Data Collections

- Using Python to Analyze Larger Data Sets

CMSC 104 Section 2
November 11, 2025

Administrative Notes

Quiz 3 is Thursday, in class

- Open-book, same as Quiz 2

Project 3 is out

- An updated “starter code” file is available on Github

Thanksgiving is in two weeks

- Right now, we are planning on having class on Tuesday, November 25
- If that changes, we’ll let you know

Large datasets

Python’s built-in list and dictionary data types are solid bases for processing much larger datasets

- They can be composed into things like “dataframes” that are commonly used to process huge datasets - thinks with millions or billions of data items
- Python’s a much better language for big data than others because of these types

Today and next Tuesday we’ll work through some “slightly larger” datasets

Lists - a recap

Built-in list operations

Operator	Meaning	Result
list1+list2	concatenation	Creates a new list containing the elements of both constituent lists
list*integer	repetition	Creates a new list with the elements of the old list repeated
list [ ]	indexing	Identifies a specific element of the list
len(list)	length	Returns an integer
list[:]	slicing	Returns a sub-list
for item in list:	iteration	Steps through the list, one element at a time
item in list	Membership check	Returns a boolean

Lists - Pre-defined list methods

Method	Meaning	Result
append	Add one element to the end of the list	
sort	Puts the list elements in order, smallest to largest	
reverse	Reverses the elements of the list	
index	Returns the index of the first occurrence of the indicated value	
insert	Inserts the item into the list at the designated index	
count	Returns the number of occurrences of the value in the list	
remove	Deletes the first occurrence of the item in the list	
pop	Deletes the list element at the indicated index	

Calculating mean, median and standard deviation

```
def calc mean(nums):
    total = 0.
    for n in nums:
        total += n
    return
total/len(nums)
```

```
def calc_std_dev(nums):
    v = 0
    xbar = calc_mean(nums)
    for n in nums:
        v = v + (n - xbar)**2
    sd = (v/ len(nums))**(1/2)
    return sd
```

```
def calc_median(nums):
    nums.sort()
    size = len(nums)
    midpos = size//2
    if size%2 == 0:
        median = (nums[midpos] + nums[midpos-1])/2
    else:
        median = nums[midpos]
    return median
```