Chapter 6

How to work with lists and tuples

© 2016, Mike Murach & Associates, Inc.

Objectives

Applied

1. Use lists in your programs.
2. Use lists of lists in your programs.
3. Use tuples in your programs.

Knowledge

1. Describe how an item in a list is accessed.
2. Describe the use of these list methods: append(), insert(), remove(), index(), and pop().
3. Distinguish between the way mutable types like a list are passed to and returned by functions and the way immutable types like integers are passed to and returned by functions.

© 2016, Mike Murach & Associates, Inc.

Objectives

4. Describe the use of a list of lists.
5. Describe the use of these functions with lists: count(), reverse(), sort(), min(), max(), choice(), shuffle(), and deepcopy().
6. Differentiate between a shallow copy of a list and a deep copy.
7. Distinguish between a tuple and a list.
8. Describe the use of a multiple assignment statement when you unpack a tuple.

© 2016, Mike Murach & Associates, Inc.

The syntax for creating a list

```
mylist = [item1, item2, ...]
```

Code that creates lists

```
temps = [48.0, 30.5, 20.2, 100.0, 42.0] # 5 float values
inventory = ["staff", "hat", "shoes"]   # 3 str values
movie = ["The Holy Grail", 1975, 9.99]  # str, int, float
test_scores = []                        # an empty list
```


© 2016, Mike Murach & Associates, Inc.

How to use the repetition operator (*) to create a list

```
scores = [0] * 5          # test scores = [0, 0, 0, 0, 0]
```


© 2016, Mike Murach & Associates, Inc.

The temps list

```
temps = [48.0, 30.5, 20.2, 100.0, 42.0]
```

Its positive and negative index values

```
temps[0]    temps[-5]      # returns 48.0
temps[1]    temps[-4]      # returns 30.5
temps[2]    temps[-3]      # returns 20.2
temps[3]    temps[-2]      # returns 100.0
temps[4]    temps[-1]      # returns 42.0
```


© 2016, Mike Murach & Associates, Inc.

How to get an item in a list

Code that gets items from the temps list

```
temps = [48.0, 30.5, 20.2, 100.0, 42.0]
temp = temps[0]          # temp = 48.0
temp = temps[4]          # temp = 42.0
temp = temps[5]          # IndexError: index out of range
```

Code that gets items from the inventory list

```
inventory = ["staff", "hat", "shoes",
             "bread", "potion", "scroll"]
item = inventory[5]      # item = "scroll "
item = inventory[3]      # item = "bread"
item = inventory[6]      # IndexError: index out of range
```

How to set an item in a list

```
temps[3] = 98.0          # replaces 100.0 with 98.0
inventory[4] = "ration" # replaces "potion" with "ration"
```


© 2016, Mike Murach & Associates, Inc.

List methods for modifying a list

```
append(item)
insert(index, item)
remove(item)
index(item)
pop([index])
```


© 2016, Mike Murach & Associates, Inc.

The append(), insert(), and remove() methods

```
stats = [48.0, 30.5, 20.2, 100.0]

inventory = ["staff", "hat", "shoes",
             "bread", "potion"]

stats.append(99.5)       # [48.0, 30.5, 20.2, 100.0, 99.5]

inventory.insert(3, "robe") # ["staff", "hat", "shoes",
                            #  "robe", "bread", "potion"]

inventory.remove("shoes")   # ["staff", "hat", "robe",
                            #  "bread", "potion"]
```


© 2016, Mike Murach & Associates, Inc.

The pop() method

```
inventory = ["staff", "hat", "robe", "bread"]

item = inventory.pop()  # item = "bread"
                        # inventory = ["staff", "hat", "robe"]

item = inventory.pop(1) # item = "hat"
                        # inventory = ["staff", "robe"]
```


© 2016, Mike Murach & Associates, Inc.

The index() and pop() methods

```
inventory = ["staff", "hat", "robe", "bread"]

i = inventory.index("hat")  # 1

inventory.pop(i)            # ["staff", "robe", "bread"]
```


© 2016, Mike Murach & Associates, Inc.

A built-in function for getting the length of a list

```
len(list)
```


© 2016, Mike Murach & Associates, Inc.

How to use the in keyword to check whether an item is in a list

```
inventory = ["staff", "hat", "bread", "potion"]

item = "bread"
if item in inventory:
    inventory.remove(item)  # ["staff", "hat", "potion"]
```


© 2016, Mike Murach & Associates, Inc.

How to print a list to the console

```
inventory = ["staff", "hat", "shoes", "bread", "potion"]
print(inventory)
```

The console

```
['staff', 'hat', 'shoes', 'bread', 'potion']
```


© 2016, Mike Murach & Associates, Inc.

The syntax for looping through a list

```
for item in list:
    statements
```

Code that prints each item in a list

```
inventory = ["staff", "hat", "shoes"]
for item in inventory:
    print(item)
```

The console

```
staff
hat
shoes
```


© 2016, Mike Murach & Associates, Inc.

How to process the items in a list

With a for loop

```
scores = [70, 80, 90, 100]
total = 0
for score in scores:
    total += score
print(total)              # 340
```

With a while loop

```
scores = [70, 80, 90, 100]
total = 0
i = 0
while i < len(scores):
    total += scores[i]
    i += 1
print(total)              # 340
```


© 2016, Mike Murach & Associates, Inc.

Four immutable types

```
str
int
float
bool
```

One mutable type

```
list
```


© 2016, Mike Murach & Associates, Inc.

How to work with immutable arguments

The double_the_number() function

```
def double_the_number(value):
    value = value * 2    # new int object created
    return value         # new int object must be returned
```

The calling code in the main() function

```
value1 = 25              # int object created
value2 = double_the_number(value1)
print(value1)            # 25
print(value2)            # 50
```


© 2016, Mike Murach & Associates, Inc.

How to work with mutable arguments

The add_to_list() function

```
def add_to_list(list, item):
    list.append(item)          # list object changed
```

The calling code in the main() function

```
# list object created
inventory = ["staff", "hat", "bread"]

add_to_list(inventory, "robe")
print(inventory)       # ["staff", "hat", "bread", "robe"]

# NOTE: no need to return list object
```

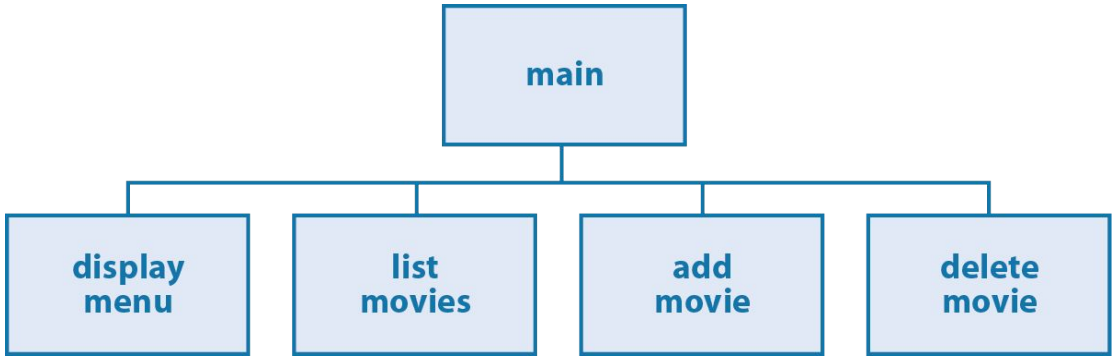

© 2016, Mike Murach & Associates, Inc.

The user interface for the Movie List program

```
COMMAND MENU
list - List all movies
add  - Add a movie
del  - Delete a movie
exit - Exit program

Command: list
1. Monty Python and the Holy Grail
2. On the Waterfront
3. Cat on a Hot Tin Roof

Command: add
Name: Casablanca
Casablanca was added.

Command: list
1. Monty Python and the Holy Grail
2. On the Waterfront
3. Cat on a Hot Tin Roof
4. Casablanca

Command: del
Number: 4
Casablanca was deleted.

Command: list
1. Monty Python and the Holy Grail
2. On the Waterfront
3. Cat on a Hot Tin Roof
```


© 2016, Mike Murach & Associates, Inc.

The hierarchy chart



© 2016, Mike Murach & Associates, Inc.

The code

```
def display_menu():
    print("COMMAND MENU")
    print("list - List all movies")
    print("add  - Add a movie")
    print("del  - Delete a movie")
    print("exit - Exit program")
    print()

def list(movie_list):
    i = 1
    for movie in movie_list:
        print(str(i) + ". " + movie)
        i += 1
    print()

def add(movie_list):
    movie = input("Name: ")
    movie_list.append(movie)
    print(movie + " was added.\n")
```


© 2016, Mike Murach & Associates, Inc.

The code (cont.)

```
def delete(movie_list):
    number = int(input("Number: "))
    if number < 1 or number > len(movie_list):
        print("Invalid movie number.\n")
    else:
        movie = movie_list.pop(number-1)
        print(movie + " was deleted.\n")

def main():
    movie_list = ["Monty Python and the Holy Grail",
                  "On the Waterfront",
                  "Cat on a Hot Tin Roof"]

    display_menu()
```


© 2016, Mike Murach & Associates, Inc.

The code (cont.)

```
    while True:
        command = input("Command: ")
        if command.lower() == "list":
            list(movie_list)
        elif command.lower() == "add":
            add(movie_list)
        elif command.lower() == "del":
            delete(movie_list)
        elif command.lower() == "exit":
            break
        else:
            print("Not a valid command. Please try again.\n")

    print("Bye!")

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


© 2016, Mike Murach & Associates, Inc.

How to define a list of lists…

With 3 rows and 4 columns

```
students = [["Joel", 85, 95, 70],
            ["Anne", 95, 100, 100],
            ["Mike", 77, 70, 80, 85]]
```

With 3 rows and 3 columns

```
movies = [["The Holy Grail", 1975, 9.99],
          ["Life of Brian", 1979, 12.30],
          ["The Meaning of Life", 1983, 7.50]]
```


© 2016, Mike Murach & Associates, Inc.

How to add to a list of lists through programming

```
movies = [["The Holy Grail", 1975, 9.99],
          ["Life of Brian", 1979, 12.30]

movie = []                                  # Create empty movie
movie.append("The Meaning of Life") # Add name to movie
movie.append(1983)                          # Add year to movie
movie.append(7.5)                           # Add price to movie
movies.append(movie)                        # Add movie to movies
```


© 2016, Mike Murach & Associates, Inc.

How to access the items in the list of movies

```
movies[0][0]    # "The Holy Grail"
movies[0][2]    # 9.99
movies[0][3]    # IndexError: index out of range
movies[1][0]    # "Life of Brian"
movies[3][0]    # IndexError: index out of range
```


© 2016, Mike Murach & Associates, Inc.

How to print a two-dimensional list

```
print(movies)
```

The console

```
[['The Holy Grail', 1975, 9.99], ['Life of
Brian', 1979, 12.3], ['The Meaning of Life',
1983, 7.5]]
```


© 2016, Mike Murach & Associates, Inc.

How to loop through the rows and columns of a 2-dimensional list

```
for movie in movies:
    for item in movie:
        print(item, end=" | ")
    print()
```

The console

```
The Holy Grail | 1975 | 9.99 |
Life of Brian | 1979 | 12.3 |
The Meaning of Life | 1983 | 7.5 |
```


© 2016, Mike Murach & Associates, Inc.

The user interface for the Movie List 2D program

```
COMMAND MENU
list - List all movies
add -  Add a movie
del -  Delete a movie
exit - Exit program

Command: list
1. Monty Python and the Holy Grail (1975)
2. On the Waterfront(1954)
3. Cat on a Hot Tin Roof (1958)

Command: add
Name: Gone with the Wind
Year: 1939
Gone with the Wind was added.

Command: list
1. Monty Python and the Holy Grail (1975)
2. On the Waterfront (1954)
3. Cat on a Hot Tin Roof (1958)
4. Gone with the Wind (1939)

Command: del
Number: 2
On the Waterfront was deleted.

Command: list
1. Monty Python and the Holy Grail (1975)
2. Cat on a Hot Tin Roof (1958)
3. Gone with the Wind (1939)
```


© 2016, Mike Murach & Associates, Inc.

The code

```
def list(movie_list):
    if len(movie_list) == 0:
        print("There are no movies in the list.\n")
        return
    else:
        i = 1
        for row in movie_list:
            print(str(i) + ". " + row[0] + " ("
                + str(row[1]) + ")")
            i += 1
        print()

def add(movie_list):
    name = input("Name: ")
    year = input("Year: ")
    movie = []
    movie.append(name)
    movie.append(year)
    movie_list.append(movie)
    print(movie[0] + " was added.\n")
```


© 2016, Mike Murach & Associates, Inc.

The code (cont.)

```
def delete(movie_list):
    number = int(input("Number: "))
    if number < 1 or number > len(movie_list):
        print("Invalid movie number.\n")
    else:
        movie = movie_list.pop(number-1)
        print(movie[0] + " was deleted.\n")

def display_menu():
    print("COMMAND MENU")
    print("list - List all movies")
    print("add -  Add a movie")
    print("del -  Delete a movie")
    print("exit - Exit program")
    print()
```


© 2016, Mike Murach & Associates, Inc.

The code (cont.)

```
def main():
    movie_list = [["Monty Python and the Holy Grail", 1975],
                  ["On the Waterfront", 1954],
                  ["Cat on a Hot Tin Roof", 1958]]

    display_menu()

    while True:
        command = input("Command: ")
        if command == "list":
            list(movie_list)
        elif command == "add":
            add(movie_list)
        elif command == "del":
            delete(movie_list)
        elif command == "exit":
            break
        else:
            print("Not a valid command. Please try again.\n")
    print("Bye!")

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


© 2016, Mike Murach & Associates, Inc.

Three more list methods

```
count(item)
reverse(list)
sort([key=function])
```

A built-in function

```
sorted(list[, key=function])
```


© 2016, Mike Murach & Associates, Inc.

The count(), reverse(), and sort() methods

```
numlist = [5, 15, 84, 3, 14, 2, 8, 10, 14, 25]

count = numlist.count(14)  # 2
numlist.reverse()  # [25, 14, 10, 8, 2, 14, 3, 84, 15, 5]
numlist.sort()     # [2, 3, 5, 8, 10, 14, 14, 15, 25, 84]
```


© 2016, Mike Murach & Associates, Inc.

The sort() method with mixed-case lists

```
foodlist = ["orange", "apple", "Pear", "banana"]
```

What happens in a simple sort

```
foodlist.sort()
# ["Pear", "apple", "banana", "orange"]
```

How to use the key argument to fix the sort order

```
foodlist.sort(key=str.lower)
# ["apple", "banana", "orange", "Pear"]
```


© 2016, Mike Murach & Associates, Inc.

The sorted() function with mixed-case lists

```
foodlist = ["orange", "apple", "Pear", "banana"]
```

What happens in a simple sort

```
sorted_foodlist = sorted(foodlist)
print(sorted_foodlist)
# ["Pear", "apple", "banana", "orange"]
```

How to use the key argument to fix the sort order

```
sorted_foodlist = sorted(foodlist, key=str.lower)
print(sorted_foodlist)
# ["apple", "banana", "orange", "Pear"]
```


© 2016, Mike Murach & Associates, Inc.

Two more built-in functions

```
min(list)
max(list)
```

Two functions of the random module

```
choice(list)
shuffle(list)
```


© 2016, Mike Murach & Associates, Inc.

How to use the min() and max() functions

```
numlist = [5, 15, 84, 3, 14, 2, 8, 10, 14, 25]
minimum = min(numlist)                  # 2
maximum = max(numlist)                  # 84
```

How to use the choice() and shuffle() functions

```
import random
numlist = [5, 15, 84, 3, 14, 2, 8, 10, 14]
choice = random.choice(numlist) # gets random item
random.shuffle(numlist)         # shuffles items randomly
```


© 2016, Mike Murach & Associates, Inc.

The deepcopy() function

```
deepcopy(list)
```

How to make a shallow copy of a list

```
list_one = [1, 2, 3, 4, 5]
list_two = list_one
list_two[1] = 4
print(list_one)     # [1, 4, 3, 4, 5]
print(list_two)     # [1, 4, 3, 4, 5]
```

How to make a deep copy of a list

```
import copy
list_one = [1, 2, 3, 4, 5]
list_two = copy.deepcopy(list_one)
list_two[1] = 4
print(list_one)     # [1, 2, 3, 4, 5]
print(list_two)     # [1, 4, 3, 4, 5]
```


© 2016, Mike Murach & Associates, Inc.

How to slice a list

The syntax for slicing a list

```
mylist[start:end:step]
```

Code that slices with the start and end arguments

```
numbers = [52, 54, 56, 58, 60, 62]
numbers[0:2]    # [52, 54]
numbers[:2]     # [52, 54]
numbers[4:]     # [60, 62]
```

Code that slices with the step argument

```
numbers[0:4:2]      # [52, 56]
numbers[::-1]       # [62, 60, 58, 56, 54, 52]
```


© 2016, Mike Murach & Associates, Inc.

How to concatenate two lists with the + and += operators

```
inventory = ["staff", "robe"]
chest = ["scroll", "pestle"]

combined = inventory + chest
# ["staff", "robe", "scroll", "pestle"]

print(inventory)
# ["staff", "robe"]

inventory += chest
# ["staff", "robe", "scroll", "pestle"]

print(inventory)
# ["staff", "robe", "scroll", "pestle"]
```


© 2016, Mike Murach & Associates, Inc.

How to create a tuple

```
mytuple = (item1, item2, ...)
```

Code that creates tuples

```
# a tuple of 5 floating-point numbers
stats = (48.0, 30.5, 20.2, 100.0, 48.0)

# a tuple of 6 strings
herbs = ("lavender", "pokeroot", "chamomile",
         "valerian", "nettles", "oatstraw")

# a tuple that stores the data for a movie
movie = ("Monty Python and the Holy Grail", 1975, 9.99)
```


© 2016, Mike Murach & Associates, Inc.

Code that accesses items in a tuple

```
herbs[0]    # lavender
herbs[-1]   # oatstraw
herbs[1:4]  # ('pokeroot', 'chamomile', 'valerian')

herbs[1] = "red clover"
# TypeError: 'tuple' object does not support
# item assignment
```


© 2016, Mike Murach & Associates, Inc.

Code that unpacks a tuple

```
tuple_values = (1, 2, 3)
a, b, c = tuple_values          # a = 1, b = 2, c = 3
```


© 2016, Mike Murach & Associates, Inc.

A function that returns a tuple

```
def get_location():
    # code that computes values for x, y, and z
    return x, y, z
```

Code that calls the get_location() function and unpacks the returned tuple

```
x, y, z = get_location()
```


© 2016, Mike Murach & Associates, Inc.

The user interface for the Number Cruncher program

```
TUPLE DATA: (0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50)
Average = 25 Median = 25 Min = 0 Max = 50 Dups = []

RANDOM DATA: [4, 6, 19, 22, 26, 29, 29, 39, 42, 45, 47]
Average = 28 Median = 29 Min = 4 Max = 47 Dups = [29]
```


© 2016, Mike Murach & Associates, Inc.

The code

```
import random

def crunch_numbers(data):
    total = 0
    for number in data:
        total += number

    average = round(total / len(data))
    median_index = len(data) // 2
    median = data[median_index]
    minimum = min(data)
    maximum = max(data)
    dups = get_duplicates(data)

    print("Average =", average,
          "Median =", median,
          "Min =", minimum,
          "Max =", maximum,
          "Dups =", dups)
```


© 2016, Mike Murach & Associates, Inc.

The code (cont.)

```
def get_duplicates(data):
    dups = []
    for i in range(51):
        count = data.count(i)
        if count >= 2:
            dups.append(i)
    return dups

def main():
    fixed_tuple = (0,5,10,15,20,25,30,35,40,45,50)
    random_list = [0] * 11
    for i in range(len(random_list)):
        random_list[i] = random.randint(0, 50)
    random_list.sort()

    print("TUPLE DATA:", fixed_tuple)
    crunch_numbers(fixed_tuple)
    print()
    print("RANDOM DATA:", random_list)
    crunch_numbers(random_list)

# if started as the main module, call the main() function
if __name__ == "__main__":
    main()
```


© 2016, Mike Murach & Associates, Inc.