Lecture 6

Tuples and Lists

Burkay Genç, Ahmet Selman Bozkır, and Selma Dilek

05/04/2023

PREVIOUS LECTURE

- functions
- recursion

TODAY

- introduce new compound data types
  - tuples
  - lists

TUPLES

Tuples

- Tuples are an ordered sequence of elements, can mix element types
- cannot change element values, **immutable**
- represented with **parentheses**

```
te = ()   # an empty tuple
t = (2, "mit", 3)   # a tuple with three elements
t
```

```
## (2, 'mit', 3)
```

Accessing Tuple Elements

```
t = (1, 2, 3, 4, "burkay", "genc")
t[3]
```

```
## 4
```

```
t[2:4]
```

```
## (3, 4)
```

```
t[3:4]
```

```
## (4,)
```

- The last one returns a tuple with **one** element.
  - The "," is included to make it a tuple.
- Be very careful
  - `t[3]` and `t[3:4]` are not the same

Tuple Operations

- concatenate and multiply

```
t1 = (1, 2, 3)
t2 = ("a", "b", "c")
t3 = t1 + t2
t3
```

```
## (1, 2, 3, 'a', 'b', 'c')
```

```
3 * t1
```

```
## (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

- can only concatenate tuples to tuples

```
t1 + 4
```

```
## TypeError: can only concatenate tuple (not "int") to tuple
```

Tuple Operations

- can nest tuples

```
t1 = ("burkay", "genc", 41)
t2 = (t1, "married", 2)
t2
```

```
## (('burkay', 'genc', 41), 'married', 2)
```

```
t2[0][1]
```

```
## 'genc'
```

Tuples Are Immutable

- Like strings, **tuples are immutable**
  - Immutable: Cannot be changed

```
t = ("dr", "burkay", "genc")
t[0] = "prof."
```

```
## TypeError: 'tuple' object does not support item assignment
```

- You should construct a new tuple as follows:

```
t = ("dr", "burkay", "genc")
t2 = ("prof",) + t[1:3]
t2
```

```
## ('prof', 'burkay', 'genc')
```

Tuples And Functions

- Tuples can be used as **arguments** and **return** values of functions

```
def foo(t):
  return 2*t

foo((1,2,3))
```

```
## (1, 2, 3, 1, 2, 3)
```

Exercise

- Write a function that takes a tuple and returns the maximum element within the tuple.

Exercise

- Write a function that takes a tuple and returns the maximum element within the tuple.

```
def max(t):
  maximum = t[0]
  for item in t:
    if item > maximum:
      maximum = item
  return maximum

tup = (3, -2, 5, 7, 8, -4, 1, 3, 1, -5)
max(tup)
```

```
## 8
```

Exercise

- Write a function that takes a tuple and computes the sum of items in the tuple.

Exercise

- Write a function that takes a tuple and computes the sum of items in the tuple.

```
def sum(t):
  curSum = 0
  for item in t:
    curSum = curSum + item
  return curSum

tup = (3, -2, 5, 7, 8, -4, 1, 3, 1, -5)
sum(tup)
```

```
## 17
```

Exercise

- Write a function that returns the **mean** of a tuple of numbers.
  - mean: the sum of items divided by the number of items

Exercise

- Write a function that returns the **mean** of a tuple of numbers.
  - mean: the sum of items divided by the number of items

```
def mean(t):
  return sum(t) / len(t)   # Reusing sum() from previous exercise

tup = (3, -2, 5, 7, 8, -4, 1, 3, 1, -5)
mean(tup)
```

```
## 1.7
```

Home Exercise

- Write a function that returns the sum of two rationals.
  - The rationals are provided as tupples

```
def sumRational(r1, r2):
  ....

sumRational( (3, 5), (2, 4) )
```

```
## (22,20)
```

LISTS

Lists

- **ordered sequence** of information, accessible by index
- a list is denoted by **square brackets**, `[]`
- a list contains **elements**
  - usually homogeneous (ie, all integers)
  - can contain mixed types (not common, use tuples)
- list elements can be changed so a list is **mutable**
  - main difference between tuples and lists

Accessing List Elements

```
a_list = []
L = [2, 'a', 4, [1,2]]
len(L)                  # evaluates to 4
```

```
## 4
```

```
L[0]                    # evaluates to 2
```

```
## 2
```

```
L[2]+1                  # evaluates to 5
```

```
## 5
```

```
L[3]                    # evaluates to [1,2], another list!
```

```
## [1, 2]
```

```
i = 2
L[i-1]                  # evaluates to ‘a’ since L[1]='a' above
```

```
## 'a'
```

List Operations

```
li1 = [1,2,3]
li2 = [4,5,6]
li1 + li2
```

```
## [1, 2, 3, 4, 5, 6]
```

```
3 * li1
```

```
## [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Changing Elements

- Elements of a list can be changed

```
L = [2, 1, 3]
L[1] = 5
L
```

```
## [2, 5, 3]
```

Exercise

- Write a function that takes a list and returns the maximum element within the list.

Exercise

- Write a function that takes a list and returns the maximum element within the list.

```
def max(li):
  maximum = li[0]
  for item in li:
    if item > maximum:
      maximum = item
  return maximum

li = [3, -2, 5, 7, 8, -4, 1, 3, 1, -5]
max(li)
```

```
## 8
```

Exercise

- Write a function that takes a list of numbers and replaces all occurrences of the maximum value with -1.

Exercise

- Write a function that takes a list of numbers and replaces all occurrences of the maximum value with -1.

```
def replaceMax(li):
  maxVal = max(li)     # reusing the function from the previous slide
  for i in range(len(li)):
    if li[i] == maxVal:
      li[i] = -1       # because lists are mutable, we can do this
  return li

myList = [4, 9, 3, 2, 0, 5, 4, 9, 8, 5, 1, 7, 9]
replaceMax(myList)
```

```
## [4, -1, 3, 2, 0, 5, 4, -1, 8, 5, 1, 7, -1]
```

Exercise

- Write a function that takes a list of numbers and computes the value $3x^2 - 2x + 1$ for each x in the list and finally returns all computed values as a list.

Exercise

- Write a function that takes a list of numbers and computes the value $3x^2 - 2x + 1$ for each x in the list and finally returns all computed values as a list.

```
def foo(li):
  result = []
  for num in li:
    result = result + [3 * num ** 2 - 2 * num + 1]
  return result

foo([1,2,3,4,5,6,7,8,9,10])
```

```
## [2, 9, 22, 41, 66, 97, 134, 177, 226, 281]
```

A Note On Objects Methods

- In Python, some structures are called **Objects**
  - We will later learn more about them
- Objects store some **data** in them
- They also have **special functions** in them
  - You can use those functions using the dot (.) notation:

```
object.function()
```

Additional List Operations

- `append` an object to the end of the list

```
li1 = [1, 2, 3]
li2 = [4, 5, 6]
li1.append(li2)
li1
```

```
## [1, 2, 3, [4, 5, 6]]
```

- `extend` a list with elements from another list

```
li1 = [1, 2, 3]
li2 = [4, 5, 6]
li1.extend(li2)
li1
```

```
## [1, 2, 3, 4, 5, 6]
```

- `count` an element in a list

```
li1 = [1, 2, 3, 4, 2, 4, 1, 3, 2, 5]
item = 2
li1.count(item)
```

```
## 3
```

Additional List Operations

- `insert` an object into a list at a specific position

```
li1 = [1, 2, 3]
item = 4
li1.insert(1, item)
li1
```

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

- `remove` the first occurrence of an item

```
li1 = [1, 2, 3, 2, 3, 1]
item = 3
li1.remove(item)
li1
```

```
## [1, 2, 2, 3, 1]
```

- find the `index` of the first occurence of an item

```
li1 = [1, 2, 3, 4, 2, 4, 1, 3, 2, 5]
item = 4
li1.index(item)
```

```
## 3
```

Additional List Operations

- `pop` the item at a specific position
  - If no position is given, then pops the last item

```
li1 = [1, 2, 3, 4, 5]
li1.pop(3)
```

```
## 4
```

```
li1.pop()
```

```
## 5
```

```
li1
```

```
## [1, 2, 3]
```

Additional List Operations

- `sort` the list

```
li1 = [4, 2, 3, 1, 3, 5]
li1.sort()
li1
```

```
## [1, 2, 3, 3, 4, 5]
```

- use `sorted` to not mutate the list and return a new list

```
li1 = [4, 2, 3, 1, 3, 5]
sorted(li1)
```

```
## [1, 2, 3, 3, 4, 5]
```

```
li1
```

```
## [4, 2, 3, 1, 3, 5]
```

- `reverse` the list

```
li1 = [1, 2, 3, 4, 2, 4, 1, 3, 2, 5]
li1.reverse()
li1
```

```
## [5, 2, 3, 1, 4, 2, 4, 3, 2, 1]
```

Exercise

- Given lists `l1` and `l2`, return a list of their common items

Exercise

- Given lists `l1` and `l2`, return a list of their common items
- **strategy**
  - Initialize *result* as an empty list
  - *Iterate* over the items in the *first list*
    - For each item, check if it *exists* in the *second list*
      - If so, then *add* it to the *result*
  - *return* result

Exercise

- Given lists `l1` and `l2`, return a list of their common items

```
def intersect(l1, l2):
  l3 = []
  for item in l1:
    if l2.count(item) > 0:
      l3.extend(item)
  return l3
```

- Is this a correct solution?
  - Do we need to refine this code?
  - Let's test

Exercise

- Given lists `l1` and `l2`, return a list of their common items

```
def intersect(l1, l2):
  l3 = []
  for item in l1:
    if l2.count(item) > 0:
      l3.append(item)
  return l3

l1 = [1,2,3,4,5,6,1,2,3,4,1,2,3]
l2 = [2,3,4]
intersect(l1, l2)
```

```
## [2, 3, 4, 2, 3, 4, 2, 3]
```

- How to fix this?
  - Each item must appear only once

Exercise

- Given lists `l1` and `l2`, return a list of their common items

```
def intersect(l1, l2):
  l3 = []
  for item in l1:
    if l2.count(item) > 0 and l3.count(item) == 0:
      l3.append(item)
  return l3

l1 = [1,2,3,4,5,6,1,2,3,4,1,2,3]
l2 = [2,3,4]
intersect(l1, l2)
```

```
## [2, 3, 4]
```

Optional In-class Exercises

- Write a function to compute the median element in a given list. The median is the middle item in a sorted list.
  - Strategy : sort the list, find the index of the middle item, return it
- Write a function that takes the top-left and bottom-right coordinates of a rectangle as two tuples. The function then returns the area of the rectangle.
  - Strategy : Find the width, find the height, multiply.
- [Difficult] Write a function (or more) that returns the most frequent character in a string.
  - Strategy : Use a second list that keeps binary tuples, each tuple is a char, frequency pair. You store the char frequencies in the corresponding tuple.
  - This exercise prepares the student to the dictionary concept of the next week.

Homework Exercises

- Write a phone book application.
  - The phone book is basically **a list of tuples**.
  - Each tuple in the list is `(name, surname, phonenumber)`
- You will write the following functions:
  - `addContact(pb, contact)`: adds `contact` tuple into `pb` phonebook
  - `removeContact(pb, contact)`: finds and removes `contact` from `pb`
  - `checkNumber(pb, (name, surname))`: checks the provided `(name,surname)` in the `pb` and **outputs** the corresponding phone number if found in `pb`. If the `(name, surname)` tuple is not found in the phone book, then the application outputs `name surname not found.`

Homework Exercises

- BONUS exercise
  - Make sure addContact does not add duplicate contact information
  - Make sure removeContact does not crash, if the contact does not exist in the phonebook
- DOUBLE BONUS exercise
  - Can you modify your program so that one (name, surname) tuple can have multiple phone numbers? Modify addContact, removeContact, checkNumber accordingly.

Copyright Information