CS 177 Python Programming

Recitation 12
Data Collections
In this class

• Lists
• Dictionaries
• Sets
• Tuples
Lists are ordered sequence of elements

With a list you can a single variable to represent an entire sequence, and the individual items in the sequence can be accessed through indexing.

```python
>>> lst = [1, 3, 5, 7, 9]
>>> print(lst[0])
1
>>> print(lst[1])
3
>>> print(lst[2])
5
>>> print(lst[3])
7
>>> print(lst[4])
9
>>> 
```
Lists are heterogenous

• You can mix arbitrary data types in a single list.

\[
\text{lst} = ["X", ["Y1", "Y2"], 3, \text{True}, 7.8]
\]
Recap: List operations

• Concatenation

```python
>>> [1,2,3]+[4,5,6]
[1, 2, 3, 4, 5, 6]
```  

• Repetition

```python
>>> [1,2,3]*2
[1, 2, 3, 1, 2, 3]
```  

• Length

```python
>>> len([1,2,3])
3
```
Recap: List operations—Indexing and Slicing

```python
>>> X = [1,2,3,4,5,6,7,8,9,10]
>>> print(X[0])
1
>>> print(X[-1])
10
>>> print(X[0:5])
[1, 2, 3, 4, 5]
>>> print(X[:3])
[1, 2, 3]
>>> print(X[3:])
[4, 5, 6, 7, 8, 9, 10]
>>> print(X[-1])
10
>>> print(X[:-1])
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>```
Recap: List operations–Indexing and Slicing

```python
>>> X = [1,2,3,4,5,6,7,8,9,10]
>>> print(X[4:8])
[5, 6, 7, 8]
>>> print(X[4:8:1])
[5, 6, 7, 8]
>>> print(X[4:8:2])
[5, 7]
>>> print(X[4:8:-1])
[]
>>> print(X[8:4:-1])
[9, 8, 7, 6]
>>> print(X[4::])
[5, 6, 7, 8, 9, 10]
>>> print(X[:8:1])
[1, 2, 3, 4, 5, 6, 7, 8]
```
Recap: List operations

• Iteration

```python
>>> for x in [1,2,3]:
    print(x)
```

```python
1
2
3
```  

• Membership

```python
>>> X = 2 in [1,2,3]
>>> print(X)
True
```
List operations

• \(<\text{list}>\).append(x): Add element x to the end of the list

```python
>>> X = [1,2,3,4,5]
>>> X
[1, 2, 3, 4, 5]
>>> X.append(6)
>>> X
[1, 2, 3, 4, 5, 6]
``` 

• \(<\text{list}>\).sort(): Sort the list, A comparison function can be an argument

```python
>>> X = [4,6,2,13,5]
>>> X
[4, 6, 2, 13, 5]
>>> X.sort()
>>> X
[2, 4, 5, 6, 13]
```
List operations

• `<list>.reverse()`: Reverse the list

```python
>>> X = [1, 2, 3, 4, 5]
>>> X
[1, 2, 3, 4, 5]
>>> X.reverse()
>>> X
[5, 4, 3, 2, 1]
>>> 
```

• `<list>.index(x)`: Returns the index of the first occurrence of `x`

```python
>>> X = [4, 6, 3, 3, 5, 1]
>>> X.index(3)
2
>>> 
```
List operations

• `<list>.insert(i,x)`: insert element x at index i (does not replace existing)

```python
>>> X = [1,2,3,4,5]
>>> X
[1, 2, 3, 4, 5]
>>> X.insert(2,6)
>>> X
[1, 2, 6, 3, 4, 5]

• `<list>.count(x)`: Returns the number of occurrences of x in list

```python
>>> X = [4,6,3,1,5,3,7,9,3]
>>> X.count(3)
3
>>> ```
List operations

• `<list>.remove(x)`: Deletes the first occurrence of `x`

```python
>>> X = [1, 2, 3, 4, 5]
>>> X
[1, 2, 3, 4, 5]
>>> X.remove(3)
>>> X
[1, 2, 4, 5]
```  

• `<list>.pop(i)`: Delete the *i*th element and return its value

```python
>>> X = [4, 6, 3, 3, 5, 1]
>>> X
[4, 6, 3, 3, 5, 1]
>>> Y = X.pop(3)
>>> X
[4, 6, 3, 5, 1]
>>> Y
3
```
Lists are mutable

The content of objects of mutable types can be changed after they are created.

```python
>>> lst = [1, 2, 3, 4]
>>> lst[3]
4
>>> lst[3] = "Hello"
>>> lst
[1, 2, 3, 'Hello']
>>> lst[2] = 7
>>> lst
[1, 2, 7, 'Hello']
>>> lst[1:3] = ["Slice", "assignment"]
>>> lst
[1, 'Slice', 'assignment', 'Hello']
```
Remember: Strings are immutable

The content of objects of immutable types cannot be changed after they are created.

```python
>>> s = "Hello There Python"
>>> s[0] = "M"
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s[0] = "M"
TypeError: 'str' object does not support item assignment

>>> 
```
Tuples are collection of ordered and immutable sequence of elements.

• Tuples are also a sequence like Strings and Lists, so indexing and slicing works with tuples as well
• A Tuple is similar to a list, the difference being they are immutable

```python
>>> T = (1, 2, 3)
>>> print(T[0])
1
>>> print(T[1])
2
>>> print(T[2])
3
>>> 
```
Tuples – Examples

T1 = (23, 45, 67)
T2 = 1, 5, 9
T3 = tuple([2, 3, 4])

T2[0:2]

T0 = ()
T1 = (23, )

Tuple Definition
Indexing works, and so does slicing
Tuples of length 0 and 1
Tuple Definitions for length 1 must be followed by a comma

No Tuple comprehension, as tuples are immutable and looping and adding values is not allowed
We cannot update or delete elements in a tuple as they are immutable. However, we can create new tuples by taking elements from existing tuples.

```python
tup1 = (3, 4)

#tup1[0] = 5  # This operation is not allowed
lst = list(tup1)  # So, convert the tuple to a list, modify and convert back.
lst[0] = 5

tup1 = tuple(lst)  # => (5, 4)

del lst[0]  # Similarly for deletion

tup1 = tuple(lst)  # => (4,)
```
Tuple Operations

• Length:

```python
>>> T = (1, 2, 3, 4, 5)
>>> len(T)
5
```
Tuple Operations

• Iteration:

```python
>>> for x in (1,2,3):
    print(x)
1
2
3
```

• Concatenation:

```python
>>> (1,2,3) + (4,5,6)
(1, 2, 3, 4, 5, 6)
```
Tuple Operations

• Membership

```python
>>> X = 2 in (2,3,4)
>>> X
True
>>> 
```

• `max(<tuple>): Maximum entry in the Tuple`

```python
>>> X = (4,6,2,3,5,8)
>>> max(X)
8
>>> 
```
Tuple Operations

• `<tuple>.index(x)`: Returns the index of the first occurrence of x

  >>> X = (4,6,3,3,5,1)
  >>> X.index(3)
  2
  >>>

• `<tuple>.count(x)`: Returns the number of occurrences of x in tuple

  >>> X = (2,5,1,5,6,3,7,9)
  >>> X.count(5)
  2
  >>>
Other Sequence operators like sort(), reverse(), remove() etc that modify sequences are not present for tuples.
Dictionaries: A collection of unordered values accessed by key rather than index.

• Instead of indexing by numbers like lists/tuples, it is indexed by keys

• Think of it as a collection of (key,value) pairs with only one value for a key
  Dictionaries can’t be accessed by slicing
  But they can be indexed by keys
Dictionaries – Define and Use

Dictionary Definition
Indexing works, but by Keys
Slicing Doesn’t work

Dictionary Definition from a list of Tuples (can use any sequence of sequences)

Dictionary comprehension

X = {"AA":1, 23:"BB"}

X = dict((("AA",1),(23,"BB")))

X[Key] = Value

X = {i: 2*i for i in range(5)}
Dictionary Operations - Iteration and Membership on Dictionary work on keys

- Length

```python
>>> len({12:21, 5:7})
2
>>> 
```

- Membership

```python
>>> X = 2 in {2:3, 5:7}
>>> X
True
>>> 
```
Dictionary Operations - Iteration and Membership on Dictionary work on keys

- Iteration

```python
>>> for x in {1:4, 2:5, 3:6}:
    print(x)

1
2
3
```

```python
>>> for x,y in {1:4, 2:5, 3:6}.items():
    print(x, ":", y)

1 : 4
2 : 5
3 : 6
```
Dictionary Operations

• `<dict>.clear()`: Empty the Dictionary

```python
>>> X = {1:2, 3:4}
>>> X
{1: 2, 3: 4}
>>> X.clear()
>>> X
{}
>>> 
```

• `<dict>.get(key)>`: Similar to Indexing

```python
>>> X = {1:2, 3:4}
>>> X.get(3)
4
>>> X.get(1)
2
>>> 
```
Dictionary Operations

• `<dict>.items()`: Returns Lists of dict’s tuples (key, value) pairs

```python
>>> X = {1:2, 3:4}
>>> X
{1: 2, 3: 4}
>>> X.items()
dict_items([(1, 2), (3, 4)])
```

• `<dict>.keys()`: Returns a list of Dictionary’s keys

```python
>>> X = {1:2, 3:4}
>>> X.keys()
dict_keys([1, 3])
```
Dictionary Operations

• `<dict>.values()`: Returns a list of Dictionary’s values

```python
>>> X = {1:2, 3:4}
>>> X.values()
dict_values([2, 4])
```

• `<dict1>.update(<dict2>)`: Adds dict2 entries to dict1

```python
>>> X = {1:2, 3:4}
>>> X
{1: 2, 3: 4}
>>> X.update({5:6})
>>> X
{1: 2, 3: 4, 5: 6}
```
Dictionary Operations

• `dict.fromkeys(<listKeys>):` Creates a new dictionary with the keys

```python
>>> dict.fromkeys([8, 9])
{8: None, 9: None}
>>> ```
Sets - A collection of unordered and non duplicated values

- Follow the abstract mathematical concept of a set
  - A collection of unique values
- Common use cases are membership testing, removing duplicates, set operations such as intersection and union etc
Sets - Define and Use

- Set Definition:
  - Empty Set must be defined with the constructor, `{}` defines a dictionary
  - Unordered, and slicing and indexing both do not work

X = {2,3,4}
X = set({})
X = {'Key'}

X = {2,3,4,3}
print(X)
{2,3,4}

Unique Values
Set Operations

• Length:

```python
>>> len({2,3,4})
3
```  

• Membership:

```python
>>> X = 2 in {2,3,4}
>>> X
True
```
Set Operations

• Iteration:

```python
>>> for x in {2,3,4}:
    print(x)
```

2  
3  
4  

• Set Containment

```python
>>> {2}.issubset({2,3,4})
True
>>> {2,3,4}.issuperset({2,3,4})
True
```
Set Operations

• `<set>.clear()`: Empty the Set

```python
>>> X = {2,3,4}
>>> X
{2, 3, 4}
>>> X.clear()
>>> X
set()

• `<set>.add(x)`: Adds x to the Set

```python
>>> X = {1,3}
>>> X
{1, 3}
>>> X.add(2)
>>> X
{1, 2, 3}
```
Set Operations

• `<set>.remove(x)`: Removes x if present, raises KeyError Otherwise

```python
>>> X = {1,3}
>>> X
{1, 3}
>>> X.remove(3)
>>> X
{1}
>>> X.remove(2)
Traceback (most recent call last):
  File "<pyshell#21>" , line 1, in <module>
    X.remove(2)
KeyError: 2
>>> X = {1,3}
>>> X
{1, 3}
>>> X.discard(3)
>>> X
{1, 3}
>>> X.discard(2)
>>> X
{1, 3}
>>> X
```
Set Operations

• `<set1>.update(<set2>): Adds set2 entries to set1`

```python
>>> X = {1,3}
>>> X
{1, 3}
>>> X.update({5})
>>> X
{1, 3, 5}
```
Examples

• Given a String line, find all the vowels in the string
  • Let us implement this with Sets
Examples

• Given a String line, find all the vowels in the string
  • Let us implement this with Sets
• Create a set of all the vowels in line.

```python
vowels = set()
for x in line:
    if x=='a' or x=='e' or x=='i' or x=='o' or x=='u':
        vowels.add(x)

print(vowels)
```
Examples

• Given a String line, find all the vowels in the string
  • Let us implement this with Sets
• Create a set of all the vowels in line.

```python
vowels = set()
for x in line:
    if x=='a' or x=='e' or x=='i' or x=='o' or x=='u':
        vowels.add(x)
print(vowels)
```

Output:

```python
line = "abcdef"
vowels: {"a"","e"}
```

```python
line: "aabcd"
vowels: {"a"}
```
Examples

• Convert an English sentence to French
  • You are given an English to French dictionary as a python dict type
  • Don’t worry about Grammar
Examples

• Convert an English sentence to French
  • You are given an English to French dictionary as a python dict type
  • Don’t worry about Grammar

frenchDict = {
  "I":"je", "am":"suis", "love":"aimer", "very":"très",
  "much":"beaucoup"
}

defineDict[“happy”]=“heureux”

english = "I love python very much . I am very happy"

for englishWord in english.split():
    if (englishWord in frenchDict):
        print(frenchDict[englishWord], end = " ")
    else:
        print(englishWord, end = " ")
Examples

• Convert an English sentence to French
  • You are given an English to French dictionary as a python dict type
  • Don’t worry about Grammar

```
frenchDict = {
  "I": "je",
  "am": "suis",
  "love": "aimer",
  "very": "très",
  "much": "beaucoup"
}

frenchDict["happy"] = "heureux"

english = "I love python very much. I am very happy"

for englishWord in english.split():
  if (englishWord in frenchDict):
    print(frenchDict[englishWord], end=" ")
  else:
    print(englishWord, end=" ")
```

Output:

```
je aimer python très beaucoup. je suis très heureux
```

• Convert an English sentence to French
  • You are given an English to French dictionary as a python dict type
  • Don’t worry about Grammar

```python
frenchDict = {
  "I": "je",
  "am": "suis",
  "love": "aimer",
  "very": "très",
  "much": "beaucoup"
}

frenchDict["happy"] = "heureux"

english = "I love python very much. I am very happy"

for englishWord in english.split():
  if (englishWord in frenchDict):
    print(frenchDict[englishWord], end=" ")
  else:
    print(englishWord, end=" ")
```

Output:

```
je aimer python très beaucoup. je suis très heureux
```
Questions?