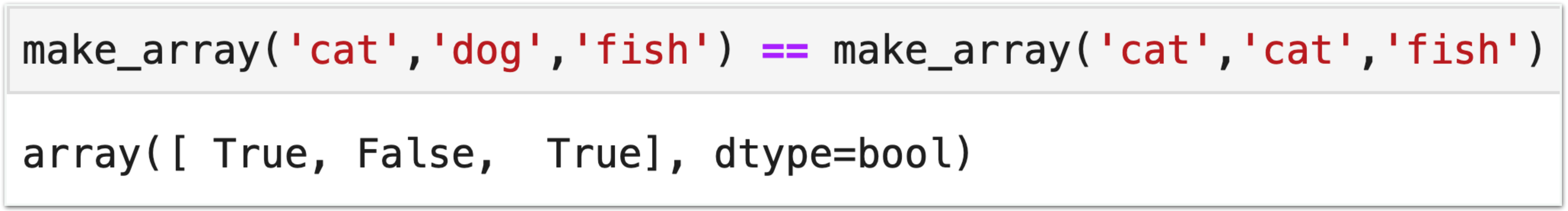
- Single values can be compared against each element in an array

- Comparing two arrays will compare element-by-element



```
make_array('cat','dog','fish') == 'fish'
```

```
array([False, False,  True], dtype=bool)
```



```
make_array('cat','dog','fish') == make_array('cat','cat','fish')
```

```
array([ True, False,  True], dtype=bool)
```

`and`, `or`, and `not`

- You can combine conditional statements using `and` & `or`
  - `and` will return `True` if **all** expressions are `True` (and `False` otherwise)
  - `or` will return `True` if **any** expressions is `True` (and `False` otherwise)
- You can negate a boolean value using `not`
  - `not True` will evaluate to `False`
  - `not False` will evaluate to `True`

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	`True`
`False or False`	`False`
`not False`	`True`

Aggregating Comparisons

- Summing an array or list of bool values will count the `True` values only

Example	Result
`True + False + True`	2
`1 + 0 + 1`	2
`sum([True, False, True])`	2

Control Statements

Control Statements

Control Statements modify *if* and/or *how many times* a block of code is executed in a program

Control Statements

- Two major types are `if` and `for`
  - `if` statements specify code that should be run conditioned on something being true
    - They can also specify if alternative code should be run otherwise
  - `for` loops allow executing code over each element in some sequence of items

`if` statements

- Conditionals begin with an `if` followed by a boolean statement
    - Runs code based on whether a boolean statement evaluates to `True`
- Conditionals can include a combination of `if`, `elif`, and `else` clauses
    - Maximum of one `if` and one `else`

`if` statements

```
if statement_1:
   first_code_block
elif statement_2:
   second_code_block
elif statement_3:
   third_code_block
else:
   fourth_code_block
```

if statements

```
if statement_1:
    first_code_block
elif statement_2:
    second_code_block
elif statement_3:
    third_code_block
else:
    fourth_code_block
```

Runs if statement_1 == True

Runs if statement_1 != True
AND statement_2 == True

statement_1 != True
AND statement_2 != True
AND statement_3 == True

nothing above == True

Iteration

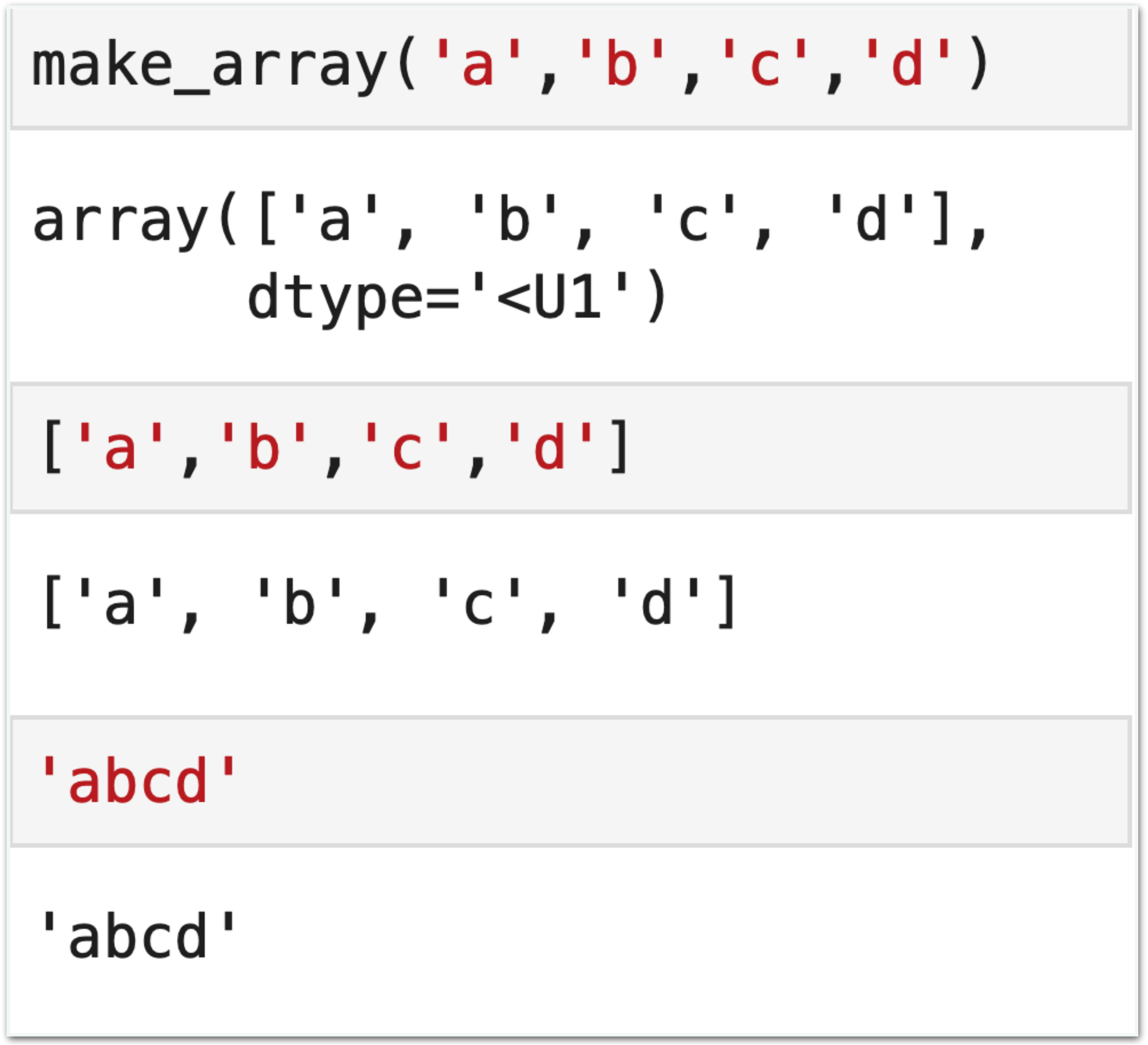
- **Iteration** means to repeat a process or steps
    - For example, coming up with a design, prototyping, testing, and then repeating these steps based on the outcome
- In programming we use this term to refer to executing code repeatedly over every element in a list/array/sequence/collection/…
    - The object being iterated over is referred to as an **iterable**

Iterables

- Formally, an iterable is any Python object capable of returning its members one at a time

- Iterables we've seen in this class include:
    - Arrays
    - Lists
    - String

We'll mostly focus on arrays



```
make_array('a','b','c','d')
```

```
array(['a', 'b', 'c', 'd'],
      dtype='<U1')
```

```
['a','b','c','d']
```

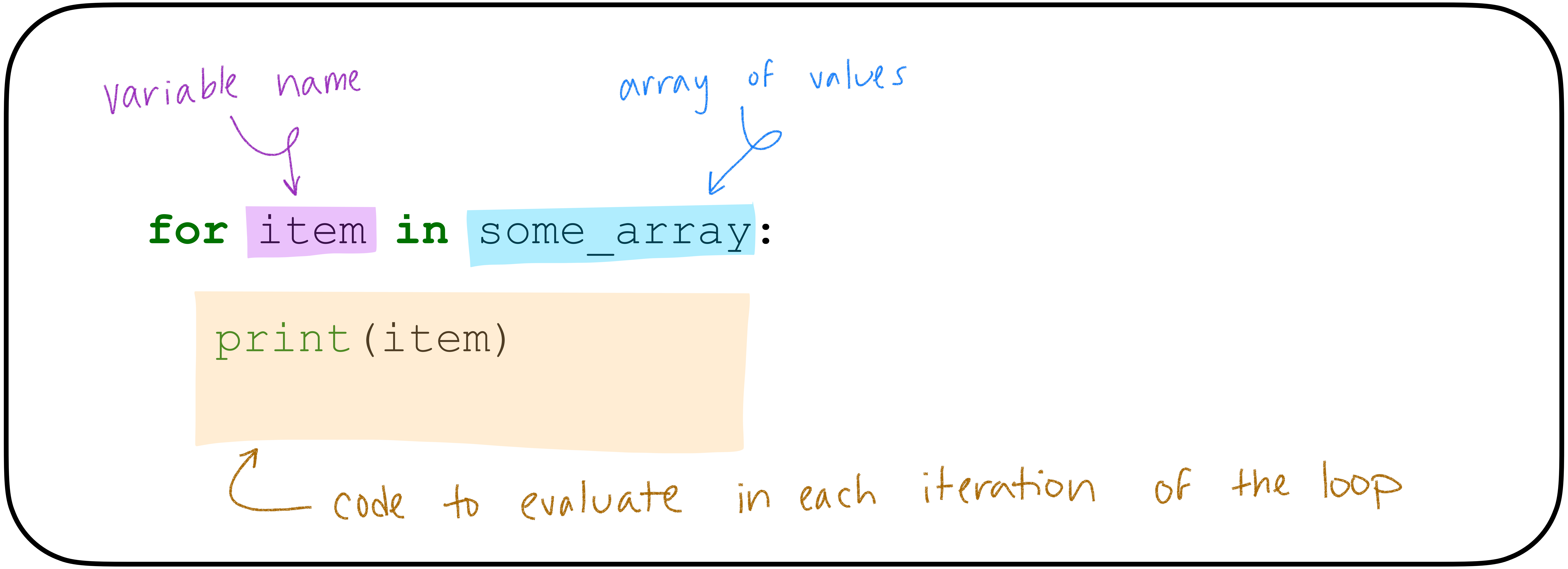
```
['a', 'b', 'c', 'd']
```

```
'abcd'
```

```
'abcd'
```

for Statements

- Executing a **for** runs code with each element in an iterable



```
for item in some_array:
    print(item)
```

Random Selection

Random Selection

```
import numpy as np
```

To select uniformly at random from array `some_array`

```
- np.random.choice(some_array)
```

To select `n` number of random elements from array `some_array`

```
- np.random.choice(some_array, n)
```

Appending Arrays

Appending Arrays

```
import numpy as np
```

Return a copy of `array_1` where `value` is added onto the end

```
np.append(array_1, value)
```

Returns an array with elements of `array_1` followed by elements of `array_2`

```
np.append(array_1, array_2)
```

Next Time

- Chance and Sampling