Lists and Tuples

A sequence is an object that holds multiple items of data, stored one after the other. You can perform operations on a sequence to examine and manipulate the items stored in it.

Both lists and tuples are

Sequences that can hold various types of data. The difference between lists and tuples is simple: a list is mutable, which means that a program can change its contents, but a tuple is immutable, which means that once it is created, its contents cannot be changed.

Lists

A list is an object that contains multiple data items. Lists are mutable, which means that their contents can be changed during a program's execution. Lists are dynamic data structures, meaning that items may be added to them or removed from them. You can use indexing, slicing, and various methods to work with lists in a program

A list is an object that contains multiple data items. Each item that is stored in a list is called an element. Here is a statement that creates a list of integers:

even_numbers = [2, 4, 6, 8, 10]

list()

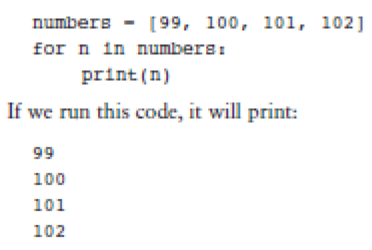
Python also has a built-in list() function that can convert certain types of objects to lists function

```
>>> no=list(range(5))

>>> no

[0, 1, 2, 3, 4]
```

Repetition Operator

```
>>> no*5

[0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]
```

Iterating over the List with the for loop



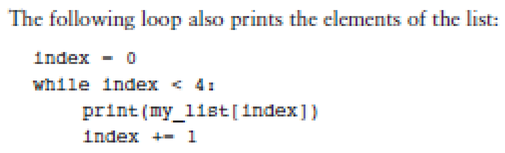
```
numbers = [99, 100, 101, 102]
for n in numbers:
    print(n)
```

If we run this code, it will print:

```
99
100
101
102
```

You can access the individual elements in a list is with an index. Each element in a list has an index that specifies its position in the list. Indexing starts at 0, so the index of the first element is 0, the index of the second element is 1, and so forth.

```
>>> mylist=[1,2,3,4]
>>> mylist[0]
1
```


The following loop also prints the elements of the list:

```
index = 0
while index < 4:
    print(my_list[index])
    index += 1
```

The len() function

```
my_list = [10, 20, 30, 40]
index = 0
while index < len(my_list):
   print(my_list[index])
   index += 1
```

Lists are mutable

Lists in Python are mutable, which means their elements can be changed. Consequently, an expression in the form list[index] can appear on the left side of an assignment operator.

```
numbers=[1,23,4,5,6,7]
>>> numbers[2]=10
>>> numbers
[1, 23, 10, 5, 6, 7]
```

Concatenating Lists

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

>>> girl_names = ['Joanne', 'Karen', 'Lori']
>>> boy_names = ['Chris', 'Jerry', 'Will']
>>> all_names = girl_names + boy_names
```

List Slicing

A slicing expression selects a range of elements from a sequence. Sometimes you want to select more than one element from a sequence. In Python, you can write expressions that select subsections of a sequence, known as slices.

```
list_name[start : end]
days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
mid_days = days[2:5]
```

When start index is not given

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

```
>>> print(numbers)
[1, 2, 3, 4, 5]
>>> print(numbers[:3])
[1, 2, 3]
```

When end index is not given

```
>>> numbers = [1, 2, 3, 4, 5] e
>>> print(numbers) e
[1, 2, 3, 4, 5]
>>> print(numbers[2:]) e
[3, 4, 5]
```

When both start and end indexes is not given

```
>>> numbers = [1, 2, 3, 4, 5] e
>>> print(numbers) e
[1, 2, 3, 4, 5]
>>> print(numbers[:])e
[1, 2, 3, 4, 5]
```

Step value slicing

```
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(numbers)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> print(numbers[1:8:2])
[2, 4, 6, 8]
```

Using –ve Sign to locate the index

```
>>> no=[1,2,3,4,5,6,7,8,9]
>>> no[-5:]
[5, 6, 7, 8, 9]
>>> no[-5:-1]
[5, 6, 7, 8]
>>> no[-3:-1]
[7, 8]
```

Finding Items in Lists with in Operator

You can search for an item in a list using the in operator. In Python you can use the in operator to determine whether an item is contained in a list. Here is the general format of an expression written with the in operator to search for an
item in a list:
item in list

```
# This program demonstrates the in operator
# used with a list.

def main():
    # Create a list of product numbers.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']

    # Get a product number to search for.
    search = input('Enter a product number: ')

    # Determine whether the product number is in the list.
    if search in prod_nums:
        print(search, 'was found in the list.')
    else:
        print(search, 'was not found in the list.')

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

Not in operator

```
if search not in prod_nums:
   print(search, 'was not found in the list.')
else:
   print(search, 'was found in the list.')
```

Some built in functions

Method	Description
`append(item)`	Adds *item* to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts *item* into the list at the specified *index*. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of *item* from the list. A `ValueError` exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

WAP to enter the items until users wants to

Hint:

```
# Append the name to the list.
 name_list.append(name)
```

WAP to check index of items in list

Hint :

```
food.index(item)
```