# Lists

# Learning Goals

- Read and write code using 1D and 2D lists

- Use list methods to change lists without variable assignment

# Lists are Containers for Data

A list is a data structure that holds an ordered collection of data values.

Example: a sign-in sheet for a class.

Sign In Here

1. Elena
2. Max
3. Eduardo
4. Iyla
5. Ayaan

Lists make it possible for us to assemble and analyze a collection of data using only one variable.

# List Syntax

We use square brackets to set up a list in Python.

```
a = [ ] # empty list
b = [ "uno", "dos", "tres" ] # list with three strings
c = [ 1, "dance", 4.5 ] # lists can have mixed types
```

# Basic List Operations

Lists share most of their basic operations with strings.

```
a = [ 1, 2 ] + [ 3, 4 ] # concatenation – [ 1, 2, 3, 4]
b = [ "a", "b" ] * 3 # repetition – [ "a", "b", "a", "b", "a", "b" ]
c = [ 1, 5 ] < [ 2, 4 ] # comparison/equality – True
d = 4 in [ "a", "b", 1, 2 ] # membership – False
```

This includes indexing and slicing.

```
lst = [ "a", "b", "c", "d" ]
print(lst[1]) # indexing – prints "b"
print(lst[2:]) # slicing – prints [ "c", "d" ]
```

# List Functions and Methods

There are a few useful built-in functions that work directly on lists.

```
len(lst) # the number of elements in lst
min(lst)/max(lst) # the smallest/largest element in lst
sum(lst) # the sum of the elements in lst
```

There are also some list methods which are called directly on the list, like string methods.

```
lst.count(element) # the number of times element occurs in lst
lst.index(element) # the first index of element in lst
```

# Looping Over Lists

Looping over lists works the same way as with strings. We can use a for loop over the indexes of the list to access each item. For example, the following loop sums all the values in prices.

```
total = 0
for i in range(len(prices)):
        total = total + prices[i]
print(total)
```

# Use s.split(c) to Turn Strings Into Lists

We'll also use a new string method, s.split(c), to split up a string into a new list based on a separator character, c. This is highly useful for working with text data (books, scripts, chat logs...).

```
def findName(sentence, name):
    words = sentence.split(" ")
    for i in range(len(words)):
        if words[i] == name:
            return True
    return False

findName("Ask Tom to phone Nina", "Tom")
# words holds ["Ask", "Tom", "to", "phone", "Nina"]
```

# Activity: Predict the Result

What will be printed after each of the following code snippets?

```
lst = ["a", "b", 1, 2]

print(lst[1]) # Q1


s = " "


a = " hello world"

for i in range(len(a)):
    s = s + str(a[i])
print(s) # Q2
```

# Example: findMax(nums)

Let's write a function that finds the maximum value in a list of integers (without using the built-in max function).

```
def findMax(nums):
    biggest = nums[0] # why not 0? Negative numbers!
    for i in range(len(nums)):
        if nums[i] > biggest:
            biggest = nums[i]
    return biggest
```

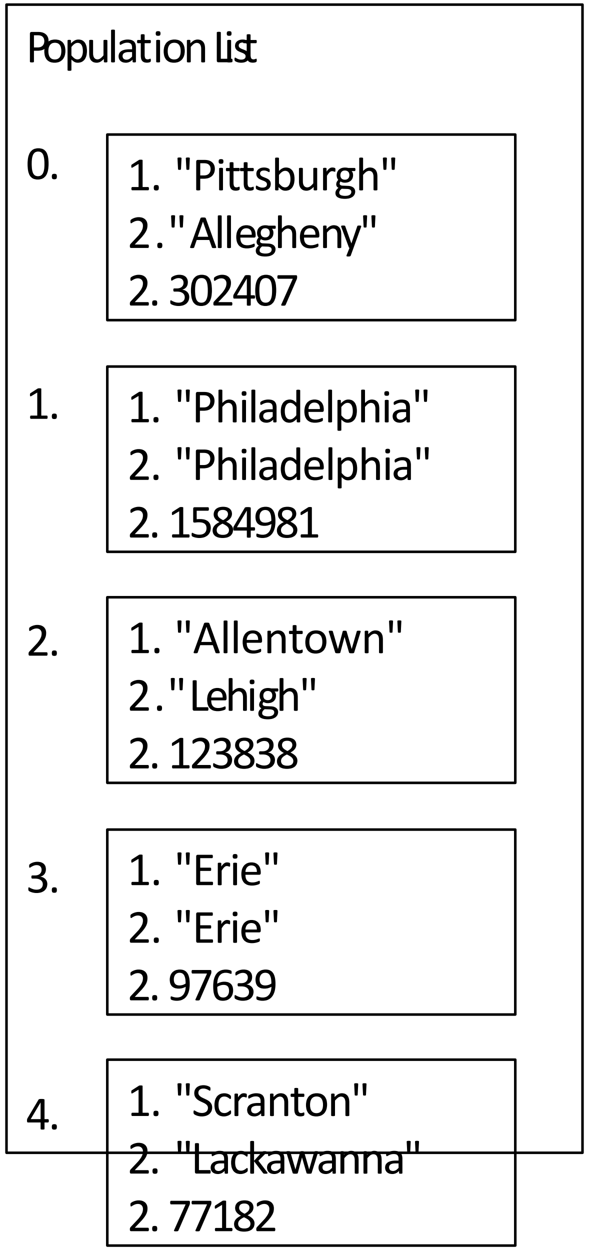
We'll often use this algorithmic structure to find the biggest/best item in a structure.

# 2D Lists

# 2D Lists are Lists of Lists

We often need to work with data that is two-dimensional, such as the coordinates on a grid, values in a spreadsheet, or pixels on a screen. We can store this type of data in a 2D list, which is just a list that contains other lists.

For example, the 2D list to the right holds population data, where each population datapoint itself contains multiple data values (city, county, and population).



Population List

0.
1. "Pittsburgh"
2."Allegheny"
2. 302407

1.
1. "Philadelphia"
2. "Philadelphia"
2. 1584981

2.
1. "Allentown"
2."Lehigh"
2. 123838

3.
1. "Erie"
2. "Erie"
2. 97639

4.
1. "Scranton"
2. "Lackawanna"
2. 77182

# Syntax of 2D Lists

Setting up a 2D list is no different than setting up a 1D list; each inner list is one data value.

```
cities =            [ ["Pittsburgh", "Allegheny", 302407],
                    ["Philadelphia", "Philadelphia", 1584981],
                    ["Allentown", "Lehigh", 123838],
                    ["Erie", "Erie", 97639],
                    ["Scranton", "Lackawanna", 77182] ]
```

When indexing into a 2D list, the first square brackets index into a row and the second index into a column. The length of a 2D list is the number of lists in the outer list.

```
cities[2]           # [ "Allentown", "Lehigh", 123838 ]
cities[2][1]        # "Lehigh"
len(cities)         # 5
```

# Looping Over 2D Lists

We can loop over a 2D list the same way we loop over a list. Indexing into a list once will produce an inner list. We'll need to index a second time to get a value.

```
def getCounty(cities, cityName):
    for i in range(len(cities)):
        entry = cities[i] # entry is a list
        if entry[0] == cityName:
            return entry[1]
    return None # city not found
```

# List Methods

# Some List Methods Change the List

Sometimes we want to modify a list directly, to add or remove elements from it. There are a set of list methods that can do this without using variable assignment at all.

```
lst = [ 1, 2, "a" ]
lst.append("b") # adds the element to the end of the list
```

Note that we do not set lst = lst.append; the list is changed in place. In fact, the append method returns None, not a list. We'll talk more about how this works in our demo.

# Example: getFactors(n)

Let's write a function that takes an integer and returns a list of all the factors of that integer.

```
def getFactors(n):
    factors = [ ]
    for num in range(1, n+1): # num is a possible factor
        if n % num == 0:
            factors.append(num)
    return factors
```

# Additional List Methods

Here are a few other useful list methods that change the list in place:

```
lst = [ 1, 2, "a" ]
lst.insert(1, "foo") # inserts 2nd param into 1st param index
lst.remove("a") # removes the given element from the list once
lst.pop(0) # removes the element at given index from the list
```