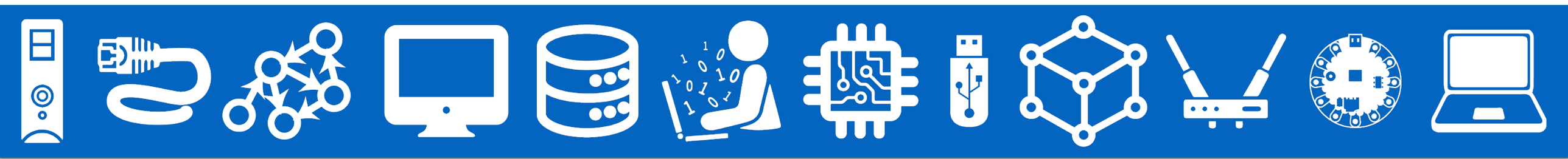
CS 134:
Querying Sequences



Announcements & Logistics

- **HW 4** due today
- **Lab 4**
  - Part 1 starts today/tomorrow, due Wednesday/Thursday
    - Test results given immediately
  - Part 2 due following Wednesday/Thursday
    - Pre-Lab for Part 2: Fix up your Part 1 code!
- Feedback for **Lab 1** and **Lab 2** are both available in Gradescope
- **Final Exam** schedule is posted: Wednesday, December 11 at 9:30am

Do You Have Any Questions?

Last Time

- New iteration statement: the **`while`** loop
    - "Conditional" looping statement
    - Useful when we don't know a sequence or stopping condition ahead of time

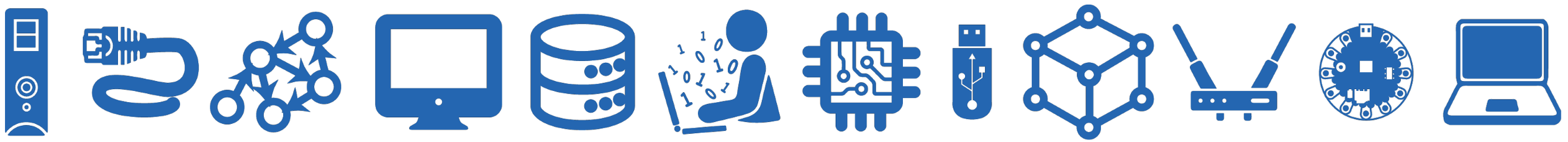
Today's Plan

- Finish practice with looping over nested lists and other sequences
- Module vs scripts
    - How to import and test functions
    - Role of the special `if __name__ == "__main__":` code block

sched.py

- example of collecting statistics over data stored in lists of lists

Modules & Scripts



Modules and Scripts: Example Code

leap.py

```
def is_leap(year):
    """Takes a year (int) as input and returns
    True if it is a leap year, else returns False"""

    # if not divisible by 4, return False
    if year % 4 != 0:
        return False

    # is divisible by 4 but not divisible by 100
    # return True
    elif year % 100 != 0:
        return True

    # is divisible by 4 and divisible by 100
    # but not divisible by 400, return False
    elif year % 400 != 0:
        return False

    # is divisible by 400 (and also 4, and 100)
    # return True
    return True
```

Modules and Scripts

- A **script** is a piece of code saved in a file, e.g., `leap.py`
  - Meant to be executed with: `python3 leap.py`
- A **module** is a collection of function definitions saved in a file (like a script)
  - Meant to be imported and used by other scripts
  - Can be used in interactive python
- Code in a `.py` file can serve as both a module and a script
- To distinguish between these two modes of operation, we can check the value of the special variable called `__name__`
- *Note: If a variable starts/ends with double __ in Python, it's a special variable*

Modules and Scripts

- Consider the code we wrote in `leap.py`
- When `leap.py` is run as a **script** then the `__name__` variable is set to the string `"__main__"`
- When we import the code as a **module**, the `__name__` variable is set to the name of the module `leap`

- Why does this matter?
    - We often want different behavior when the code is run as a script vs when it's imported as a module

`if __name__ == '__main__':`

- This is just an if statement with an equality Boolean expression:
  - Checks whether the `__name__` variable is set to the string `'__main__'`. Tells us the code is being run as a script
- We place code that we want to run only when our module is executed as a script inside the `if __name__ == "__main__"`: block
- Useful for testing code that we do not want to run when we import functions in interactive Python

Example: Script vs Module

```
# name.py
# test the role of __name__variable
print("__name__ is set to", __name__, "\n\n")
```

```
terminal % python3 name.py
__name__ is set to __main__

terminal % python 3
Python 3.10.8 (main)
Type "help", "copyright", "credits" or "license" for more information.
>>> import name
__ name__ is set to name
```

leap.py

```python
# function to check if a given year is a leap year

def is_leap(year):
    """Takes a year (int) as input and returns
    True if it is a leap year, else returns False"""

    # if not divisible by 4, return False
    if year % 4 != 0:
        return False

    # is divisible by 4 but not divisible by 100
    # return True
    elif year % 100 != 0:
        return True

    # is divisible by 4 and divisible by 100
    # but not divisible by 400, return False
    elif year % 400 != 0:
        return False

    # is divisible by 400 (and also 4, and 100)
    # return True
    return True


#  following code only run when run as a script
if __name__ == "__main__":
    # ask user to enter year
    year = int(input("Enter a year: "))

    # call isLeap
    if is_leap(year):
        print(year, "is a leap year!")
    else:
        print(year, "is not a leap year.")
```

Running leap as a Script and Module

Running leap as a Script and Module

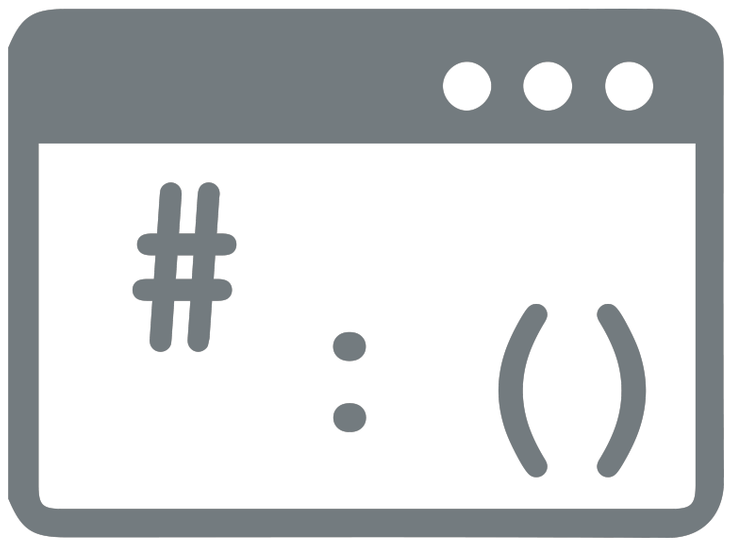
- Running leap.py as a script (notice the code in the if block runs!)

```
terminal$ python3 leap.py
Enter a year: 1900
1900 is not a leap year.
terminal$ python3 leap.py
Enter a year: 2040
2040 is a leap year!
```

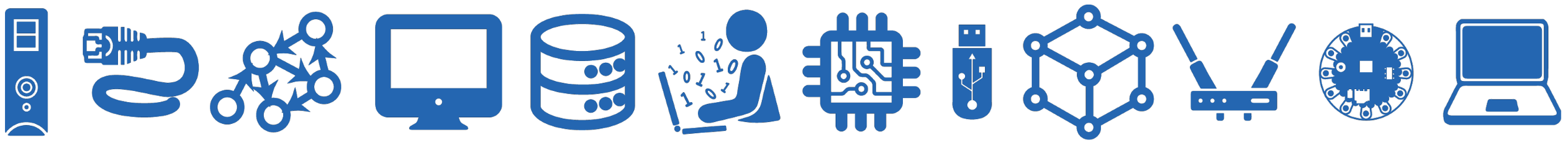
- Running leap.py as a module in interactive Python

```
terminal$ python3
Python 3.10.8 (main)
Type "help", "copyrigh...
>>> from leap import *
>>> is_leap(1900)
False
>>> is_leap(2040)
True
```

Examples:
Script & Module



Processing Sequences of Strings



Processing Sequences of Strings

- Lots of string & list & sequence operators!
- What to do with it all?!
- Let's play with some lists of names!

Character List

- Given a `name_list` (of strings) and a `char`acter, return a list of all names that start with that character

```
def character_list(name_list, char):
    """ Take a list of names/strings and a string character
    and returns a list of names whose name start with that character"""
    # initialize accumulation list
    # look at each name
        # if it starts with the character
            # accumulate it!
    # when done, return accumulated names
```

Character List

- Given a `name_list` (of strings) and a `char`acter, return a list of all names that start with that character

```
def character_list(name_list, char):
    """ Take a list of names/strings and a string character
    and returns a list of names whose name start with that character"""
    result = []
    for name in name_list:
        if name[0] == char:
            result = result + [name]
    return result
```

```
>>> character_list(["lida doret", "mark hopkins", "iris howley", "shikha
singh", "bill jannen", "sam mccauley"], 's')
['shikha singh', 'sam mccauley']
```

Long & Short Names

- Given a `name_list` (of strings) and a `long` and `short` threshold, return a list of lists containing the names with **first** names longer than `long` and shorter than `short`

```
def longshort_names(name_list, lon, short):
    """ Takes a list of strings/names, and two integers representing a long
    and short threshhold and returns a list of lists holding all names from
    name_list longer than lon, and another of names shorter than short"""
    # initialize long accumulation list
    # initialize short accumulation list
    # look at each name
        # get the first name
        # if it's longer than long
            # accumulate it!
        # if it's shorter than short
            # accumulate it!
    # when done, return accumulated names
```

Long & Short Names

- Given a `name_list` (of strings) and a `long` and `short` threshold, return a list of lists containing the names with **first** names longer than `long` and shorter than `short`

```
def longshort_names(name_list, lon, short):
    """ Takes a list of strings/names, and two integers representing a long
    and short threshhold and returns a list of lists holding all names from
    name_list longer than lon, and another of names shorter than short"""
    long_names = []
    short_names = []
    for name in name_list:
        firstname = get_firstname(name) # Need to implement this!
        if len(firstname) > lon:
            long_names = long_names + [name]
        elif len(firstname) < short:
            short_names = short_names + [name]
    return [long_names, short_names]
```

First Name

- Given a **`name`** (string) return a string containing only the name's first name

```
def get_firstname(name):
    """ a helper method to grab the first name from a given str, name"""
    # initialize string accumulator variable
    # look at each character
        # if it's a space, we're done! return the name
        # otherwise, accumulate this character
    # when we run out of characters, just return the whole name
```

First Name

- Given a **`name`** (string) return a string containing only the name's first name

```
def get_firstname(name):
    """ a helper method to grab the first name from a given str, name"""
    firstname = ''
    for char in name:
        if char == ' ':
            return firstname
        else:
            firstname = firstname + char
    return name
```

```
>>> get_firstname("lida doret")
'lida'
```

Long & Short Names

- Given a `name_list` (of strings) and a `long` and `short` threshold, return a list of lists containing the names with **first** names longer than `long` and shorter than `short`

```
def longshort_names(name_list, lon, short):
    """ Takes a list of strings/names, and two integers representing a long
    and short threshhold and returns a list of lists holding all names from
    name_list longer than lon, and another of names shorter than short"""
    long_names = []
    short_names = []
    for name in name_list:
        firstname = get_firstname(name)
        if len(firstname) > lon:
            long_names = long_names + [name]
        elif len(firstname) < short:
            short_names = short_names + [name]
    return [long_names, short_names]
```

```
>>> longshort_names(["lida doret", "mark hopkins", "iris howley", "shikha singh", "bill jannen", "someone b. somebody"],6,5)
[['someone b. somebody'], ['lida doret', 'mark hopkins', 'iris howley', 'bill jannen']]
```

Last Names

- Given a `name_list` (of strings) return a list of strings representing only the last names in `name_list`

```
def last_names(name_list):
    """ takes a list of names (strings) and returns just a
    list of last names """

    # initialize accumulation list
    # look at each name
        # get the last name
        # accumulate it!
    # when done, return accumulated names
```

Last Names

- Given a `name_list` (of strings) return a list of strings representing only the last names in `name_list`

```
def last_names(name_list):
    """ takes a list of names (strings) and returns just a
    list of last names """

    lastnames = []
    for name in name_list:
        lastnames = lastnames + [get_lastname(name)] # Need to implement this!
    return lastnames
```

Get Last Name

- Given a **`name`** return a string with just last name...or something more generalizable that can be used for both first and last name?!

```
def split(a_string, char):
    """ Given a string, a_string, split based upon character, char.
    Return as list.
    """
    # initialize accumulation list
    # initialize accumulation string
    # for each character in the string
        # if that character is char
            # append the current string we're building to accumulation list
            # reset the string accumulator
        # otherwise
            # accumulate the character on our current string
    # append the last word we were building, if we were building a word

    # when out of characters, return
```

Get Last Name

- Given a `a_string` (string) return a list containing strings split by `char`

```
def split(a_string, char):
    """ Given a string, a_string, split based upon character, char.
    Return as list.
    """
    result = []
    curr_string = ''
    for ch in a_string:
        if ch == char:
            result = result + [curr_string]
            curr_string = ''
        else:
            curr_string = curr_string + ch
    if curr_string:
        result = result + [curr_string]
    return result
```

```
>>> split("lida doret")
['lida', 'doret']
>>> split("madonna")
['madonna']
```

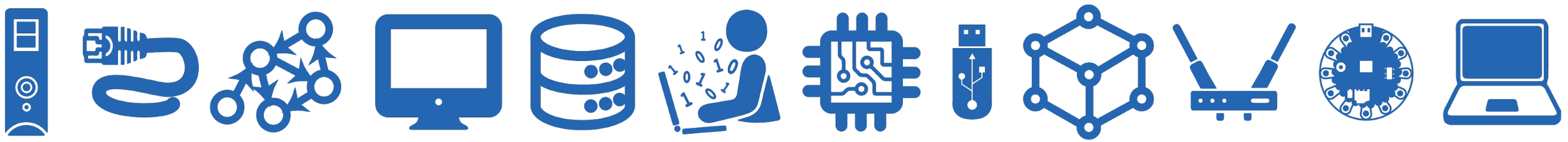
Last Names

- Given a `name_list` (of strings) return a list of strings representing only the last names in `name_list`

```
def last_names(name_list):
    """ takes a list of names (strings) and returns just a
    list of last names """

    lastnames = []
    for name in name_list:
        lastnames = lastnames + [split(name, ' ')][-1]
    return lastnames
```

```
>>> last_names(["lida doret", "mark hopkins", "iris howley", "shikha singh", "bill jannen",
"someone b. somebody"])
['doret', 'hopkins', 'howley', 'singh', 'jannen', 'somebody']
```

Querying Sequences



Querying Lists

- Asking for the min/max of a list according to a specified definition, or filtering the list according to some criteria is a task that comes up frequently in computer science.
- **ex: produce a list of names with the most number of vowels from a list of names**
- Decomposing the problem:
    - Will need to know if a character is a vowel or not
    - Will need to count the number of vowels
    - Will need to keep track of highest number
    - If something's higher, overwrite the old max

`is_vowel()` function

- Consider two versions of an `is_vowel()` function that takes a character (a string) as input and returns whether or not it is a vowel
  - Use `in` operator to simplify code (fewer boolean expressions)

```
def old_is_vowel(c):
    """ is_vowel function """
    return (c == 'a' or c == 'e' or c == 'i' or c == 'o' or c == 'u' or c == 'A' or c == 'E' or c == 'I' or c == 'O' or c == 'U')
```

```
def is_vowel(char):
    """ Simpler is_vowel function """
    return char in 'aeiouAEIOU'
```

Counting Vowels

- We can use a for loop to implement a `count_vowels()` function
- Notice how `count` "accumulates" values in the loop
- Recall, `count` here is called an **accumulation variable**

```
def count_vowels(word):
    """ Takes a string as input and returns
    the number of vowels in it """

    count = 0 # initialize the counter

    # iterate over the word one character at a time
    for char in word:
        if is_vowel(char): # call helper function
            count = count + 1
    return count
```

Exercise: Name Fun Facts!

- Write a function **`max_vowels`** that can be used to identify what the most number of vowels in all student names are. (Hint: use **`count_vowels()`** which returns the number of vowels in a string.)

```
def max_vowels(name_list):
    """ Takes a list of strings name_list and returns the number
    representing the maximum number of vowels in a name"""
```

- General strategy for finding max in list of sequences?
  - Initialize a max value BEFORE the loop to a very small number
  - If you see a value bigger than max while looping, update max

```
>>> max_vowels(["Lida", "Mark", "Rohit", "Anna", "Genevieve", "Maximilian"])
5
```

Exercise: Name Fun Facts!

- Write a function **`max_vowels`** that can be used to identify what the most number of vowels in all student names are. (Hint: use **`count_vowels()`** which returns the number of vowels in a string.)

```
def max_vowels(name_list):
    """ Takes a list of strings name_list and returns the number
    representing the maximum number of vowels in a name """

    max_so_far = 0

    for name in name_list:
        count = count_vowels(name)
        if count > max_so_far:
            # update found a name with more vowels
            max_so_far = count



    return max_so_far
```

What if we wanted a list of the names with the maximum number of vowels, instead of count?

```
>>> max_vowels(["Lida", "Mark", "Rohit", "Anna", "Genevieve", "Maximilian"])
5
```

Exercise: Name Fun Facts!

- Write a function **`most_vowels`** that can be used to compute the list of students with the most vowels in their first name. (Hint: use **`count_vowels()`** which returns the number of vowels in a string.)

```
def max_vowels(name_list):
    """ Takes a list of strings name_list and returns the number
    representing the maximum number of vowels in a name """

    max_so_far = 0

    for name in name_list:
        count = count_vowels(name)
        if count > max_so_far:
            # update found a name with more vowels
            max_so_far = count

    return max_so_far
```

Will need to initialize accumulator variable, and return that instead of the max num

New max found, throw out old accumulated values

What if it has the same number as our max? Add it to our accumulator!

```
>>> max_vowels(["Lida", "Mark", "Rohit", "Anna", "Genevieve", "Maximilian"])
5
```

Exercise: Name Fun Facts!

- Write a function **`most_vowels`** that can be used to compute the list of students with the most vowels in their first name. (Hint: use **`count_vowels()`** which returns the number of vowels in a string.)

```
def most_vowels(name_list):
    """ Takes a list of strings name_list and returns a list
    of names with the most number of vowels """

    max_so_far = 0
    result = []
    for name in name_list:
        count = count_vowels(name)
        if count > max_so_far:
            # update found a name with more vowels
            max_so_far = count
            result = [name]

        elif count == max_so_far:
            result = result + [name]

    return result
```

How would you modify this to compute the least number of vowels instead?

```
>>> most_vowels(["Lida", "Mark", "Rohit", "Anna", "Genevieve", "Maximilian"])
['Genevieve', 'Maximilian']
```

Exercise: Name Fun Facts!

- Write a function **`least_vowels`** that can be used to compute the list of students with the least vowels in their first name. (Hint: use **`count_vowels()`** again.)

```
def most_vowels(name_list):
    """ Takes a list of strings name_list and returns a list
    of names with the most number of vowels """

    max_so_far = 0
    result = []
    for name in name_list:
        count = count_vowels(name)
        if count > max_so_far:
            # update found a name with more vowels
            max_so_far = count
            result = [name]

        elif count == max_so_far:
            result = result + [name]

    return result
```

Rather than set the max low, set the min high?

Rather than looking for vals higher, want lower!

Need to use consistent variable names

```
>>> most_vowels(["Lida", "Mark", "Rohit", "Anna", "Genevieve", "Maximilian"])
['Genevieve', 'Maximilian']
```

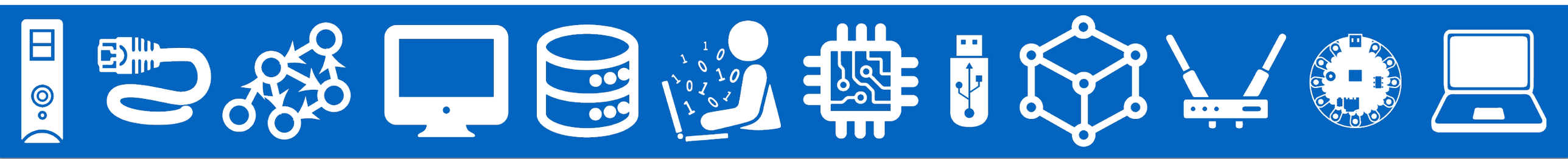
Exercise: Student Fun Facts!

- Write a function **`least_vowels`** that can be used to compute the list of students with the least vowels in their first name. (Hint: use **`count_vowels()`** again.)

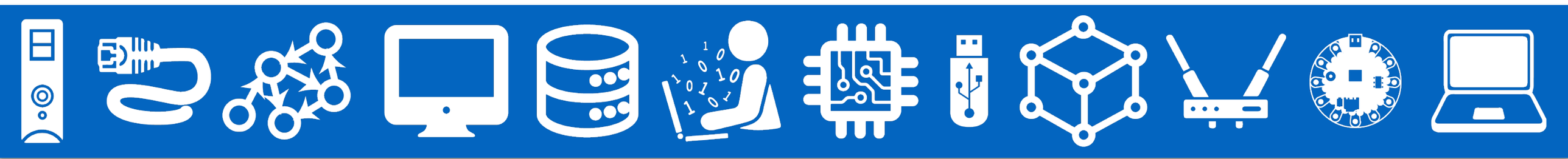
```
def least_vowels(name_list):
    """ Takes a list of strings, name_list, and returns a list
    of names with the least number of vowels """

    min_so_far = 100000 # when might this break? Do we have something better?
    result = []
    for name in name_list:
        count = count_vowels(name)
        if count < min_so_far:
            # update found a name with fewer vowels
            min_so_far = count
            result = [name]

        elif count == min_so_far:
            result = result + [name]

    return result
```

```
>>> least_vowels(["Lida", "Iris", "Rohit", "Anna", "Genevieve", "Maximilian"])
['Lida', 'Iris', 'Rohit', 'Anna']
```

The end!



Lab 4



Lab 4 Goals

- In Lab 4 you will implement several voting algorithms and helpful functions for manipulating election data

- Lab 4 will give you experience with :
  - Lists of strings
  - Lists of lists of strings
  - Loops
  - Sequence operators

- Pay close attention to expected input (lists of strings, list of lists of strings, etc) and expected output

Ballot Data

- Ballot data is represented in various text files
- Each line represents a single voter's ranked choices

```
[['kona', 'dickason', 'ambrosia', 'wonderbar', 'house'],
 ['kona', 'house', 'ambrosia', 'wonderbar', 'dickason'],
 ['kona', 'ambrosia', 'dickason', 'wonderbar', 'house'],
 ['kona', 'ambrosia', 'wonderbar', 'dickason', 'house'],
 ['house', 'kona', 'dickason', 'wonderbar', 'ambrosia'],
 ['kona', 'house', 'dickason', 'ambrosia', 'wonderbar'],
 ['kona', 'house', 'dickason', 'ambrosia', 'wonderbar'],
 ['dickason', 'ambrosia', 'wonderbar', 'kona', 'house'],
 ['house', 'kona', 'ambrosia', 'dickason', 'wonderbar'],
 ['ambrosia', 'house', 'wonderbar', 'kona', 'dickason'],
 ['wonderbar', 'ambrosia', 'kona', 'house', 'dickason'],
 ['house', 'wonderbar', 'kona', 'ambrosia', 'dickason']]
```

Ballot Data

- 0th ballot is :
  - `['kona', 'dickason', 'ambrosia', 'wonderbar', 'house']`
- Ranked 'kona' as their first choice, 'dickason' as second, 'ambrosia' as their third, etc. etc.

```
[['kona', 'dickason', 'ambrosia', 'wonderbar', 'house'],
 ['kona', 'house', 'ambrosia', 'wonderbar', 'dickason'],
 ['kona', 'ambrosia', 'dickason', 'wonderbar', 'house'],
 ['kona', 'ambrosia', 'wonderbar', 'dickason', 'house'],
 ['house', 'kona', 'dickason', 'wonderbar', 'ambrosia'],
 ['kona', 'house', 'dickason', 'ambrosia', 'wonderbar'],
 ['kona', 'house', 'dickason', 'ambrosia', 'wonderbar'],
 ['dickason', 'ambrosia', 'wonderbar', 'kona', 'house'],
 ['house', 'kona', 'ambrosia', 'dickason', 'wonderbar'],
 ['ambrosia', 'house', 'wonderbar', 'kona', 'dickason'],
 ['wonderbar', 'ambrosia', 'kona', 'house', 'dickason'],
 ['house', 'wonderbar', 'kona', 'ambrosia', 'dickason']]
```

Working with Ballot Data

```
>>> all_coffee[1] # access second inner list
['kona', 'house', 'ambrosia', 'wonderbar', 'dickason']

>>> all_coffee[0][1] # access second element in first inner list
'dickason'

>>> # access second character of second element of first inner list
>>> all_coffee[0][1][1]
'i'
```

```
>>> # create a list of only last elements of inner lists
>>> last_coffee = []
>>> for coffee in all_coffee:
...     last_coffee = last_coffee + [coffee[-1]]
>>> last_coffee
```

```
['house',
 'dickason',
 'house',
 'house',
 'ambrosia',
 'wonderbar',
 'wonderbar',
 'house',
 'wonderbar',
 'dickason',
 'dickason',
 'dickason']
```

You'll use string and list operators to process the data and implement several different voting algorithms

Remember

examples

from lecture!

Only use concepts we've learned from class so far!

The lab is doable using only what we've learned in class - that's the point of lab, to practice what we've learned!