Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Fall 2022
Week 12

Let's Discuss

Key Questions

How do I use the mathematical concepts of **sets** and **Boolean logic** to design Python programs that are easier to implement and understand?

Learning Objectives

To **remember** and **understand** some concepts about the **set**,exploring how its use can simplify the implementation of programs.

ALLEGHENY COLLEGE

Lists in Python

Lists, similar to arrays, are collections which are ordered and changeable.

Creating lists with append maintains position information

```
myList_list = []
myList_list #or print(myList_list)
   # []
myList_list.append("x")
myList_list.append("x")   # again
myList_list   # ['x', 'x']
```

Creating lists in entirety

```
myList_list = ["a","b","c","d"]
myList_list #or print(myList_list)
   #['a', 'b', 'c', 'd']
type(myList_list)
   #<class 'list'>
```

- With a `list`, position of character is maintained, not so with a `set`.

Lists in Python

Removing an element

```
myList_list = ["a"]
print(myList_list)
   #   ['a']
myList_list.remove("a")
print(myList_list)
   #   []
```

Reverse the entire list, no assignment necessary

```
myList_list = ["a","b","c","d"]
myList_list.reverse()
myList_list  #or print(myList_list)
   # ['d', 'c', 'b', 'a']
```

Lists in Python

Each element has a location

```
myList_list = ["a","b","c","d"]
myList_list[0]  # 'a'
myList_list[3]  # 'd'
myList_list[300] #IndexError
```

Print each element by location

```
for i in range(len(myList_list)):
   print("index = ",i)
   print("   myList_list[i] = ",myList_list[i])
#   index =  0
#   myList_list[i] =  a
#       ...
#   index =  3
#   myList_list[i] =  d
```

Iterating Through Elements in Lists

Iteration

```
l_list = ["a","b","c","d"]
for i in l_list:
  print(i)
```

Iteration

```
l_list = ["a","b","c","d"]
for i in range(len(l_list)):
  print("i = ",i," and l_list[i] = ",l_list[i])
```

Note

- With lists, we know which element will be printed first (the first element, from above).

Lambda Functions

We will use these to create lists ...

Lambda function definition

- The lambda operator or lambda function is a way to create small anonymous functions (i.e. functions without a name), and are *throw-away* functions

General syntax

lambda argument_list: expression

```
g = lambda x: 3*x + 1
g(2) #   7
```

```
sum = lambda x, y : x + y
sum(3,4) #   7
```

List Comprehensions to build lists

List comprehensions definition

- List comprehensions provide a concise way to create lists (or sets)

General syntax

[expression for item in list if conditional]

Make list

```
[i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Assign list to variable

```
b_list = [i for i in range(10)]
type(b_list)
<class 'list'>
```

List Comps and Lambda Functions to build lists

Build a list with an anonymous function

```
g_list = lambda x: list(i for i in range(x))
g_list(4)   #   [0, 1, 2, 3]
myList_list = g_list(4)
myList_list #   [0, 1, 2, 3]
# slicing particular elements
myList_list[0:2] #   [0, 1]
```

Tuples

A Tuple is a collection of Python objects separated by commas

An empty tuple

```
empty_tuple = ()
print (empty_tuple)
type(empty_tuple)   # <class 'tuple'>
```

A non-empty tuple

```
nonEmpty_tuple = ("a","b","c","d")
nonEmpty_tuple[0]   #   'a'
nonEmpty_tuple[len(nonEmpty_tuple)-1]   #   'd'
```

Check to see that elements are in a tuple

```
nonEmpty_tuple  #  ('a', 'b', 'c', 'd', 4, 'Hi')
"Hi" in nonEmpty_tuple   #  True
4 in nonEmpty_tuple   # True
3 in nonEmpty_tuple   # False
```

Tuples

Checking for sub-elements of elements at a tuple location

```
nonEmpty_tuple = ("a","b","c","d", 4, "Hi", "My music")
nonEmpty_tuple
   #  ('a', 'b', 'c', 'd', 4, 'Hi', 'My music')
"my" in nonEmpty_tuple   # False
"My" in nonEmpty_tuple   # False

# check to see if detail is in a substring in tuple
"My" in nonEmpty_tuple[6]  # True
```

Adding to Tuples

Convert tuple to list, add element, convert back

```
a_tuple = ('2',) #define Tuple
items = ['a', 'b', 'c', 'd'] # elements to add
l_list = list(a_tuple)# make a list
for x in items:
   l_list.append(x) # add items to list
#output as a tuple
print(tuple(l_list))
```

Iterating Through Elements in Tuples

Iteration

```
nonEmpty_tuple = ("a","b","c","d", 4, "Hi", "My music")
for i in nonEmpty_tuple:
  print(i)
```

Iteration

```
for i in range(len(nonEmpty_tuple)):
  print("i= ",i, "nonEmpty_tuple[i]=",nonEmpty_tuple[i])
```

Note

- With tuples (like lists), we know which element will be printed first (the first element, from above).

Dictionaries

An array of a key and a value that is connected for quick searching

- A dictionary maps a set of objects (keys) to another set of objects (values).
- A Python dictionary is a mapping of unique keys to values.
- Dictionaries are mutable, which means they can be changed.
- The values that the keys point to can be any Python value

An empty dictionary

```
myDictionary_dict = {}
print (myDictionary_dict)
type(myDictionary_dict)   # <class 'dict'>
```

Dictionaries

Adding to a dictionary

```
myDictionary_dict = {}
myDictionary_dict[0] = "zero"
myDictionary_dict[0] # gives 'zero'

myDictionary_dict[1] = "one"
print (myDictionary_dict)  #{1: 'one', 0: 'zero'}
```

Removing elements from a dictionary

```
myDictionary_dict = {}
myDictionary_dict[3] = "three"

del myDictionary_dict[3]
print (myDictionary_dict)  #{} (is empty)
```

Randomly Choosing Elements

Choosing Elements from a List

```
import random
abc_list = ['a','b','c','d','e']
random.choice(abc_list)    # 'c'
random.choice(abc_list)    # 'd'
```

Choosing Elements from a List

```
import random
abc_set = set(['a','b','c','d','e'])
  # convert to list
abc2_list = list(abc_set)
random.choice(abc2_list)   # 'd'
```

Randomly Choosing Elements

Choosing Elements from a Dictionary

```
import random
abc_dict = {1:"one",2:"two",3:"Three"} # {vals : keys}
num_list = list(abc_dict) # convert dict to list
n = random.choice(num_list) # pick a number in list
abc_dict[n]  # sub in n to get key value
  #  'two'
```

How to use lists

```
import random
aliceVocab_list = ["I like cats", "I like dogs",
"I like rabbits", "I gave carrots to horses",
"I live on a farm"]

# choose random element
aliceSays_str = random.choice(aliceVocab_list)
print("  This is Alice. I say to Bob :", aliceSays_str)

bobVocab_list = ["I have two cats", "I have three dogs",
"I know several rabbits","I love carrots",
"I love horses","I also live on a farm"]

bobSays_str = random.choice(bobVocab_list)
print("  This is Bob. I reply to Alice :",bobSays_str)
```

Removing Stop-Words

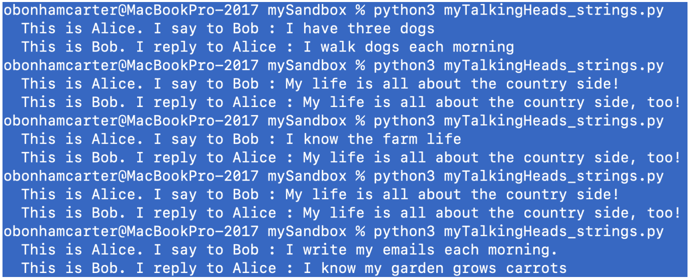
Remove Words from Strings Using Lists

```
stopWords_list =["I", "have","know",
"like", "love", " to ", " a "]

  # we remove stop words
  # as they do not add specificity to the strings

def removeStopWords(in_str): # string input
  for s in stopWords_list:
    in_str = in_str.replace(s,"") #word with empty space
  return in_str.strip()  # remove spaces, return.
  #end of removeStopWords()
```

- Remove stop-words and compare the lists for common words.
- When you find the common words between two lists, you have found a contextual link between them.

Talking Heads Demo

File: sandbox/myTalkingHeads_strings.py



```
obonhamcarter@MacBookPro-2017 mySandbox % python3 myTalkingHeads_strings.py
  This is Alice. I say to Bob : I have three dogs
  This is Bob. I reply to Alice : I walk dogs each morning
obonhamcarter@MacBookPro-2017 mySandbox % python3 myTalkingHeads_strings.py
  This is Alice. I say to Bob : My life is all about the country side!
  This is Bob. I reply to Alice : My life is all about the country side, too!
obonhamcarter@MacBookPro-2017 mySandbox % python3 myTalkingHeads_strings.py
  This is Alice. I say to Bob : I know the farm life
  This is Bob. I reply to Alice : My life is all about the country side, too!
obonhamcarter@MacBookPro-2017 mySandbox % python3 myTalkingHeads_strings.py
  This is Alice. I say to Bob : My life is all about the country side!
  This is Bob. I reply to Alice : My life is all about the country side, too!
obonhamcarter@MacBookPro-2017 mySandbox % python3 myTalkingHeads_strings.py
  This is Alice. I say to Bob : I write my emails each morning.
  This is Bob. I reply to Alice : I know my garden grows carrots
```

- Two lists that interact with each other
- Parsing: Searching for words