Lists

Sequence: an object storing multiple data items, one after another

List: a type of sequence that is a dynamic data structure

Python syntax: `list_variable= [value1, value2,value3,...]`

Example:

```
grade_list = [98, 76, 89, 90, 65, 86]
name_list = ['Mickey Mouse', 'Cinderella', 'Wall-E']
employee_list = ['Jon Smith', 65000.00, 24]
course_list = []
```

Accessing Elements

We can use `for` loops to iterate through the elements of a list.

```
grade_list= [98, 76, 89, 90, 65, 86]
for grade in grade_list:
    print(grade)

print(grade_list)
```

Indexing

Indexing – allows us to access individual elements of a list

1. Index specifies the position of an element in a list
2. Index of the first element is always 0
3. Index of the last element is always 1 less than the list length

Python syntax: `list_name[index]`

```
grade_list= [98, 76, 89, 90, 65, 86]

print(grade_list[0])
print(grade_list[5])
print(grade_list[6]) #Generates an IndexError exception

size = len(grade_list) #Returns the length of the list
```

Adding Elements to Lists

Lists are **dynamic** – we can add elements to a list while the program runs

```
#Adds elementto the end of the list
list_variable.append(element)

#Inserts elementin the indexposition;
# all elements after index move to their index + 1
list_variable.insert(index, element)
```

Example:

```
grade_list= [98, 76, 89, 90, 65, 86]
grade_list.append(78)
print(grade_list)
# Prints [98, 76, 89, 90, 65, 86, 78]

grade_list.insert(2, 84)
print(grade_list)
# Prints [98, 76, 84, 89, 90, 65, 86, 78]

print("The number of grades is", len(grade_list))
# Prints "The number of grads is 8"
```

List Slicing

List slicing – allows us to select a range of elements from a list

1. Start – the index of the first element of the slice (element included in slice)
2. End – the index of the end of the slice (element **NOT** included in slice)
3. Step – how to increase index value (like in `range()` function, is optional)

```
list_name[start:end:step]

# Returns elements from index 0 up to the end index
list_name[:end]
# Returns elements from the start index through the end of the list
list_name[start:]
```

Example:

```
grade_list= [98, 76, 89, 90, 65, 86]

print(grade_list[1:4])
print(grade_list[:3])
print(grade_list[3:])
print(grade_list[1:5:2])
```

Searching Lists

We can use the `in` operator to look for an item in a list

1. item – the value you are searching for
2. list_name – a list
3. Returns `True` if item is found in the list
4. Returns `False` if item is not found in the list

Python syntax : `item in list_name`

Example:

```
name_list = ["Belle","Gaston","Beast"]
name = input("Enter a character’s name: ")
if name in name_list:
    print(name, "was found in the list")
else:
    print(name, "was not found in the list")
```

Other Useful Functions

```
# Returns index of the first value of item in the list
list_name.index(item)

# Sorts items in the list in ascending order
list_name.sort()

# Removes first occurrence of item in the list
list_name.remove(item)

# Reverses the order of the items in the list
list_name.reverse()

# Returns the smallest value in the list
min(list_name)

# Returns the largest value in the list
max(list_name)

# Removes item at a specific position in the list
del list_name[index]
```

Processing Lists

1. List elements can be used in calculations
2. To calculate total of numeric values in a list use loop with accumulator variable
3. To average numeric values in a list:
    1. Calculate total of the values
    2. Divide total of the values by `len(list)`
4. List can be passed as an argument to a function
5. A function can return a reference to a list
6. To save the contents of a list to a file:
    1. Use the file object's writelinesmethod
        1. Does not automatically write `'\n'` at then end of each item
        2. Use a `for` loop to write each element and `'\n'`
    2. To read data from a file use the file object's `readlines` method

Two-Dimensional Lists

1. **Two-dimensional list**: a list that contains other lists as its elements
    1. Also known as nested list
    2. Common to think of two-dimensional lists as having rows and columns
    3. Useful for working with multiple sets of data

2. To process data in a two-dimensional list need to use two indexes
3. Typically use nested loops to process