SI 506 Lecture 08

Topics

1. `break` and `continue` statements
2. `range()` type
3. `while` loop (indefinite iteration)
4. Built-in `input()` function

Vocabulary

- **Argument**. A value passed to a function or method that corresponds to a parameter defined for the function or method.
- **Boolean**. A type (`bool`) or an expression that evaluates to either `True` or `False`.
- **Built-in Function**. A function defined by the Standard Library that is always available for use.
- **Conditional Statement**. A statement that determines a computer program's *control flow* or the order in which particular computations are to be executed.
- **Dictionary**. An associative array or a map, wherein each specified value is associated with or mapped to a defined key that is used to access the value.
- **Expression**. An accumulation of values, operators, and/or function calls that return a value. `len(< some_list >)` is considered an expression.
- **Function**. A defined block of code that performs (ideally) a single task. Functions only run when they are explicitly called. A function can be defined with one or more *parameters* that allow it to accept *arguments* from the caller in order to perform a computation. A function can also be designed to return a computed value. Functions are considered "first-class" objects in the Python eco-system.
- **f-string**. Formatted string literal prefixed with `f` or `F`.
- **Immutable**. Object state cannot be modified following creation. Strings are immutable.
- **Iterable**. An object capable of returning its members one at a time. Both strings and lists 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.
- **Method**. A function defined by and bound to an object. For example the `str` type is provisioned with a number of methods including `str.strip()`.
- **Mutable**. Object state can be modified following creation. Lists are mutable.
- **Operator**. A symbol for performing operations on values and variables. The assignment operator (`=`) and arithmetic operators (`+`, `-`, `*`, `/`, `**`, `%`, `//`).
- **Parameter**. A named entity in a function or method definition that specifies an argument that the function or method accepts.
- **Sequence**. An ordered set such as `str`, `list`, or `tuple`, the members of which (e.g., characters, elements, items) can be accessed.
- **Slice**. A subset of a sequence. A slice is created using the subscript notation `[]` with colons separating numbers when several are given, such as in `variable_name[1:3:5]`. The bracket notation uses slice objects internally.
- **Statement**. An instruction that the Python Interpreter can execute. For example, assigning a variable to a value such as `name = 'arwhyte'` is considered a statement.

- **Tuple**. An ordered sequence that cannot be modified once it is created.

1.0 `break` and `continue` statements

You can interrupt control flow inside a loop using the `break` and `continue` statements.

1.1 `break` statement

The `break` statement is employed in a `for` loop to exit the loop and proceed to the next statement in the code. Any statements inside the loop that follow the `break` statement will be ignored. The break statement is usually triggered by a specified condition. Using a `break` statement prevents unnecessary looping and can result in performance gains if the sequence being looped over is large.

For example, given a select list of model year 2021 electric passenger vehicles derived from the US Department of Energy's Fuel Economy how could you confirm programmatically whether or not the the Chinese battery and vehicle manufacturer *Kandi Technologies Group* is included in the `elec_vehicles` list?

```
elec_vehicles = [
    ['automaker', 'brand', 'model', 'year', 'range', 'range_hwy',
'range_city', 'highway_08_mpg', 'charge_240v_hrs'],
    ['Ford Motor Company', 'Ford', 'Mustang Mach-E AWD', 2021, 211, 193.7,
225.5, 86, 8.5],
    ['Kandi Technologies Group', 'Kandi', 'K27', 2021, 59, 51.6, 64.3,
102, 7.0],
    ['General Motors Co.', 'Chevrolet', 'Bolt EV', 2021, 259, 235.1,
277.7, 108, 9.3],
    ['Volkswagen AG', 'Audi', 'e-tron', 2021, 222, 221.9408, 222.74, 77,
10.0],
    ['Nissan Motor Co.', 'Nissan', 'Leaf (40 kW-hr battery pack)', 2021,
149, 131.3, 163.2, 99, 8.0],
    ['Tesla, Inc.', 'Tesla', 'Model 3 Performance AWD', 2021, 315, 299.0,
328.7, 107, 10.0],
    ['Volvo Group', 'Volvo', 'XC40 AWD BEV', 2021, 208, 188.0, 223.6, 72,
8.0],
    ['Volkswagen AG', 'Volkswagen', 'ID.4 1st', 2021, 250, 230.1587,
266.7659, 89, 7.5],
    ['Volvo Group', 'Polestar', '2', 2021, 233, 222.1, 241.9, 88, 8.0],
    ['Bayerische Motoren Werke AG', 'BMW', 'i3s', 2021, 153, 136.4, 166.5,
102, 7.0],
    ['Bayerische Motoren Werke AG', 'Mini', 'Cooper SE Hardtop 2 door',
2021, 110, 101.9, 116.9, 100, 4.0],
    ['Tesla, Inc.', 'Tesla','Model S Performance (19in Wheels)', 2021,
387, 373.2, 398.3, 106, 14.7]
]

headers = elec_vehicles[0] # column headers
```

One solution is to implement a `for` loop, checking for the existence of the automaker in the list and, if found, assign the boolean value `True` to a variable (e.g., `has_kandi`) initialized outside the loop. Once the variable assignment is made, a `break` statement is added in order to exit the loop and avoid unnecessary loop iterations (a performance gain).

```
has_kandi = False
for vehicle in elec_vehicles[1:]:
    if vehicle[0].lower() == 'kandi technologies group':
        has_kandi = True
        break # exit loop
```

1.2 Challenge

Implement a `for` loop that includes a statement block that confirms if any vehicles in the `elec_vehicles` list are capable of (re)charging their batteries within a three (3.0) hour period.

1. Setup
    1. Create a variable named `quick_charge` and assign it a float value of `3.0`.
    2. Create a second variable named `can_quick_charge` and assign it a value of `False`.
2. Inside the `for` loop write a conditional statement that checks if *any* of the vehicles in the `elec_vehicles` list can (re)charge their batteries in less than or equal to three (`3.0`) hours.
3. If such a vehicle is located assign `True` to `quick_charge` and then exit the loop *immediately*.

1.3 `continue` statement

The `continue` statement is employed in a `for` loop to end the current iteration and proceed directly to the next iteration in the loop (if any), skipping any trailing statements.

In the example below, the goal is to return a list of electric vehicles that represent "outliers" in terms of city driving battery range. If a vehicle's range is between 75 miles and 275 miles (exclusive) the `continue` statement is executed and the trailing `list.append()` operation is skipped. Only vehicles with a city range that falls on either side of the 75 - 275 mile range is added to the `outliers` list.

💡 Use comparison operators arranged as `x < y < z` or `x <= y <= z` to test if a value (typically a number but also a letter) is *between* two values. The expression returns either `True` or `False`.

```
outliers = []
for vehicle in elec_vehicles[1:]:
    city_range = vehicle[headers.index('range_city')]
    if 75.0 < city_range < 275.0:
        continue # proceed to next iteration (skip)
    outliers.append(vehicle)
```

1.4 Challenge

Given a tuple named `us_automakers` that contains the names of three US automakers, write a `for` loop that includes a statement block that assigns Asian and European automakers' vehicles in the `elec_vehicles` list to an empty list named `non_us_vehicles`.

1. Setup: note the following variable assignments:

   ```
   us_automakers = ('ford motor company', 'general motors co.', 'tesla,
   inc.') # tuple
   non_us_vehicles = []
   ```

2. Inside the `for` loop write a conditional statement that checks whether or not each vehicle's *manufacturer* is a listed in `us_automakers` . If `True` ignore the vehicle and check the next vehicle. If the conditional statement returns `False` assign the vehicle to the `non_us_vehicles` list.

💡 utilize the membership operator `in` to check whether or not a vehicle's manufacturer is a US automaker included in the `us_automakers` tuple.

2.0 range() data type

Although the Python official documentation includes `range()` in its table of built-in functions, the `range` object is actually an immutable sequence type like a `list` or a `tuple`.

The `range` object is employed to generate an *immutable* sequence of numbers. The Default behavior starts the sequence at 0 and then increments by 1 up to but *excluding* the specified stop value passed to the object as an argument. Optional start and step arguments can be passed to `range()` including negative step values that reverse the sequence.

```
range([start,] stop[, step])
```

```
seq = range(10) # instantiate the range object with a stop argument of 10

print(f"\n2.0.1 seq (type={type(seq)}) = {seq}") # <class 'range'>

seq = list(range(10)) # returns [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

seq = list(range(5, 10)) # returns [5, 6, 7, 8, 9]

seq = list(range(5, 21, 5)) # returns [5, 10, 15, 20]

seq = list(range(20, 4, -5)) # returns [20, 15, 10, 5]
```

Using a `for` loop in conjunction with `range()` provides a mechanism for specifying the number of loop iterations.

```
for i in range(10):
    print("I want to own an EV!")
```

You can use `range()` to iterate over a list:

```
automakers = [
    'Bayerische Motoren Werke AG',
    'Ford Motor Co.',
    'General Motors Co.',
    'Kandi Technologies Group',
    'Nissan Motor Co.',
    'Volkswagen AG',
    'Volvo Group',
    'Tesla, Inc.'
    ]

for i in range(len(automakers)):
    print(automakers[i])
```

2.1 Challenge

Use `range()` in conjunction with a `for` loop in order to assign *every other* vehicle in `elec_vehicles` to an accumulator list named `select_vehicles`.

💡 Consider breaking the problem down into sub-problems, solving each sub-problem in turn:

1. Use `range()` to print out all elements in `elec_vehicles`.
2. Adjust the `range()` start and stop arguments to return only the vehicles (skip the header).
3. Add a step argument to return every other vehicle.

3.0 Indefinite iteration: the `while` loop

The `while` loop repeats a set of one or more statements *indefinitely*; that is, until a condition is imposed that evaluates to `False` and terminates the loop.

```
while < expression >:
    < statement A >
    < statement B >
```

In the example below, a counter `i` is initialized with a default value of zero (`0`). The `while` loop, once initiated, will continue to iterate over the loop block *indefinitely* until the expression `i < 10` returns `False`. Note that the only way to terminate the looping operation is to increment the counter value by `1` _inside the loop block.

```
i = 0
while i < 5:
    print(i)
    i += 1 # increment (addition assignment operator)
```

3.1 Infinite loops

If a `while` loop is implemented incorrectly it will trigger an *infinite loop*, a runaway process that, over time, will consume ever greater memory resources to the detriment of your both your operatings system and hardware (you will hear the fans kick on as the laptop's internal temperature rises). Eventually, your system will crash unless you kill the process.

Typically, a `while` loop is implemented when the number of required iterations is unknown. There are other use cases, one of which we will explore below.

The following example is guaranteed to trigger an infinite loop since the `while` condition remains `True` indefinitely:

```
while True:
    print("infinite loop triggered") # Don't do this
```

You can tame a `while` loop condition initialized to `True` by adding a conditional statement that includes a `break` statement in its code block.

```
i = 0
while True:
    print('infinite loop triggered')
    if i == 5:
        print('infinite loop terminated\n')
        break # exit the loop
    i += 1 # increment (note indention)
```

❗ if you trigger an infinite loop while running your module in VS Code click the terminal pane's trash can icon in order to kill the session and end the runaway process.

3.2 `while` loop `else` condition

The `while` loop includes a built-in `else` condition that you can use to execute one or more statements after the loop terminates.

```
i = 0
while i < 5:
    print('I want an EV.')
    i += 1 # increment
else:
    print('Enough said. We believe you.')
```

3.3 `while` loop conditional statements

You can employ `if-elif-else` logic inside a `while` loop in order to determine the control flow of each iteration. In the following example the modulus (`%`) operator is used to identify even and odd numbers between 0 and 10.

💡 use the modulus operator to return the remainder after one number is divided by another. If the remainder equals zero the number evaluated is an even number.

```
i = 0
while i < 10:
    if i % 2 == 0:
        print(f"{i} is an even number.")
    else:
        print(f"{i} is an odd number.")
    i += 1 # increment

i = 10
while i >= 0:
    if i % 2 == 0:
        print(f"{i} is an even number.")
    else:
        print(f"{i} is an odd number.")
    i -= 1 # decrement
```

3.4 Challenge

The following data set was sourced from the National Renewable Energy Laboratory (NREL) Alternative Fuel Stations API. The API can be used to retrieve information on biodiesel, compressed natural gas, ethanol, electric charging, hydrogen, liquefied natural gas, and propane station locations in the US.

The data covers all electric charging stations located in Ann Arbor, MI. The `data` list comprise nested lists of charging station locations. Each nested list represents a modest subset of the information otherwise collected by the NREL. Each nested charging station list includes the following elements described in the "header" element:

- station name
- street address
- ev connector types (1 or more)
- ev network

```
data = [
    ['station_name', 'street_address', 'ev_connector_types',
'ev_network'],
    ['Ann Arbor Downtown Development Authority - Library Parking
Structure', '319 S Fifth Ave', 'J1772', 'Non-Networked'],
    ['Ann Arbor Downtown Development Authority - Ann Ashley Parking
Structure', '120 W Ann St', 'J1772', 'Non-Networked'],
    ['Ann Arbor Downtown Development Authority - Catherine and Fourth
Surface Lot', '121 Catherine St', 'J1772', 'Non-Networked'],
    ['Ann Arbor Downtown Development Authority - Forrest Parking
Structure', '650 Forrest St', 'J1772', 'Non-Networked'],
    ['Ann Arbor Downtown Development Authority - Maynard Parking
Structure', '316 Maynard St', 'J1772', 'Non-Networked'],
    ['Ann Arbor Downtown Development Authority - William Street Parking
Structure', '115 William St', 'J1772', 'Non-Networked'],
    ['U-M ANN ARBOR ANN STREET #2', '1101-1189 E Ann St', 'J1772',
'ChargePoint Network'],
    ['U-M ANN ARBOR ICL EDU #1', '1000 Greene St', 'J1772', 'ChargePoint
Network'],
    ['U-M ANN ARBOR WALGREEN #1', '1300 Murfin Ave', 'J1772', 'ChargePoint
Network'],
    ['BMW ANN ARBOR STATION 01', '501 Auto Mall Dr', 'J1772', 'ChargePoint
Network'],
    ['U-M ANN ARBOR WALL STREET #2', '1041 Wall St', 'J1772', 'ChargePoint
Network'],
    ["DOMINO'S FARMS DOMINO'S FARMS2", '24 Frank Lloyd Wright Dr',
'J1772', 'ChargePoint Network'],
    ['Ann Arbor Downtown Development Authority - Ashley and Washington
Parking Structure', '215 W Washington', 'J1772', 'Non-Networked'],
    ['MEADOWLARK BLDG STATION 2', '3250 W Liberty Rd', 'J1772',
'ChargePoint Network'],
    ['Shell', '2991 S State St', 'CHADEMO, J1772COMBO', 'eVgo Network'],
    ['Meijer - Tesla Supercharger', '3145 Ann Arbor-Saline Rd', 'TESLA',
'Tesla'],
    ['Sheraton Ann Arbor Hotel - Tesla Destination', '3200 Boardwalk Dr',
'J1772, TESLA', 'Tesla Destination'],
    ['MEIJER STORES 064 SALINE RD 1', '3145 Ann Arbor-Saline Rd',
'CHADEMO, J1772COMBO', 'ChargePoint Network'],
    ['U-M ANN ARBOR NCRC STATION 2', 'NCRC', 'J1772', 'ChargePoint
Network'],
    ['FLEET SERVICES CITY HALL STA 4', '301 E Huron St', 'J1772',
'ChargePoint Network'],
    ['173 - Ann Arbor', '5645 Jackson Road', 'CHADEMO, J1772COMBO',
'Greenlots'],
    ['Car & Driver - Tesla Destination', '1585 Eisenhower Place', 'TESLA',
'Tesla Destination'],
    ['U-M ANN ARBOR ANN STREET #1', '1101-1189 E Ann St', 'J1772',
'ChargePoint Network'],
    ['U-M ANN ARBOR ANN STREET #3', '1101-1189 E Ann St', 'J1772',
'ChargePoint Network'],
    ['U-M ANN ARBOR WALL STREET #1', '1041 Wall St', 'J1772', 'ChargePoint
Network'],
    ["DOMINO'S FARMS DOMINO'S FARMS", '24 Frank Lloyd Wright Dr', 'J1772',
'ChargePoint Network'],
    ['MEADOWLARK BLDG STATION 1', '3250 W Liberty Rd', 'J1772',
'ChargePoint Network'],
    ['MEIJER STORES 064 SALINE RD 2', '3145 Ann Arbor-Saline Rd',
'CHADEMO, J1772COMBO', 'ChargePoint Network'],
    ['U-M ANN ARBOR NCRC STATION 1', 'NCRC', 'J1772', 'ChargePoint
Network'],
    ['FLEET SERVICES POLICE SPACE #5', '301 E Huron St', 'J1772',
'ChargePoint Network'],
    ['BEEKMAN BEEKMAN ST1', '1200 Broadway St', 'J1772', 'ChargePoint
Network']
  ]
```

3.4.1 Challenge

Return a count of the total number of charging stations in Ann Arbor. Assign the value to a variable named `station_count`. Then count the number of U-M charging stations. Assign the U-M count to a variable named `umich_station_count`.

1. Access the "header row" in `data` and assign the return value to the variable named `headers`.
2. Access the charging stations in `data` and assign the return value to a variable named `charging_stations`.
3. Use a `while` loop and `range()` to accomplish the task.

   💡 Utilize the membership operator `in` when basing a `while` loop on a `range()` sequence.

4. Look up each charging station's "station_name" using `list.index()` rather than hard-coding the index value when crafting your `if` statement. Create a list of "header" elements to meet this requirement.

   💡 when working with nested lists you can "chain" index values in your expressions in order to access an element in the nested list as the following example illustrates:

```
chargepoint_network_count = 0
i = 0
while i < len(data[1:]):
    if data[i][-1] == 'ChargePoint Network': # index value chaining
        chargepoint_network_count += 1
    i += 1
```

3.4.2 Challenge

Convert the charging station names to *mixed case* while retaining the all caps `U-M` and `BMW` abbreviations. Doing so will regularize case usage across the charging station names.

1. Employ a `while` loop to convert the charging station names to mixed case while retaining the all caps `U-M` and `BMW` abbreviations.

❗ limit the number of `while` loop iterations to the length of the `charging_stations` list; increment the counter `i` inside the loop and use it to access each charging station in the list.

💡 The problem solution involves a number of `str` methods calls. In such cases I recommend breaking the problem down into sub-problems by treating each `str` method call as a sub-problem to be solved.

❗ The official abbreviation for the University of Michigan is "U-M" (not "UMich").

3.4.3 Challenge

If you review the data closely the "ev_connector_types" value is in certain cases a list masquerading as a string (e.g., `'CHADEMO, J1772COMBO'`). Convert this value to a list for all charging stations in the `charging_stations` list.

1. Employ a `while` loop to convert each charging station "ev_connector_type" string to a list (e.g., `'CHADEMO, J1772COMBO'` to a list `['CHADEMO', 'J1772COMBO']`.
2. Look up each charging station's "ev_connector_type" using `list.index()` rather than hard-coding the index value when crafting your `if` statement.

```
# TODO Implement
```

4.0 `while` loop and the built-in `input()` function

The built-in `input()` function accepts user-supplied strings from the command prompt. It is often positioned inside a `while` loop in order to process user input. The pattern is illustrated in the following example.

```
ev_automakers = (
    'Audi',
    'BMW',
    'Ford',
    'GM',
    'Hyundai',
    'Kandi',
    'Kia',
    'Jaguar',
    'Nissan',
    'Tesla',
    'Volkswagen',
    'Volvo'
    )
prompt = '\nWho is your favorite EV automaker?: '

while True:
    automaker = input(prompt)
    if automaker in ev_automakers:
        print(f"\nThanks for selecting {automaker}.\n\nFinis.")
        break # terminate loop
    print(f"\n'{automaker}' is not listed among the EV manufacturers.")
    print('Please check spelling and enter again or provide a different
automaker.')
```