Chapter 6

How to work with lists and tuples

Objectives (part 1)

Applied

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

Knowledge

4. Describe how an item in a list is accessed.
5. Describe the use of these methods for modifying a list: append(), insert(), remove(), index(), and pop().
6. Describe the use of the enumerate() and zip() methods for processing the items in a list.
7. Describe the use of these methods for working with lists: map(), filter(), list(), and reduce().

Objectives (part 2)

5. Explain how to use list comprehensions to create a list from another list.
6. 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.
7. Describe the use of a list of lists.
8. Describe the use of these functions with lists: count(), reverse(), sort(), min(), max(), sum(), choice(), shuffle(), and deepcopy().
9. Differentiate between a shallow copy of a list and a deep copy.
10. Distinguish between a tuple and a list.
11. Describe the use of a multiple assignment statement when you unpack a tuple.

Knowledge objectives (part 2)

The syntax for creating a list

```
list_name = [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
```

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

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

The temps list

```
temps = [48.0, 30.5, 20.2, 100.0, 42.0]
Index =    0      1      2      3      4
Length  = len(temps) = 5
```

Its positive and negative index values

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

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"
```

Methods for modifying a list

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

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]
                          # appends at the end

inventory.insert(3, "robe") # ["staff", "hat", "shoes",
                            #  "robe", "bread", "potion"]
                               # needs position & what to
insert

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

The pop() method

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

item = inventory.pop()   # item = "bread"
                         # inventory = ["staff", "hat", "robe"]
                    # pops the last one by default

item = inventory.pop(1) # item = "hat"
                         # inventory = ["staff", "robe"]
                    # pops the (1+1=2nd) item from current list
```

The index() and pop() methods

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

i = inventory.index("hat")    # 1,
               # gives index position based on the item

inventory.pop[i]     # ["staff", "robe", "bread"]
               # popping based on the index position
```

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

```
len(list)
```

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"]
```

How to print a list to the console

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

The console

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

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
```

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
```

Four immutable types -- can’t be changed

```
str
int
float
bool
```

One mutable type -- can be changed

```
List ☾ objects can be added, removed etc..
```

Two built-in functions for processing list items

```
enumerate(list, [start=0]) # list item one by one
zip(list1, list2, ...) # zips list1, list2 correspondingly

x = ['apple', 'banana', 'cherry']
y = enumerate(x)
print(list(y)) Ⓟ [(0, 'apple'), (1, 'banana'), (2, 'cherry’)]

 a = ("John", "Charles", "Mike")
 b = ("Jenny", "Christy", "Monica")
 x = zip(a, b)
 Print(list(x))
 ☾ [('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica')]
```

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
```

How to get a counter value when processing the items in a list (part 1)

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

Using a counter variable

```
i = 1
for item in inventory:
    print(f"{i}. {item}")
    i += 1
```

Using the value returned by the range() function

```
for i in range(len(inventory)):
    item = inventory[i]
    print(f"{i + 1}. {item}")
```

Using the value returned by the enumerate() function

```
for i, item in enumerate(inventory, start=1):
   print(f"{i}. {item}")
```

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
```

How to get a counter value when processing the items in a list (part 2)

The console for all three examples

```
1. staff
2. hat
3. bread
4. potion
```

How to process two lists in parallel

```
inventory = ["staff", "hat", "bread", "potion"]
prices = [27.99, 10.99, 5.99, 19.99]

for item, price in zip(inventory, prices):
    print(f"{item} (${price})")
```

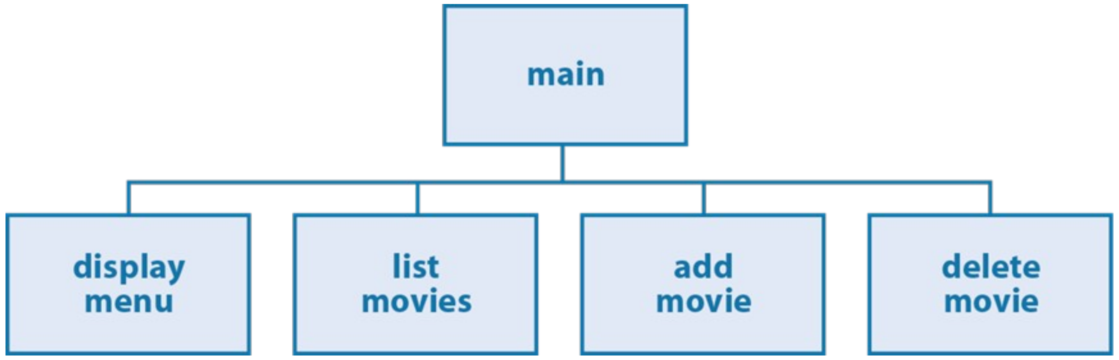
The console

```
staff ($27.99)
hat ($10.99)
bread ($5.99)
potion ($19.99)
```

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
```

The hierarchy chart for the Movie List program



The code for the Movie List program (part 1)

```
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):
    for i, movie in enumerate(movie_list, start=1):
        print(f"{i}. {movie}")
    print()

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

The code for the Movie List program (part 2)

```
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(f"{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()
```

The code for the Movie List program (part 3)

```
    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()
```

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, 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]]
```

How to add to a list of lists

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

# Create movie list
movie = ["The Meaning of Life", 1983, 7.5]

# Add movie list to movies list
movies.append(movie)
```

How to access the items in the list of movies

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

```
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
```

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]]
```

How to loop through the rows and columns of a two-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 |
```

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)
```

The code for the Movie List 2D program (part 1)

```
def list(movie_list):
    if len(movie_list) == 0:
        print("There are no movies in the list.\n")
    else:
        for i, movie in enumerate(movie_list, start=1):
            print(f"{i}. {movie[0]} ({movie[1]})")
    print()

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

The code for the Movie List 2D program (part 2)

```
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(f"{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()
```

The code for the Movie List 2D program (part 3)

```
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.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()
```

Three more list methods

```
count(item)                  # nbr of items in the list

reverse(list)    # reverses the list

sort([key=function]) # sorts the items in the list
```

A built-in function

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

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]
```

The sort() method with mixed-case lists

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

What happens in a simple sort

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

How to use the key argument to fix the sort order

```
foodlist.sort(key=str.lower)  #sorting the lower letter first
print(foodlist)  # ["apple", "banana", "orange", "Pear"]
```

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"]
```

Three more built-in functions for use with lists

```
min(list)
max(list)
sum(list[, start])
```

A list that's used in the following examples

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

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 sum() function

```
total = sum(numlist)                    # 180
total = sum(numlist, start=100)         # 280
```

Two functions of the random module for use with lists

```
choice(list) # gets one random item from list

shuffle(list) # shuffles items randomly
```

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

```
import random
numlist = [5, 15, 84, 3, 14, 2, 8, 10, 14, 25]
choice = random.choice(numlist) # gets random item
random.shuffle(numlist)         # shuffles items randomly
```

The deepcopy() function

```
deepcopy(list)
```

```
# Assignment statement doesn't copy object but creates a
new variable for binding between target & object. In order
to create a real object, copy (deep & swallow) is used
```

How to make a shallow copy of a list

```
list_one = [1, 2, 3, 4, 5]
list_two = list_one    #changes list_one also
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) #doesn't change list_one
list_two[1] = 4
print(list_one)                  # [1, 2, 3, 4, 5]
print(list_two)                  # [1, 4, 3, 4, 5]
```

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] #1st thru 3rd in step-2
numbers[::-1] # [62, 60, 58, 56, 54, 52] #reverses list
```

```
  b = "Hello, World!"
  print(b[2:5]) # position 2 to 5(not incl)
#llo
  print(b[:5]) # start to pos-5(not incl)
#Hello
  print(b[2:]) # pos-2 to end #llo, World!"
lo, World!
```

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"]
```

Three more built-in functions for use with lists

```
X = map(func, list)    # apply function = func to the list
list(X)                # gives the list on which func was applied
Y = filter(func,list)  # apply function = func to the list, choose
                       # the func logistically
```

A list that's used in the following examples

```
numlist = [1, 2, 3, 4, 5, 6]
```

How to use the map() and list() functions

```
def square(n):
    return n * n
X = map(square, numlist)  # map is used to all the elements
print(list(X))            # [1, 4, 9, 16, 25, 36]
```

How to use the filter() and list() functions

```
def is_even(n):
    return n % 2 == 0
Y = filter(is_even, numlist) # filter is used on SOME object, not all
Print(list(Y))                # evens is [2, 4, 6]
```

A function of the functools module

```
reduce(function, list[, start])
```

How to use the reduce() function

```
import functools as fn # lib needed for reduce function

numlist = [1, 2, 3, 4, 5, 6]

def addendo(x,y): return x+y
def maxendo(x,y): return x if x > y else y

print("The sum of the list elements is : ", end="")
print(fn.reduce(addendo, numlist))    # prints 17

print("The maximum element of the list is : ", end="")
print(fn.reduce(maxendo, numlist))    # max = 6
```

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to -- previously attained result and nbr just succeeding the 2nd element and the result is again stored.
- This process continues till no more elements are left in the container.

Basic syntax for a list comprehension

```
newlist = [expression for item in list [if condition]]
```

A list of numbers used by the following examples

```
numbers = [1, 2, 3, 4, 5, 6]
```

A loop that creates a list of squares

```
squares = []
for n in numbers:
    squares.append(n * n)    # squares is [1, 4, 9, 16, 25, 36]
```

A list comprehension that creates a list of squares

```
squares = [n * n for n in numbers] #above 2-lines can be written as one
                         # squares is [1, 4, 9, 16, 25, 36]
```

A list comprehension that uses a conditional expression for filtering

```
numbers = [1, 2, 3, 4, 5, 6]

even_squares = [n * n for n in numbers if n % 2 == 0]
# even_squares is [4, 16, 36]

# putting a condition in the list itself to cut-down coding
```

A list comprehension that calls functions

```
numbers = [1, 2, 3, 4, 5, 6]

def square(n):
    return n * n

def is_even(n):
    return n % 2 == 0

even_squares = [square(n) for n in numbers if is_even(n)]
# even_squares is [4, 16, 36]
```

A list comprehension that uses an assignment expression

```
import random

def get_number():
    return random.randrange(1, 10)

squares = [square(num) for n in range(10) if (
    num := get_number()) <= 6]
# squares is [4, 9, 1, 36, 16, 16, 4]
```

- Can be done otherwise

How to create a tuple

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

TUPLE	LIST
Uses () to store data	Uses [] to store data
Can't be changed	Can be changed (mutable)
Uses Less memory	More
Bit Faster	Slower
Stores Hetero/Homo-generous data	Stores Homo-generous data

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)
```

Code that accesses items in a tuple

```
herbs = ("lavender", "pokeroot", "chamomile",
         "valerian", "nettles", "oatstraw")

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

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

Code that unpacks a tuple

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

A function that returns multiple value thru 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()

def get_rect_area_perim(len, wid):
    area = len * wid
    perim = 2*(len+wid)
    return area, perim

ar, prm = get_rect_area_perim(12, 8)
print(ar, prm)     # 96 40
```

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 = []

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

The Number Cruncher program (part 1)

```
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)
```

The Number Cruncher program (part 2)

```
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)  # generating tuple data
    random_list = [0] * 11   #generating list data using random
    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("LIST DATA:", random_list)
    crunch_numbers(random_list)

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