Announcements

Reminder:
Program 5 – due tomorrow by 11:59 pm

Practice From Last Time

- Write a function that returns a Rhodes student email address. (Assume this email address is for a new student). Your function will need to take in 4 arguments: first name, last name, middle name and class year.
- Write a function called reverse that takes a string argument and returns the string argument will all characters in the reverse order.
  - Ex: reverse("Welsh") returns "hsleW"
- Re-write your total_time function from Friday so that it can now accept any number of minutes and seconds. It will still take in a string argument with two numbers separated by a colon. The function should return the total number of seconds in the time given.
  - Ex: total_time("1:40") returns 100
  - Ex: total_time("10:40") returns 640
  - Ex: total_time("123:456") returns 7836

Sequences

- Sequence: an object that contains multiple items of data
  - The items are stored in sequence one after another
- Python provides different types of sequences, including lists and tuples
  - The difference between these is that a list is mutable and a tuple is immutable
Introduction to Lists

- **List**: an object that contains multiple data items
  - **Element**: An item in a list
  - **Format**: `list = [item1, item2, etc.]`
  - Can hold items of different types

- **print** function can be used to display an entire list

- **list()** function can convert certain types of objects to lists

Example Using Lists

```python
def main():
    # Create a list with some items.
    food = ['Pizza', 'Burgers', 'Chips']

    # Display the list.
    print('Here are the items in the food list:')
    print(food)

    # Call the main function.
    main()
```

Program Output

Here are the items in the food list:
['Pizza', 'Burgers', 'Chips']

The Repetition Operator and Iterating over a List

- **Repetition operator**: makes multiple copies of a list and joins them together
  - The `*` symbol is a repetition operator when applied to a sequence and an integer
    - Sequence is left operand, number is right
  - General format: `list * n`
  - Works similarly to string – often used to initialize a list to all zeros

- You can iterate over a list using a `for` loop
  - Format: `for x in list:`
Indexing

- **Index**: a number specifying the position of an element in a list
  - Enables access to individual element in list
  - Index of first element in the list is 0, second element is 1, and n’th element is n-1
  - Negative indexes identify positions relative to the end of the list
    - The index -1 identifies the last element, -2 identifies the next to last element, etc.

The `len` function

- An `IndexError` exception is raised if an invalid index is used
- **len function**: returns the length of a sequence such as a list
  - Example: `size = len(my_list)`
  - Returns the number of elements in the list, so the index of last element is `len(list)-1`
  - Can be used to prevent an `IndexError` exception when iterating over a list with a loop

Lists Are Mutable

- **Mutable sequence**: the items in the sequence can be changed
  - Lists are mutable, and so their elements can be changed
- An expression such as `list[1] = new_value` can be used to assign a new value to a list element
  - Must use a valid index to prevent raising of an `IndexError` exception
Using the repetition operator to initialize list

Examples of Concatenation

```python
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)  # prints [1, 2, 3, 4, 5, 6]

mylist = ['a', 'b', 'c']
other = ['d', 'e', 'f']
print(mylist + other)  # ['a', 'b', 'c', 'd', 'e', 'f']
```

List Slicing

```python
# The NUM_DAYS constant holds the number of
# days we will gather sales data for.
NUM_DAYS = 5

def main():
    # Create a list to hold the sales.
    # For each day:
    sales = [0] * NUM_DAYS
    # Create a variable to hold an index.
    index = 0
    # Prompt the user for sales for each day.
    while index < NUM_DAYS:
        print(f'Day {index + 1}: Enter sales value: ', sep='', end='')
        sales[index] = float(input())
        index += 1
    # Display the sales entered.
    for value in sales:
        print(value)
    # Call the main function.
    main()
```

**Concatenating Lists**

- **Concatenate**: join two things together
- The `+` operator can be used to concatenate two lists
  - Cannot concatenate a list with another data type, such as a number
- The `+=` augmented assignment operator can also be used to concatenate lists

**Examples of Concatenation**

```python
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)  # prints [1, 2, 3, 4, 5, 6]

mylist = ['a', 'b', 'c']
other = ['d', 'e', 'f']
print(mylist + other)  # ['a', 'b', 'c', 'd', 'e', 'f']
```

**List Slicing**

- **Slice**: a span of items that are taken from a sequence
  - List slicing format: `list[start : end]`
  - Span is a list containing copies of elements from `start` up to, but not including, `end`
    - If `start` not specified, 0 is used for start index
    - If `end` not specified, `len(list)` is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of list
Examples of List Slices

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numbers[2:]  # [3, 4, 5, 6, 7, 8, 9, 10]
numbers[:2]  # [1, 2, 3, 4, 5, 6, 7, 8]
numbers[1:8:2] # [2, 4, 6, 8]
numbers[5::1] # [6, 5, 4, 3, 2, 1]
numbers[::-1] # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

Practice

Get the file Oct29.py from my Public directory. It has the main function written for you and stubs for 2 other functions that you will need to write.

findAverage(numbers) – will return the average of all the numbers in the list.

countNumbers(numbers, average) - will return 2 values; it counts the number of above average and below average numbers in a list.

Next Time

More Lists
Chapter 7 in your book