

SI 506 Lecture 08

Topics

1. Warmup
 1. Challenge 01
 2. Challenge 02
2. **if-else** statements
 1. Challenge 03
3. Accumulator pattern (practice)
 1. Challenge 04
 2. Challenge 05
 3. Challenge 06

Vocabulary

- **Boolean.** A type (**bool**) or an expression that evaluates to either **True** or **False**.
- **Conditional Statement.** A statement that determines a computer program's *control flow* or the order in which particular computations are to be executed.
- **Index.** Numeric position of an element or item contained in an ordered sequence. Python indexes are zero-based, i.e., the first element's index value is 0 not 1. **len(< some_list >)** is considered an expression.
- **Iterable.** An object capable of returning its members one at a time. Strings, lists, and tuples are examples of an iterable.
- **Iteration.** Repetition of a computational procedure in order to generate a possible sequence of outcomes. Iterating over a **list** using a **for** loop is an example of iteration.
- **Operator.** A **symbol** for performing operations on values and variables. The assignment operator (**=**) and arithmetic operators (**+**, **-**, *****, **/**, ******, **%**, **//**).
- **Subscript operator.** Square brackets (**[]**) enclosing either an index value or a slicing expression that is used to access individual or groups of sequence characters, elements or items.

Reference

Open the following [w3schools](#) reference pages in your browser and bookmark them. The pages provide useful summaries of **str**, **list**, and **tuple** methods.

1. w3schools, "[Python Built-in Functions](#)" or python.org "[Built-in Functions](#)".
2. w3schools, "[Python Operators](#)".
3. w3schools, "[Python String Methods](#)".
4. w3schools, "[Python List Methods](#)".
5. w3schools, "[Python Tuple Methods](#)".

Lecture data

Today's lecture data is drawn from the US Dept of Energy's [Alternative Fuels Data Center](#) information on model year 2022 [electric vehicles](#) (EVs).

💡 Each EV is represented by a nested list. In addition, the data elements ("headers") that comprise each nested list are described in the first nested list.

```
elec_vehicles = [  
    ['powertrain', 'make', 'model', 'type', 'propulsion', 'drivetrain',  
    'city/combined/highway_mpge', 'range'],  
    ['BEV', 'Audi', 'e-tron GT', 'Sedan/Wagon', '175 kW electric motor;  
129 Ah battery', 'AWD', '81/82/83', '238'],  
    ...  
]
```

1.0 Warmup

Let's start with an in-class challenge that leverages the accumulator pattern.

1.1 Challenge 01

Task: Start to get to know the data by populating a list of *unique* EV vehicle "type" values.

The previous lecture introduced the membership operators `in` and `not in`. Utilize the accumulator pattern and the appropriate membership operator to populate a list of *unique* EV vehicle types (i.e., "Sedan/Wagon", "SUV", etc.).

1. Create an empty list and assign it to the variable `ev_types`.
2. Loop over a slice of `elec_vehicles` that **excludes** the first nested list element (the "headers").
3. Implement an `if` statement that checks whether or not the current EV "type" value has been added to `ev_types` employing the appropriate membership operator (e.g., `in` or `not in`).
4. If the type has *yet to be added* to `ev_types`, add the value. If the type was added to the accumulator list during a previous loop iteration *do not* add the value to the list in order to ensure the **uniqueness** of the list elements.
5. Uncomment `print()` and check your work.

1.2 Challenge 02

Task: Change the automaker name "Ford" to "Ford Motor Company" in the list `elec_vehicles`.

1. Employ `range` in a `for` loop together with an `if` statement to mutate the "maker" value from "Ford" to "Ford Motor Company".
2. You *must* skip the "headers" nested list when constructing your loop. To accomplish that requirement pass the appropriate `start` value (an integer) and `stop` value (an expression) to `range()`.

! Recall that you are looping over a sequence of numbers and not the list `elec_vehicles`. As you loop over the numbers utilize each to access the `elec_vehicles` elements. The access the "maker" element you *must* chain the subscript operators.
3. After mutating Ford EV "maker" values uncomment `print()` and check your work.

2.0 if-else conditions

Execution of an `if` statement's indented code block occurs *only* if the condition to be tested evaluates to `True`. If `False` is returned and a need exists to execute other statements in response an `else` statement can be added together with an indented code block.

```
if < condition >:
    < statement A >
    # ...
else:
    < statement B >
    # ...
```

The `if-else` block below evaluates an EV's "type"; if the EV is classified as a "Sedan/Wagon" a string representation of the vehicle is appended to the list `sedan_wagon`. Otherwise, string representations of all other EV types encountered are appended to the list `suv_pickup`.



the "headers" element is excluded from the loop.

```
sedan_wagon = []
suv_pickup = []
for vehicle in elec_vehicles[1:]:
    string = f"{vehicle[1]} {vehicle[2]} {vehicle[3]}"
    if vehicle[3] == ev_types[0]:
        sedan_wagon.append(string)
    else:
        suv_pickup.append(string)
```

A second example illustrates how to check if a value exists between a range of values. Assume that the automotive industry considers an EV battery range between 225-325 mpge (exclusive) as a "standard" range. Any EV battery ranges that fall on either side of the standard range are consider outliers. You can determine the number of standard and outlier EV ranges in `elec_vehicles` by implementing the following `if-else` statements:

```
standard_ranges = []
outlier_ranges = []
for vehicle in elec_vehicles[1:]:
    string = f"{vehicle[1]} {vehicle[2]} (range = {vehicle[-1]})"
    if 225 < int(vehicle[-1]) < 325:
        standard_ranges.append(string)
    else:
        outlier_ranges.append(string)
```

Translate the expression

```
225 < int(vehicle[-1]) < 325
```

as the EV battery range **between** 225 and 325 mpge (exclusive) with "exclusive" being interpreted as excluding the values 225 and 325 together with all other values outside the specified range.



if the minimum and/or maximum values are considered *inclusive* (i.e., part of the the range of values under consideration) use the less than or equal to (\leq) or greater than or equal to (\geq) comparison operators in the expression.

2.1 Challenge 03

Task. Return counts of domestic and foreign designed EVs.

Assume that EVs are designed (though not necessarily manufactured) in the automaker's country of origin. For this challenge you will employ the `us_automakers` list as a filter that permits you to distinguish between US and foreign automakers.

```
us_automakers = [
    'Chevrolet',
    'Ford',
    'Lucid USA, Inc.',
    'Polestar Automotive USA',
    'Rivian'
]
```

1. Assign zero (0) to two variables named `domestic_count` and `foreign_count` respectively.
2. Loop over a slice of `elec_vehicles` that **excludes** the first nested list element (the "headers").
3. Implement `if-else` statements that increment the counts assigned to `domestic_count` and `foreign_count`. In the `if` statement utilize the membership operator `in` to test whether or not an EV "maker" is considered a US automaker. If the membership check returns `True` increment the `domestic_count` by one (1). Otherwise, increment the `foreign_count` by one (1).

💡 the membership operator `in` returns `True` if a specified value `x` is a member (e.g., element, item, character) of a specified sequence `y` (`x in y`).

4. Uncomment `print()` and check your work.

3.0 Accumulator pattern (more practice)

Let's continue leveraging the accumulator pattern as we explore the `elec_vehicles` list.

3.1 Challenge 04

Task: Retrieve EVs featuring all-wheel drive (AWD) capabilities.

1. Create an empty list and assign it to the variable `awd_vehicles`.
2. Loop over a slice of `elec_vehicles` that **excludes** the first nested list element (the "headers").
3. Implement an `if` statement that filters on "drivetrain" and checks whether or not the EV's drivetrain is designated **"AWD"** (i.e., test for equality).
4. If `True` construct an f-string comprising the EV's "make" and "model" and add it to `awd_vehicles`. Format the f-string as follows:

```
f"< make > < model >"
```

5. Uncomment `print()` and `pprint()` and check your work.

3.2 Challenge 05

Task: Identify the EV with the shortest battery range.


The previous lecture demonstrated how to locate an EV with the longest battery range, although ties were ignored. This challenge involves locating the EV with the shortest battery range (again ignoring possible ties).

1. Create a list named `ev_range` that contains two elements. Assign default values to the elements as described below:

Index	data type	Description	Default value
0	<code>str</code>	EV " <code>< make > < model ></code> "	<code>None</code>
1	<code>int</code> <code>float</code>	EV range	A number that exceeds all listed EV range values

2. Loop over a slice of `elec_vehicles` that **excludes** the first nested list element (the "headers").

3. Access the vehicle "range" value and assign it to a variable (you choose the name) inside the `for` loop.

 the range value is a string and *must* be converted to a number.
4. Implement an `if` statement that checks whether or not the "current" EV's range **is less** than the "previous" range value referenced in `ev_range`.
5. If `True` assign the following "current" EV values to `ev_range`:
 - Element 0: EV "< make > < model >" f-string
 - Element 1: EV range value integer or float
6. Uncomment `print()` and check your work.

3.3 Challenge 06

Task: Compute the mean (average) EV "combined" mpge value.

The EV data set includes data on EV fuel economy distances for city, combined, and highway distances (e.g., "81/82/83"). Your job is to extract the **combined** fuel economy value (e.g., "82") embedded in the `city/combined/highway_mpge` string for each EV and add to an accumulator list. Then compute the mean by summing the numbers and then dividing by the total number of EVs surveyed. You will employ the built-in `round()` function to limit the fractional component of the mean to two (2) decimal places.

1. Create an empty list and assign it to the variable `combined_mpge_values`.
2. Loop over a slice of `elec_vehicles` that **excludes** the first nested list element (the "headers").
3. Employ the appropriate string method to extract the three mileage values into a list. Then access the "combined" (2nd element) and convert to an integer. Consider assigning the number to a variable (you choose the name) inside the `for` loop.
4. After extracting the "combined" value add it to the `combined_mpge_values` list.
5. After the loop terminates, compute the mean of the accumulated values referenced in `combined_mpge_values`. Perform this computation **outside** the loop block and assign the mean to the variable `combined_mpge_mean`.



Utilize the built-in function `sum()` to help you accomplish this task. See w3schools, "[The Python sum\(\) Function](#)" for usage details.



If you've forgotten how to compute the mean (average) of a group of numbers, see MathisFun, "[How to Find the Mean](#)".

6. After computing the mean, utilize the built-in function `round()` to limit the fractional component of the mean value to two (2) decimal places.



See w3schools, "[The Python round\(\) Function](#)" for usage details.



Steps 5-6 can be accomplished with a single line of code.

7. Uncomment `print()` and check your work.