CN101

Lecture 5 (Part 1)

Lists and Tuples

Topics

- Sequences
- Introduction to Lists
- The Repetition Operator and Iterating over a List
- Indexing
- The `len` function
- Lists Are Mutable
- Concatenating Lists
- List Slicing
- The `sorted` function

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

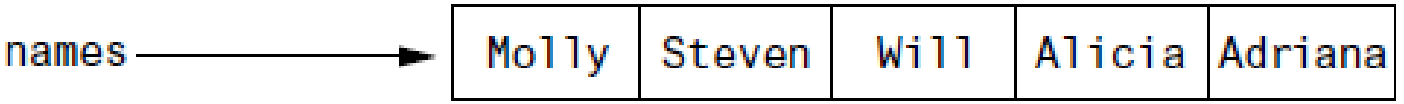
Introduction to Lists

- List: an object that contains multiple data items
  - Element: An item in a list
  - Format: `list1 = [item1, item2, etc.]`
  - Can hold items of different types
    - `list2 = [10, "Hello", 3.14, True]`

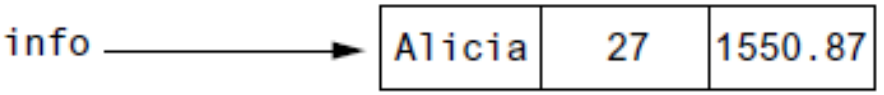
Introduction to Lists (cont'd.)

- Here is a statement that creates a list of integers:
  `even_numbers = [2, 4, 6, 8, 10]`

- The following is another example:
  `names = ['Molly', 'Steven', 'Will', 'Alicia', 'Adriana']`



- A list can hold items of different types, as shown in the following example: `info = ['Alicia', 27, 1550.87]`



Introduction to Lists (cont'd.)

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

```
>>> numbers = [5, 10, 15, 20]
>>> numbers
[5, 10, 15, 20]
```

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

```
>>> numbers = list(range(1, 10, 2))
>>> numbers
[1, 3, 5, 7, 9]
```

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: *list1 * n*

```
>>> numbers = [1, 2, 3] * 3
>>> numbers
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

The Repetition Operator and Iterating over a List (cont'd.)

- You can iterate over a list using a `for` loop
  - Format: `for` *`x`* `in` *`list1`*`:`

```
>>> numbers = [1, 2, 3, 4, 5]
>>> for n in numbers:
...     print(n)
...
1
2
3
4
5
```

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.
  - An `IndexError` exception is raised if an invalid index is used

Indexing (cont'd.)

```
>>> numbers = [1, 2, 3, 4, 5]
>>> print(numbers[0], numbers[2], numbers[4])
1 3 5
>>> print(numbers[-1], numbers[-3], numbers[-5])
5 3 1
```

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[1] + numbers[3]
6
>>> numbers[-1] * numbers[-5]
5
```

The `len` function

- `len` function: returns the length of a sequence such as a list
  - Example: `size = len(list1)`
  - Returns the number of elements in the list, so the index of last element is `len(list1)-1`
  - Can be used to prevent an `IndexError` exception when iterating over a list with a loop

The `len` function (cont'd.)

```
>>> numbers = [1, 2, 3, 4, 5]
>>> len(numbers)
5
```

```
>>> numbers = [1, 2, 3, 4, 5]
>>> for i in range(len(numbers)):
...     print(numbers[i])
...
1
2
3
4
5
```

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
  - `list1[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

Lists Are Mutable (cont'd.)

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers[1] = 20
>>> numbers
[1, 20, 3, 4, 5]
>>> numbers[3] *= 10
>>> numbers
[1, 20, 3, 40, 5]
>>> numbers[5] = 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Lists Are Mutable (cont'd.)

```
>>> numbers = [1, 2, 3, 4, 5]
>>> for i in range(len(numbers)):
...     numbers[i] *= 2
...
>>> numbers
[2, 4, 6, 8, 10]
```

sales_list.py

```
1  # The NUM_DAYS constant holds the number of
2  # days that we will gather sales data for.
3  NUM_DAYS = 5
4
5  # Create a list to hold the sales
6  # for each day.
7  sales = [0] * NUM_DAYS
8
9  # Create a variable to hold an index.
10 index = 0
11
12 print('Enter the sales for each day.')
13
14 # Get the sales for each day.
15 while index < NUM_DAYS:
16     sales[index] = float(input(f'Day #{index + 1}: '))
17     index += 1
18
19 # Display the values entered.
20 print('Here are the values you entered:')
21 for value in sales:
22     print(value)
```

Program Output

```
Enter the sales for each day.
Day #1: 1000
Day #2: 2000
Day #3: 3000
Day #4: 4000
Day #5: 5000
Here are the values you entered:
1000.0
2000.0
3000.0
4000.0
5000.0
```

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

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [5, 6, 7, 8]
>>> list3 = list1 + list2
>>> list3
[1, 2, 3, 4, 5, 6, 7, 8]
```

Concatenating Lists (cont'd.)

```
>>> girl_names = ['Joanne', 'Karen', 'Lori']
>>> girl_names += ['Jenny', 'Kelly']
>>> girl_names
['Joanne', 'Karen', 'Lori', 'Jenny', 'Kelly']
```

employees1.py

```python
# List of employee names
employees = ["Alice", "Bob"]
print(f"List of employees: {employees}")

# User input for a new employee
new_employee = input("Enter the name of the new employee: ")

# Adding the new employee using concatenation
employees = employees + [new_employee]

# Display the updated list of employees
print("List of employees after adding a new one:")
for index in range(len(employees)):
    print(f"{index + 1}. {employees[index]}")
```

Program Output

```
List of employees: ['Alice', 'Bob']
Enter the name of the new employee: Peter
List of employees after adding a new one:
1. Alice
2. Bob
3. Peter
```

employees2.py

```
1  # List of employee names
2  employees = ["Alice", "Bob", "Peter"]
3  print(f"List of employees: {employees}")
4
5  # User input for the name to search and the new name
6  old_name = input("Enter the name of the employee to replace: ")
7  new_name = input("Enter the new name: ")
8
9  # Searching for the name and replacing it
10 for index in range(len(employees)):
11     if employees[index] == old_name:
12         employees[index] = new_name
13         break
14
15 # Display the updated list of employees
16 print(f"Updated list of employees: {employees}")
```

Program Output

```
List of employees: ['Alice', 'Bob', 'Peter']
Enter the name of the employee to replace: Bob
Enter the new name: David
Updated list of employees: ['Alice', 'David', 'Peter']
```

List Slicing

- List slicing in Python is a technique used to extract a subset of elements from a list. It allows you to access a portion of the list by specifying a range of indices. The basic syntax for list slicing is:

```
list[start:stop:step]
```

  - start: The index where the slice begins (inclusive).
  - stop: The index where the slice ends (exclusive).
  - step: The step or stride between each element in the slice.

List Slicing (cont'd.)

Basic Slicing:

```
>>> list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[2:5]
[2, 3, 4]
```

Omitting Start and Stop:

```
>>> list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[:5]
[0, 1, 2, 3, 4]
>>> list1[5:]
[5, 6, 7, 8, 9]
>>> list1[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List Slicing (cont'd.)

Using Negative Indices:

```
>>> list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[-5:]
[5, 6, 7, 8, 9]
>>> list1[:-5]
[0, 1, 2, 3, 4]
```

Specifying a Step

```
>>> list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[0:9:2]
[0, 2, 4, 6, 8]
>>> list1[1::2]
[1, 3, 5, 7, 9]
```

List Slicing (cont'd.)

Using Negative Step:

```
>>> list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list1[-1::-2]
[9, 7, 5, 3, 1]
>>> list1[5:0:-1]
[5, 4, 3, 2, 1]
>>> list1[5:0:-2]
[5, 3, 1]
```

get_top_scores.py

```
1  # Define the scores list
2  scores = [98, 95, 93, 89, 87, 85, 80, 75]
3
4  # Get the number of top scores to retrieve from the user
5  num_top = int(input("Enter the number of top scores to retrieve: "))
6
7  # Retrieve the top scores
8  top_scores = scores[:num_top]
9
10 # Print the top scores
11 print(f"Top {num_top} scores: {top_scores}")
```

Program Output

```
Enter the number of top scores to retrieve: 2
Top 2 scores: [98, 95]
```

Program Output

```
Enter the number of top scores to retrieve: 5
Top 5 scores: [98, 95, 93, 89, 87]
```

employees3.py

```
1  employees = ["Alice", "Bob", "Peter"]
2  print(f"List of employees: {employees}")
3
4  # User input for the name to search and remove
5  name_to_remove = input("Enter the name of the employee to remove: ")
6
7  # Searching for the name and removing it
8  for index in range(len(employees)):
9      if employees[index] == name_to_remove:
10         employees = employees[:index] + employees[index+1:]
11         break
12
13 # Display the updated list of employees
14 print(f"Updated list of employees: {employees}")
```

Program Output

```
List of employees: ['Alice', 'Bob', 'Peter']
Enter the name of the employee to remove: Bob
Updated list of employees: ['Alice', 'Peter']
```

temperature.py

```
# Example temperature data for a month (30 days)
temperatures = [
    22.5, 23.0, 21.5, 22.0, 24.5, 25.0, 23.5,
    24.0, 26.5, 27.0, 25.5, 26.0, 28.5, 29.0,
    30.5, 28.0, 27.5, 28.0, 29.5, 30.0, 31.5,
    32.0, 33.5, 31.0, 30.5, 32.0, 33.5, 34.0,
    35.5, 36.0
]

# Get user input for operations
week_number = int(input("Enter the week number (1-4) to view temperatures: "))
num_hottest_days = int(input("Enter the number of hottest days to retrieve: "))
start_day = int(input("Enter the start day for average temperature calculation (1-30): "))
end_day = int(input("Enter the end day for average temperature calculation (1-30): "))

# Extract weekly temperatures
start = (week_number - 1) * 7
end = week_number * 7
weekly_temperatures = temperatures[start:end]
print(f"Temperatures for week {week_number}: {weekly_temperatures}")

```

temperature.py (cont'd.)

```
22 # Find the hottest days
23 hottest_days = sorted(temperatures, reverse=True)[:num_hottest_days]
24 print(f"The top {num_hottest_days} hottest days: {hottest_days}")
25
26 # Calculate average temperature for a specified period
27 period_temperatures = temperatures[start_day-1:end_day]
28 sum_temperatures = 0
29 for temperature in period_temperatures:
30     sum_temperatures += temperature
31 average_temperature = sum_temperatures / len(period_temperatures)
32 print(f"Average temperature from day {start_day} to day {end_day}:
   {average_temperature:.2f}°C")
33
```

Program Output

```
Enter the week number (1-4) to view temperatures: 2
Enter the number of hottest days to retrieve: 3
Enter the start day for average temperature calculation (1-30): 10
Enter the end day for average temperature calculation (1-30): 20
Temperatures for week 2: [24.0, 26.5, 27.0, 25.5, 26.0, 28.5, 29.0]
The top 3 hottest days: [36.0, 35.5, 34.0]
Average temperature from day 10 to day 20: 28.14°C
```

The `sorted` function

- The sorted() function in Python is used to return a new sorted list from the elements of any iterable (like a list, tuple, or string).

```
sorted(iterable, key=None, reverse=False)
```

Parameters

- **iterable**: The sequence (like a list, tuple, string, etc.) that you want to sort.
- **key** (optional): A function that serves as a key for the sort comparison. Defaults to None, which means the elements are compared directly.
- **reverse** (optional): A boolean value. If True, the sorted list is reversed (or sorted in descending order). Defaults to False.

sorted_function.py

```
1  numbers = [4, 2, 9, 1, 5, 6]
2  sorted_numbers = sorted(numbers)
3  print(sorted_numbers)
4
5  words = ["banana", "apple", "cherry", "date"]
6  sorted_words = sorted(words)
7  print(sorted_words)
8
9  numbers = [4, 2, 9, 1, 5, 6]
10 sorted_numbers_desc = sorted(numbers, reverse=True)
11 print(sorted_numbers_desc)
```

Program Output

```
[1, 2, 4, 5, 6, 9]
['apple', 'banana', 'cherry', 'date']
[9, 6, 5, 4, 2, 1]
```

product_prices.py

```python
# List of products and their prices
products = ["Laptop", "Smartphone", "Tablet", "Headphones", "Smartwatch"]
prices = [1200.00, 800.00, 400.00, 100.00, 200.00]

# Original products and prices
print("\nOriginal products and prices:")
for index in range(len(products)):
    print(f"{products[index]}: ${prices[index]:.2f}")

# User input for the product to modify and the new price
product_to_modify = input("\nEnter the name of the product to modify the price: ")
new_price = float(input(f"Enter the new price for {product_to_modify}: "))

# Searching for the product and updating its price using list slicing
found = False
for i in range(len(products)):
    if products[i] == product_to_modify:
        prices = prices[:i] + [new_price] + prices[i+1:]
        found = True
        break

if not found:
    print(f"Product named {product_to_modify} not found in the list.")

```

product_prices.py (cont'd.)

```
25 # Updated products and prices
26 print("\nUpdated products and prices:")
27 for index in range(len(products)):
28     print(f"{products[index]}: ${prices[index]:.2f}")
```

Program Output

```
Original products and prices:
Laptop: $1200.00
Smartphone: $800.00
Tablet: $400.00
Headphones: $100.00
Smartwatch: $200.00

Enter the name of the product to modify the price: Tablet
Enter the new price for Tablet: 350

Updated products and prices:
Laptop: $1200.00
Smartphone: $800.00
Tablet: $350.00
Headphones: $100.00
Smartwatch: $200.00
```