Data Collections

CS 177 – Recitation 12

Announcements

- Midterm 2 on Nov 17, Tuesday, 6:30-7:30 pm, WTHR 200
  - Material: Chapter 1-11, Recitations, labs and lectures program

- Project 4 will be up soon

Objectives

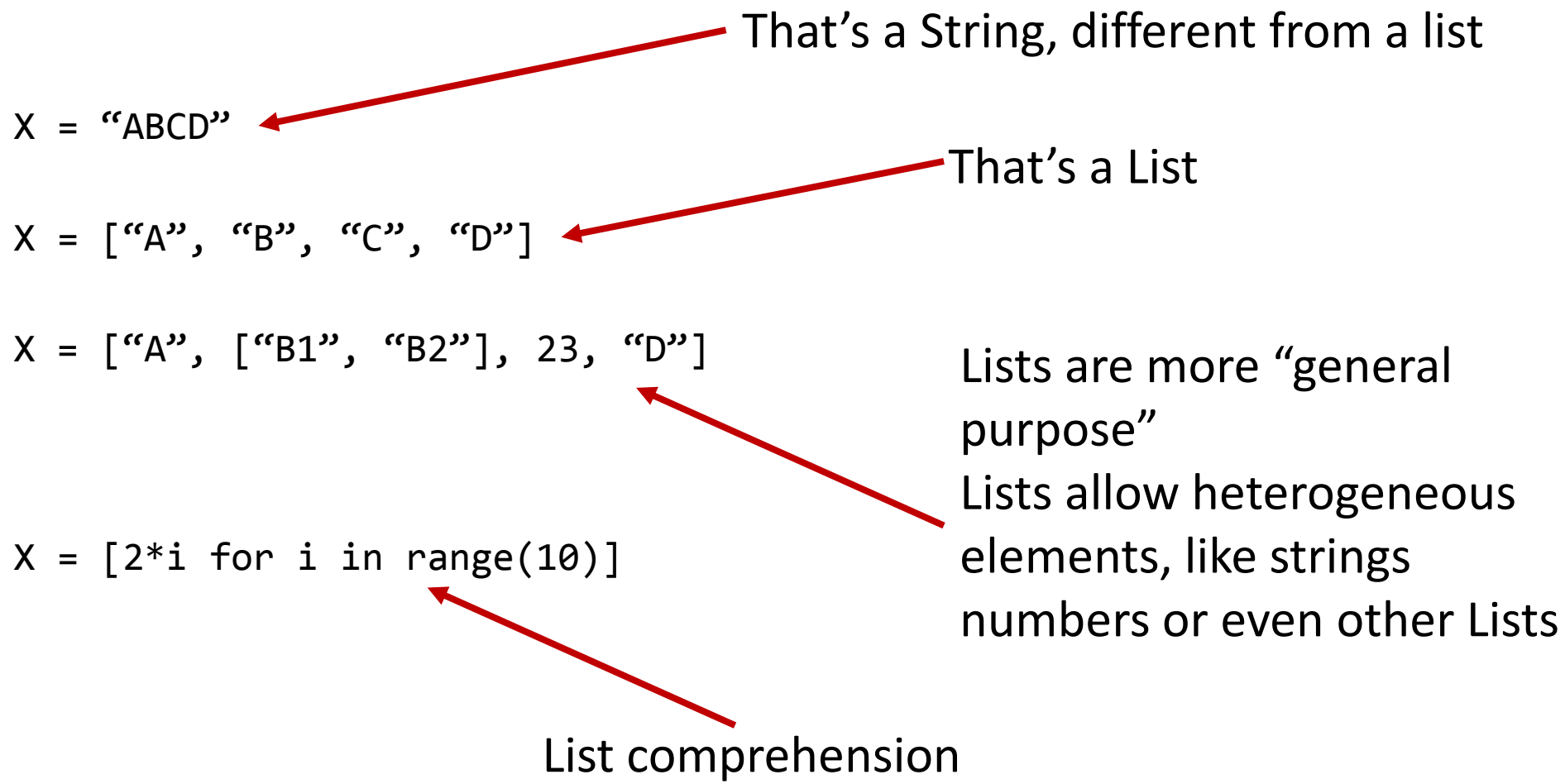
- Review the use of lists to represent a collection of data

- Other Data Collections in Python
  - Dictionaries
  - Sets
  - Tuples

Sequences

- When we have multiple **elements** stored in order in memory, it is called a **List**
- When we have multiple **characters** stored in order in memory, it is called a **String**
- Both of these structures are sequence structured, individual elements can be accessed by indexing
- Ranges, List, Tuples and String are indexed from 0

Sequence Example - Lists – Define



Sequences – Indexing and Slicing

- [] notation can be used to index into lists, ranges and strings.

- A[i] is the name for ***i****th* element for a sequence
  - The index starts from 0 (So, *(i+1)th element if indexing from 1*) to (length-1)
  - Indexing can also be done from -1 to -length

- Slicing is done with : inside [] notation
  - Slicing returns a subsequence of the original sequence back
  - Just like range function, slicing can take three arguments (start, stop, step)

Sequences – Indexing and Slicing

```
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]
```

Sequences – Indexing and Slicing

```
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::1])                   [5,6,7,8,9,10]

print(X[:8:1])                   [1,2,3,4,5,6,7,8]
```

Sequence/List Operations

- Concatenation

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

- Repetition

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

- Indexing

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

- Length

```
len([1,2,3])             3
```

Sequence/List Operations

- Slicing

```
[1,2,3][1:2]
```

```
[2]
```

- Iteration

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

```
1
2
3
```

- Membership

```
X = 2 in [2,3]
print(X)
```

```
True
```

Sequence/List Operations

- <list>.append(x): Add element x to the end of the list

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

- <list>.sort(): Sort the list, A comparison function can be an argument

```
A=[4,6,2,3,5,1]                  [1,2,3,4,5,6]
A.sort()
```

- <list>.reverse(): Reverse the list

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

- <list>.index(x): Returns the index of the first occurance of x

```
A=[4,6,3,3,5,1]                  2
A.index(3)
```

Sequence/List Operations

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

```
A=[1,2,3,4,5]                    [1,2,6,3,4,5]
A.insert(2,6)
```

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

```
A=[4,6,3,3,5,1]                  2
A.count(3)
```

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

```
A=[1,2,3,4,5]                    [1,2,4,5]
A.remove(3)
```

- <list>.pop(i): Delete the ith element and return its' value

```
A=[4,6,3,3,5,1]                  [4,6,3,5,1]
A.pop(2)
```

9. Given the following assignment:

```
fun = [['C','i','s',["7","1"]],['S'],['cs17','
  177','f'],[('c','s',1,7,7)]]
```

Which of the following code snippets will display the string "cs177"?

I
```
for i in range(len(fun[3][0])):
    print(fun[3][0][i], end='')
```

II `fun[3]`

III `fun[2][0]+fun [0][3][0]`

IV `fun [2][1]`

A. I

B. I and II

C. I and III

D. I, II and III

E. The statement assigning a value to variable fun will result in an error message

9. Given the following assignment:

```
fun = [['C','i','s',["7","1"]],['S'],['cs17','
  177','f'],[('c','s',1,7,7)]]
```

Which of the following code snippets will display the string "cs177"?

I
```
for i in range(len(fun[3][0])):
    print(fun[3][0][i], end='')
```

II `fun[3]`

III `fun[2][0]+fun[0][3][0]`

IV `fun[2][1]`

A. I

B. I and II

C. I and III

D. I, II and III

E. The statement assigning a value to variable fun will result in an error message

Answer: C

I. cs177

II. [('c', 's', 1, 7, 7)]

III. cs177

IV. 177

11. What is the output of the following python program?

```
def theAnswer():
    X = [4*[10] for j in range(7)]
    Y = [3*['Arthur'] for j in range(len(X)-1)
        ]
    Z = [3*['Trillian'] for j in range(len(X)*
       len(Y))]
    print(len(Z))


theAnswer()
```

A. 36

B. 42

C. 504

D. 6804

E. None of the above

11. What is the output of the following python program?

```
def theAnswer():
    X = [4*[10] for j in range(7)]
    Y = [3*['Arthur'] for j in range(len(X)-1)
        ]
    Z = [3*['Trillian'] for j in range(len(X)*
       len(Y))]
    print(len(Z))

theAnswer()
```

A. 36

B. 42

C. 504

D. 6804

E. None of the above

Answer: B

len(x) = 7, len(y) = 6

Dictionaries, Sets and Tuples

- A collection of **unordered** values accessed by key rather than index is called a **Dictionary**
- A collection of **unordered** and non duplicated values is called a **Set**
- A collection of ordered and immutable sequence of elements is called a **tuple**
- Note: As Dictionary/Set are unordered, there is no accessing of elements by index or slicing, instead there are other functions to check membership (dictionary elements can be accessed by key)

Differences between Data Collections

Data Collection	Description
List	Sequentially ordered, mutable, can have duplicates, heterogeneous elements
String	Sequentially ordered, immutable, can have duplicates, character elements
Dictionary	Unordered, mutable, no duplicates, heterogeneous elements
Set	Unordered, mutable, no duplicates, heterogeneous elements
Tuple	Sequentially ordered, immutable, can have duplicates, heterogeneous elements

Tuples

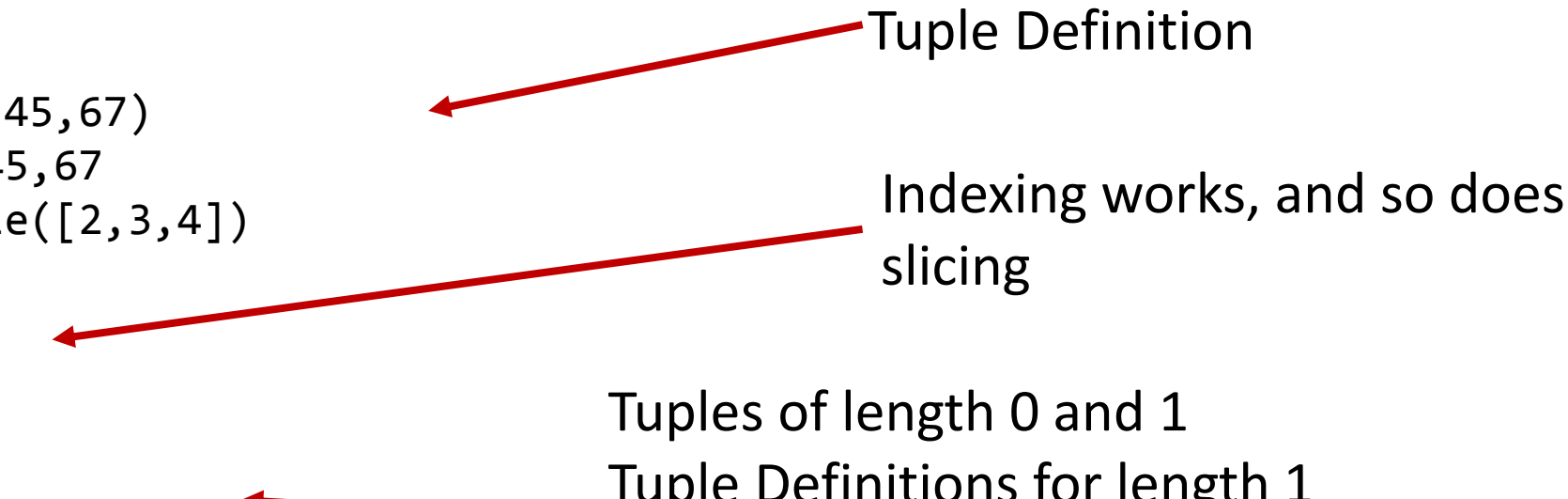
- A collection of ordered and immutable sequence of elements is called a **tuple**
- A Tuple is similar to a list, the difference being they are immutable
- Tuples normally used for heterogeneous items (but not required)
- Tuples are also a sequence like Strings and Lists, so indexing and slicing works with tuples as well
- Tuples are specifically used in value packing and unpacking, which is basically the mechanism via which functions return multiple return values

Tuples – Define and Use

```
X = (23,45,67)
X = 23,45,67
X = tuple([2,3,4])

X[0:2]


X = ()
X = (23,)
```



No Tuple comprehension, as tuples are immutable and looping and adding values is not allowed

Tuples – Update and Deletion

As Tuples are immutable, we cannot update or delete elements in a tuple.
However, we can create new tuples by taking elements from existing tuples

```
Tup1 = (3,4)

# Tup1[0]=5

Lst = list(tup1)
Lst[0] = 5
Tup1 = tuple(Lst)

del Lst[0]
Tup1 = tuple(Lst)
```

This operation is not allowed
So, convert the tuple to a list, modify and convert back.

(5,4)

Similarly for deletion
(4,)

Tuple Operations

- Length:

```
A=(1,2,3,4,5)
len(A)
```

```
5
```

- Repetition:

```
A=(4,)*2
```

```
(4,4)
```

- Iteration:

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

```
1
2
3
```

- Concatenation:

```
(1,2,3)+(4,5,6)
```

```
(1,2,3,4,5,6)
```

Tuple Operations

Other Sequence operators like sort(), reverse(), remove() etc that modify sequences are not present for tuples

- Membership

```
X = 2 in (2,3)                    True
print(X)
```

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

```
A=(4,6,2,3,5,1)                   6
max(A)
```

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

```
A=(4,6,3,3,5,1)                   2
A.index(3)
```

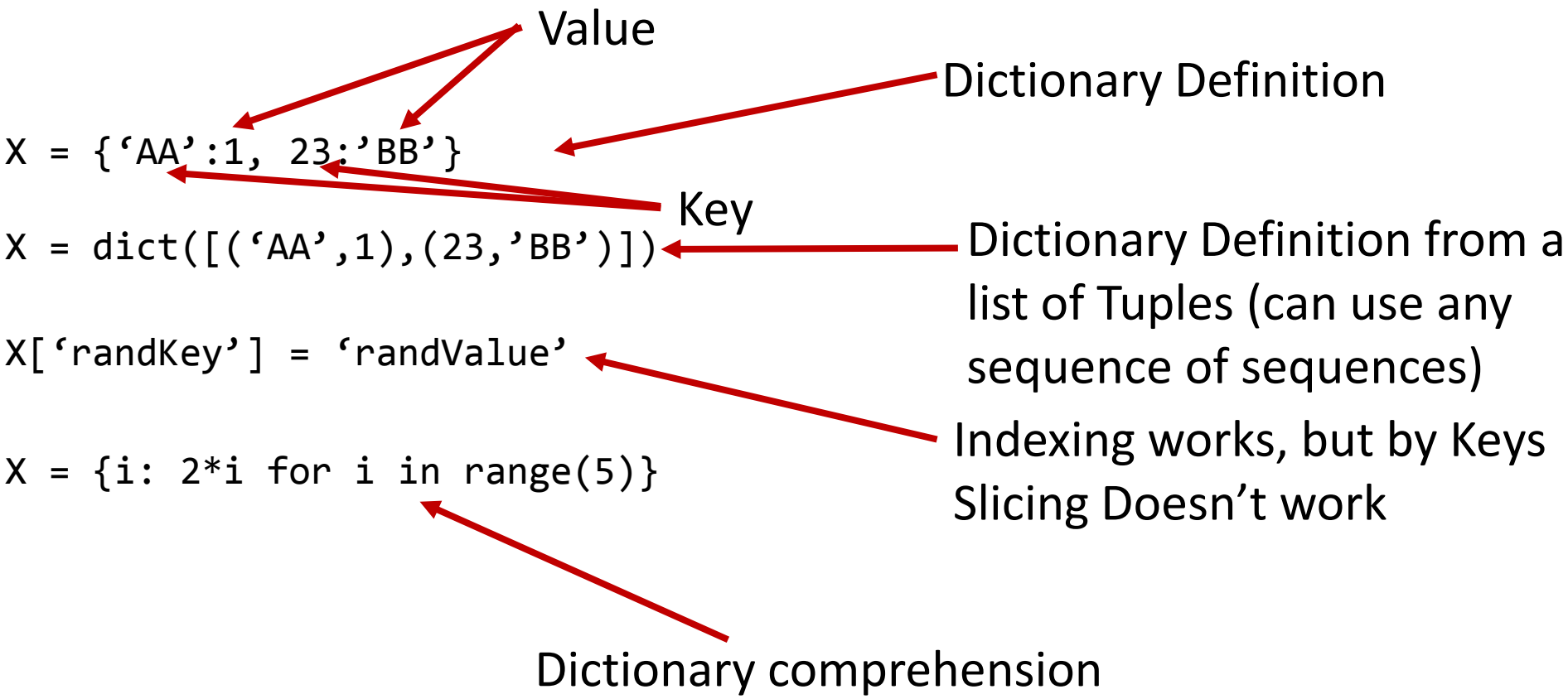
- <tuple>.count(x): Returns the number of occurrences of x in tuple

```
A=(4,6,3,3,5,1)                   2
A.count(3)
```

Dictionaries

- A collection of **unordered** values accessed by key rather than index is called a **Dictionary**
- Also known as associative arrays
- Instead of indexing by numbers like sequences, 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 indexed by keys

Dictionaries – Define and Use



Dictionary Operations

- Length

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

```
2
```

- Membership

```
X = 2 in {2:4,3:5}
print(X)
```

```
True
```

- Iteration

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

```
1
2
3
```

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

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

Iteration and Membership on Dictionary work on keys

Dictionary Operations

Sequence methods that employ order don't work on dictionary and on sets, like sort(), reverse() etc,

- <dict>.clear(): Empty the Dictionary

```
A={1:2,3:4}                    {}
A.clear()
```

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

```
A={1:2,3:4}                    4
A.get(3)
```

- <dict>.items(): Returns Lists of dict's tuples (key, value) pairs

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

Dictionary Operations

- <dict>.keys(): Returns a list of Dictionary's keys

```
A={1:2,3:4}                                    [1,3]
A.keys()
```

- <dict>.values(): Returns a list of Dictionary's values

```
A={1:2,3:4}                                    [2,4]
A.values()
```

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

```
A={1:2,3:4}                                    {1:2,3:4,5:6}
A.update({5:6})
```

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

```
dict.fromkeys([8,9])                           {8:None,9:None}
```

Read following instructions and answer Question 39 and 40:

Given two dictionaries, d1 and d2, create a new dictionary with the following property: for each entry (a, b) in d1, if there is an entry (b, c) in d2, then the entry (a, c) should be added to the new dictionary. For example, if d1 is 2:3, 8:19, 6:4, 5:12 and d2 is 2:5, 4:3, 3:9, then the new dictionary should be 2:9, 6:3 Associate the new dictionary with the variable d3

```
def mydictionaries(d1, d2):
    (*****1*****)
    for i in d1:
        if d1[i] in d2.keys():
            (*****2*****)
```

39. For (*****1*****) you should have:

A. d3 = {}

B. d1 = {}

C. d1 = {}
   d2 = {}

D. d1 = {}
   d2 = {}
   d3 = {}

E. d3 = {}
   i = 0

Answer: A

40. For (*****2*****) you should have:

I d3 [i] = d2[d1[i]]

II d3 [i] = d1[i]

III d3.update(i: d2[d1[i]])

IV d3.update(d2[i]: d2[d1[i]])

A. I

B. I or II

C. I or II or III

D. I or III or IV

E. III or IV

Answer: A

Sets

- A collection of **unordered** and non duplicated values is called a **Set**
- 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

```
X = {2,3,4}

X = set({})

###X['randKey']
```

Set Definition
Empty Set must defined with the constructor, {} defines a dictionary

Unordered, and slicing and indexing both do not work

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

Unique Values

Set Operations

- Length:

```
len({12, 5})                    2
```

- Membership:

```
X = 2 in {2,3}                  True
print(X)
```

- Iteration:

```
for x in {1,2,3}:               1
    print(x)                    2
                                3
```

- Set Containment

```
{1}.issubset({1,2,3})           True
{1,2,3}.issuperset({1})         True
```

Set Operations

Sequence methods that employ order don't work on sets, like sort(), reverse() etc,

- <set>.clear(): Empty the Set

```
A={1,2}                    {}
A.clear()
```

- <set>.add(x): Adds x to the Set

```
A={1,3}                    {1,2,3}
A.add(2)
```

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

```
A={1,3}                    {1}
A.remove(3)
```

- <set>.discard(x): Removes x if present

```
A={1,3}                    {1}
A.discard(3)
```

Set Operations

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

```
A={1,3}                                    {1,3,5}
A.update({5})
```

- Set Theory Operations
  - <set1>.intersection(<set2>): New Set with elements common to both sets
  - <set1>.union(<set2>): New Set with elements from both sets
  - <set1>.difference(<set2>): New Set with elements in set1 but not in set2
  - <set1>.symmetric_difference(<set2>): New Set with elements in either set1 or set2, but not in both

38. What is the output of the following Python program?

```
S = set()
T = {1, 0, 2, 3, 2, 3}
U = {9, 5, 1, 4, 3}
S.update(T)
S.update(U)
print(S)
```

A. 1, 0, 2, 3, 2, 9, 5, 1, 4, 3

B. 0, 1, 2, 3, 4, 5, 9

C. 9, 5, 1, 4, 3

D. 9, 5, 1, 4, 3, 3

E. None of the above is correct

38. What is the output of the following Python program?

```
S = set()
T = {1, 0, 2, 3, 2, 3}
U = {9, 5, 1, 4, 3}
S.update(T)
S.update(U)
print(S)
```

Answer: B

A. 1, 0, 2, 3, 2, 9, 5, 1, 4, 3

B. 0, 1, 2, 3, 4, 5, 9

C. 9, 5, 1, 4, 3

D. 9, 5, 1, 4, 3, 3

E. None of the above is correct

QUESTIONS?