Lists

Lists

- A list is a sequence of items. Other programming languages refer to this structure as an array.
- To crete a list, here is the syntax:

  aList = [item1, item2, item3, ...]

  Examples,

  ```
  scores = [10, 8, 7, 6, 9]
  readings = [5.8, 7.2, 5,2]
  books = [‘C++’, ‘Java’, ‘PHP’, ‘Python’]
  tests = []
  ```
- Use an index to refer to the intel’s in the list. The index starts with 0.
- Use -1 to refer to the last item, -2 to refer to the second last item, and so on.

  ```
  scores = [10, 8, 7, 6, 9]
  print(scores[0], scores[-5])      # 10 10
  print(scores[1], scores[-4])      # 8 8
  print(scores[2], scores[-3])      # 7 7
  print(scores[3], scores[-2])      # 5 5
  print(scores[4], scores[-1])      # 2 2
  ```
- Use operator (*) to repeat items in a list.

  Examples

  ```
  >>> scores = [0] * 5
  >>> print(scores)
  [0, 0, 0, 0, 0]
  >>> scores = [2] * 5
  >>> print(scores)
  [2, 2, 2, 2, 2]
  ```

Adding and Removing Items

- The append(item) - appends the specified item.
- The insert(index, item) - inserts the specified item at the specified index.
- The remove(item) - removes the first item in the list that is equal to the specified item. If the item isn't found, this method raises a ValueError.
- index(item) - returns the index of the first occurrence of the specified item in the list. If not found, a ValueError is returned.
- pop([index]) - returns the item at the specified index and removes it. If no index is provided, the last item is returned and removed.

```
>>> scores = [10, 8, 7, 6, 9]
>>> scores.append(5)
>>> print(scores)
[10, 8, 7, 6, 9, 5]
>>> scores.insert(2,4)
>>> print(scores)
[10, 8, 4, 7, 6, 9, 5]
>>> scores.remove(4)
>>> print(scores)
[10, 8, 7, 6, 9, 5]
>>> print(scores.index(9))
4
>>> scores.pop(4)
9
>>> print(scores)
[10, 8, 7, 6, 5]
```

Iterating Items in a list

- The len(list) - returns the number of items in the list.

```
# getaverages.py

def get_average_while(list):
    sum = 0
    i = 0
    while i < len(list):
        sum += list[i]
        i += 1
    return sum

def get_average_for(list):
    sum = 0
    i = 0
    for item in list:
        sum += item
    return sum

def main():
    scores = [10, 7, 8, 9, 6]
    print("get_average_while(scores) = ", get_average_while(scores))
    print("get_average_for(scores) = ", get_average_for(scores))


if __name__ == "__main__":
    main()
```

```
get_average_while(scores) =  40
get_average_for(scores) =  40
```

Two-Dimensional List

- To create a two-dimensional list, you define list of lists that is a list where each item is a list. You can use two indexes to access an item in the list as row and column index.

```
# colleges.py

# create a college
college = []

# create three courses
java_course = []
java_course.append("Java Programming")
java_course.append("Henry")
java_course.append("3")
java_course.append("Tuesdays")

python_course = []
python_course.append("Python Programming")
python_course.append("Peter")
python_course.append("3")
python_course.append("Wednesdays")

network_course = []
network_course.append("Network Fundamental")
network_course.append("Chester")
network_course.append("3")
network_course.append("Mondays")

# add courses to college
college.append(java_course)
college.append(python_course)
college.append(network_course)

print(college)

# print the list of lists as a 2-D list
for program in college:
    for item in program:
        print(item, end=',')
    print()

print()
# print the list of lists as a 2-D list using indexes
row = 0
col = 0
while row < len(college):
    while col < len(college[row]):
        print(college[row][col], end=',')
        col += 1
    print()
    col = 0
    row += 1
```

Output

```
[['Java Programming', 'Henry', '3', 'Tuesdays'], ['Python
Programming', 'Peter', '3', 'Wednesdays'], ['Network Fundamental',
'Chester', '3', 'Mondays']]

Java Programming,Henry,3,Tuesdays,
Python Programming,Peter,3,Wednesdays,
Network Fundamental,Chester,3,Mondays,

Java Programming,Henry,3,Tuesdays,
Python Programming,Peter,3,Wednesdays,
Network Fundamental,Chester,3,Mondays,
```

Adding and Removing Items

- The append(item) - appends the specified item.
- The insert(index, item) - inserts the specified item at the specified index.
- The remove(item) - removes the first item in the list that is equal to the specified item. If the item isn't found, this method raises a ValueError.
- index(item) - returns the index of the first occurrence of the specified item in the list. If not found, a ValueError is returned.
- pop([index]) - returns the item at the specified index and removes it. If no index is provided, the last item is returned and removed.

```
>>> scores = [10, 8, 7, 6, 9]
>>> scores.append(5)
>>> print(scores)
[10, 8, 7, 6, 9, 5]
>>> scores.insert(2,4)
>>> print(scores)
[10, 8, 4, 7, 6, 9, 5]
>>> scores.remove(4)
>>> print(scores)
[10, 8, 7, 6, 9, 5]
>>> print(scores.index(9))
4
>>> scores.pop(4)
9
>>> print(scores)
[10, 8, 7, 6, 5]
```

More List’s Methods

- count(item) - returns the number of occurrences of an item in the list. If not found, it returns 0.
- reverse() - reverses the order of the items in the list.
- sort([key=function]) - sorts the items in place. The optional parameter “key” is a function to be called on each item before sorting.
- sorted(list[, key=function]) - returns a new list storing the sorted items of the original list. The optional parameter “key” is a function to be called on each item before sorting.
- min(list) - returns the minimum value in the list.
- max(list) - returns the maximum value in the list.
- choice(list) - returns a randomly selected item from the list.
- shuffle(list) - shuffles the items in the list on a random basis.
- deepcpy(list) - returns a deep copy of the list. A new list is a separate copy of the original list.

Examples of List’s Methods

```
import random
import copy     # deep_copy()

scores = [2, 4, 2, 6, 7, 8, 10, 2]
print('Original scores =', scores)

# demonstrate the count()
print('scores.count(2) =', scores.count(2))
print('scores.count(5) =', scores.count(5))

# demonstrate the reverse()
scores.reverse()
print('Reversed scores = ', scores)

# demonstrate the sort()
scores.sort()
print('Sorted scores = ', scores)

# demonstrate the sort(key)
fruits = ['Orange', 'pear', 'Apple', 'banana', 'Watermelon']
fruits.sort()
print('fruits.sort() = ', fruits)
fruits.sort(key=str.lower)
print('fruits.sort(key=str.lower) = ', fruits)

# demonstrate the sorted(list)
numbers = [6, 5, 3, 2, 7, 1]
numbers2 = sorted(numbers)
print('numbers =', numbers)
print('mumbers2 =', numbers2)

# demonstrate the min(), max(), choice(), shuffle()
print('min(numbers) =', min(numbers))
print('max(numbers) =', max(numbers))
print('random.choice(numbers) =', random.choice(numbers))
random.shuffle(numbers)
print('random.shuffle(numbers) =', numbers)

# demonstrate the deep_copy
num = [6, 5, 3, 2, 7, 1]
num_copy = copy.deepcopy(num)
print('num =', num)
print('num_copy =', num_copy)
num_copy.sort()
print('num_copy.sort() =', num_copy)
print('num =', num)
```

Output

```
Original scores = [2, 4, 2, 6, 7, 8, 10, 2]
scores.count(2) = 3
scores.count(5) = 0
Reversed scores =  [2, 10, 8, 7, 6, 2, 4, 2]
Sorted scores =  [2, 2, 2, 4, 6, 7, 8, 10]
fruits.sort() =  ['Apple', 'Orange', 'Watermelon',
'banana', 'pear']
fruits.sort(key=str.lower) =  ['Apple', 'banana',
'Orange', 'pear', 'Watermelon']
numbers = [6, 5, 3, 2, 7, 1]
mumbers2 = [1, 2, 3, 5, 6, 7]
min(numbers) = 1
max(numbers) = 7
random.choice(numbers) = 3
random.shuffle(numbers) = [7, 6, 3, 1, 5, 2]
num = [6, 5, 3, 2, 7, 1]
num_copy = [6, 5, 3, 2, 7, 1]
num_copy.sort() = [1, 2, 3, 5, 6, 7]
num = [6, 5, 3, 2, 7, 1]
```

Tuples

- Tuples like lists can store multiple items.
- To create a tuple, you use parentheses ()
- Unlike lists, tuples are immutable. So, you can’t add, remove, or set items.
- You can unpack the values of a tuple into multiple variables.
- Tuples are more efficient than lists because they are immutable. So, if you don’t need to change the items inside the list, you should use tuples instead of lists.

```
def divide(x, y):
    q = x // y
    r = x % y
    return q, r

def main():
    num = (6, 5, 7)

    #num.append(8)       # AttributeError: 'tuple' object has no attribute 'append'
    #num.sort()          # AttributeError: 'tuple' object has no attribute 'sort'
    #num[0] = 10         # TypeError: 'tuple' object does not support item assignment
    print('num =', num)
    print('num[1:3]', num[1:3])

    x, y, z = num
    print('x =', x)
    print('y =', y)
    print('z =', z)

    quotient, remainder = divide(5, 2)
    print('quotient =', quotient)
    print('remainder =', remainder)

if __name__ == "__main__":
    main()
```

Output

```
num = (6, 5, 7)
num[1:3] (5, 7)
x = 6
y = 5
z = 7
quotient = 2
remainder = 1
```

Mutable vs Immutable Objects in Python

- Mutable vs Immutable Objects in Python
- Mutable vs Immutable Objects in Python – A Visual and Hands-On Guide