COMS BC1016
Introduction to Computational Thinking and Data Science

Lecture 11: Simulations and Sampling

Spring 2025

February 23, 2026

Upcoming Labs

- **Labs are in-person today and tomorrow**
  - We will update if anything gets moved online
- **After this week, we will not have labs until after spring recess**
  - No labs next week or the week after
  - Labs resume March 25/26

Midterm Logistics

- Midterm will be an in-class paper exam on Wednesday, March 11
  - *Please contact CARDS or ODS as soon as possible if you need any accommodations*
- You are permitted to bring a single formula sheet (5”x8” index card, double-sided) that will be submitted along with the exam
  - Exam is otherwise closed-note and no computers
- Questions will be a mix of multiple choice and short answer
  - You will not be asked to write programs on paper but you should expect to read code and understand it
- There will be a review session during class on Monday, March 9

Final Project

- There is no final exam in this course
- Instead, you will be working on a final project analyzing real life data sets
  - Groups of 2 people
  - After spring recess we'll ask everyone to declare who they want to work with for the final project
    - Start thinking about who you want to work with now!
    - You can optionally work in groups of 3 (with an extra requirement) or alone (with the same amount of work required for a group of 2)
- We'll share more details after spring recess

Review: Conditionals

`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
```

`if` statements

```
if statement_1:
    first_code_block
else:
    second_code_block
```

Runs if statement_1 == True

nothing above == True

`if` statements

Shorthand for "else if"

```
if statement_1:
    first_code_block
elif statement_2:
    second_code_block
```

Runs if statement_1 == True

Runs if statement_1 != True AND statement_2 == True

`if` statements

Shorthand for "else if"

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

Runs if statement_1 == True

Runs if statement_1 != True AND statement_2 == True

nothing above == True

`if` statements

Shorthand for "else if"

```
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

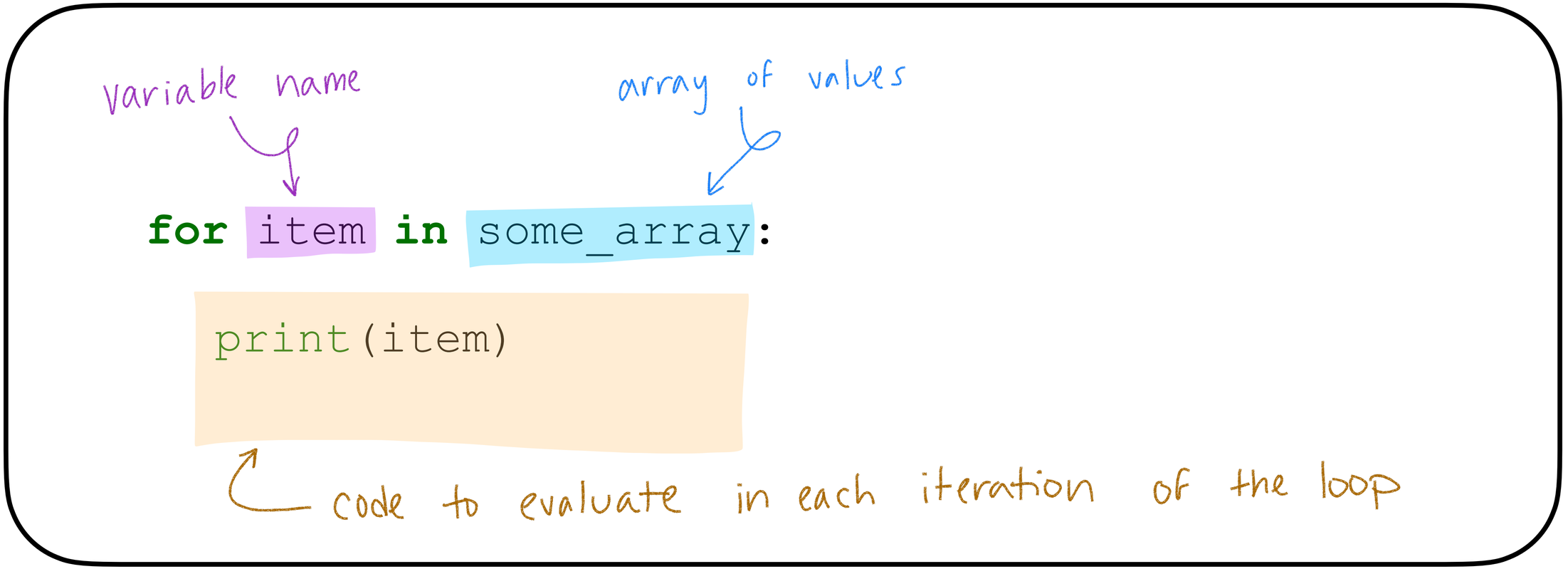
NYC Weather Demo (again)

Review: For Loops

`for` Statements

e.g., arrays, lists, strings, …

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



`for` Example

```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

code to evaluate in each iteration of the loop

```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

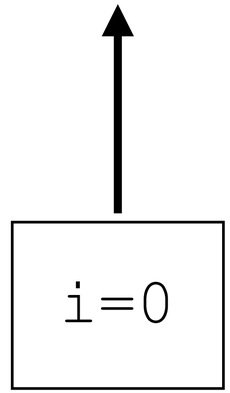
`for` Example

```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
```

```
np.arange(4)
```

```
array([0, 1, 2, 3])
```



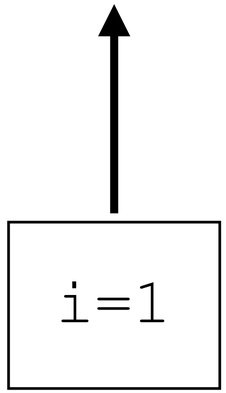
`for` Example

```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
1
```

```
np.arange(4)
```

```
array([0, 1, 2, 3])
```



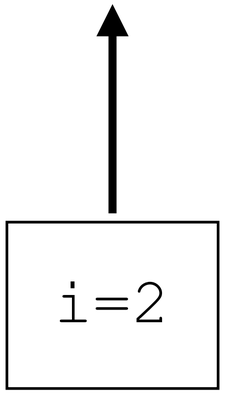
`for` Example

```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
1
3
```

```
np.arange(4)
```

```
array([0, 1, 2, 3])
```



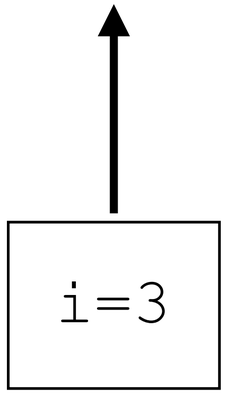
`for` Example

```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
1
3
6
```

```
np.arange(4)
```

```
array([0, 1, 2, 3])
```



Experimentation and Simulation

Experimentation

- Why do we want to run experiments?
  - Test a hypothesis
- Let’s say we want to test coin flipping. What could be a hypothesis for our experiment?
  - Is the coin loaded (i.e., not fair)?

General Process for Simulations

1. Figure out what you want to simulate

   - Example: Outcomes of a coin toss

General Process for Simulations

1. Figure out what you want to simulate

   - Example: Outcomes of a coin toss

2. Write a function whose output is the outcome of a single simulation

General Process for Simulations

1. Figure out what you want to simulate

   - Example: Outcomes of a coin toss

2. Write a function whose output is the outcome of a single simulation

3. Repeat the simulation for some number of iterations

   - Keep track of the results of every iteration in an array

General Process for Simulations

1. Figure out what you want to simulate

   - Example: Outcomes of a coin toss

2. Write a function whose output is the outcome of a single simulation

3. Repeat the simulation for some number of iterations

   - Keep track of the results of every iteration in an array

4. Add results array to a table so you can plot the results

General Process for Simulations

1. Figure out what you want to simulate
   - Example: Outcomes of a coin toss
2. Write a function whose output is the outcome of a single simulation
3. Repeat the simulation for some number of iterations
   - Keep track of the results of every iteration in an array
4. Add results array to a table so you can plot the results

Typically what we want to know about is influenced by chance or randomness

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)`

Note: Random does not mean arbitrary.

We mean each output has some chance of happening (probability)

General Process for Simulations

1. Figure out what you want to simulate

   - Example: Outcomes of a coin toss

2. Write a function whose output is the outcome of a single simulation

3. Repeat the simulation for some number of iterations

   - Keep track of the results of every iteration in an array

4. Add results array to a table so you can plot the results

To keep track of our results, we will want to append (add to the end) elements onto our 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)
```

Simulation Demo

Next Class

- Today
  - Simulations and Sampling
- Monday
  - Probability Review
  - Sampling Distributions and Models