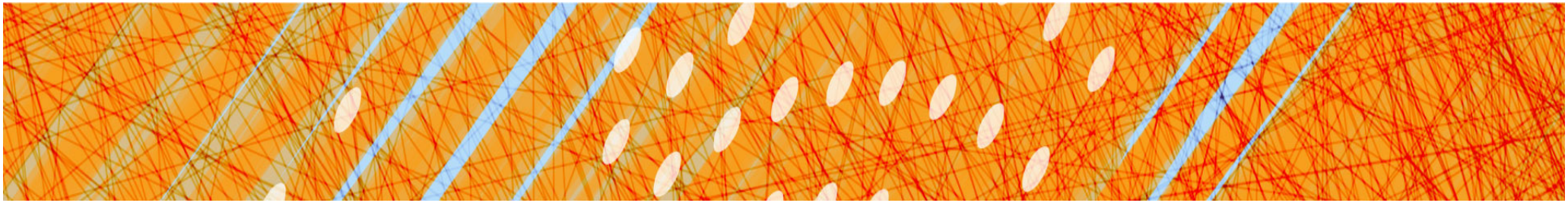


Lecture 10:
Lists and Dictionaries

(Sections 10.0-10.2, 10.4-10.6, 11.1-11.5)

CS 1110

Introduction to Computing Using Python

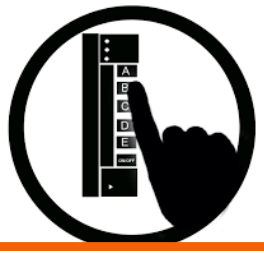
[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Announcements

- No OH over February Break
- See recent Canvas Announcement for all other due dates

How far are you with A2?

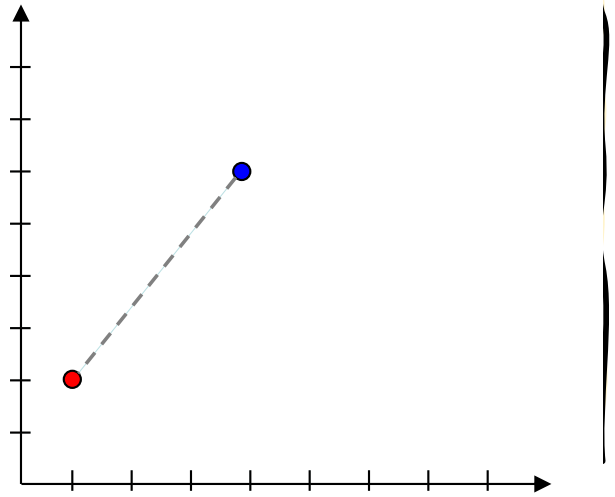
A. Done or almost done.
B. I've started the diagrams but have a lot left.
C. I've run / stepped through the code, but haven't started the diagrams.
D. I've read the assignment but haven't started the diagrams.
E. Yeah, I haven't even looked at it yet.



Recall: 2 Points Make a Line

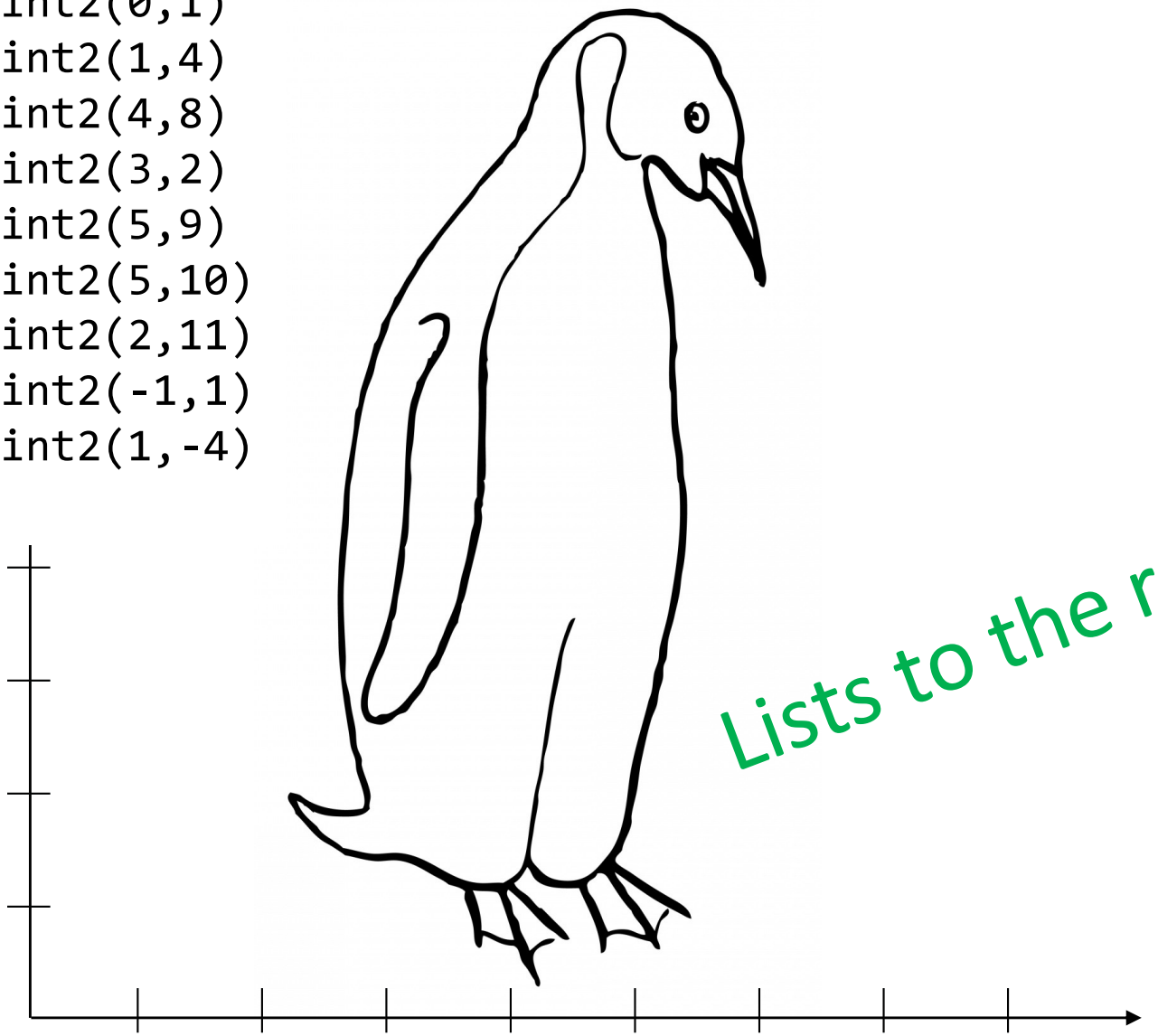
```
def get_coord(name):
    x = input(name+": ")
    return int(x)

def configure(pt, end):
    print("Where does the line " + end + "?
    pt.x = get_coord("x")
    pt.y = get_coord("y")
    print("The line " +end+ "s at ("+str(pt.x)
)

start = shape.Point2(0,0)
stop = shape.Point2(0,0)
configure(start, "start")
configure(stop, "stop")
```

```
Where does the line start?
x: 1
y: 2
The line starts at (1,2).
Where does the line stop?
x: 4
y: 6
The line stops at (4,6).
```



What if we need many points?

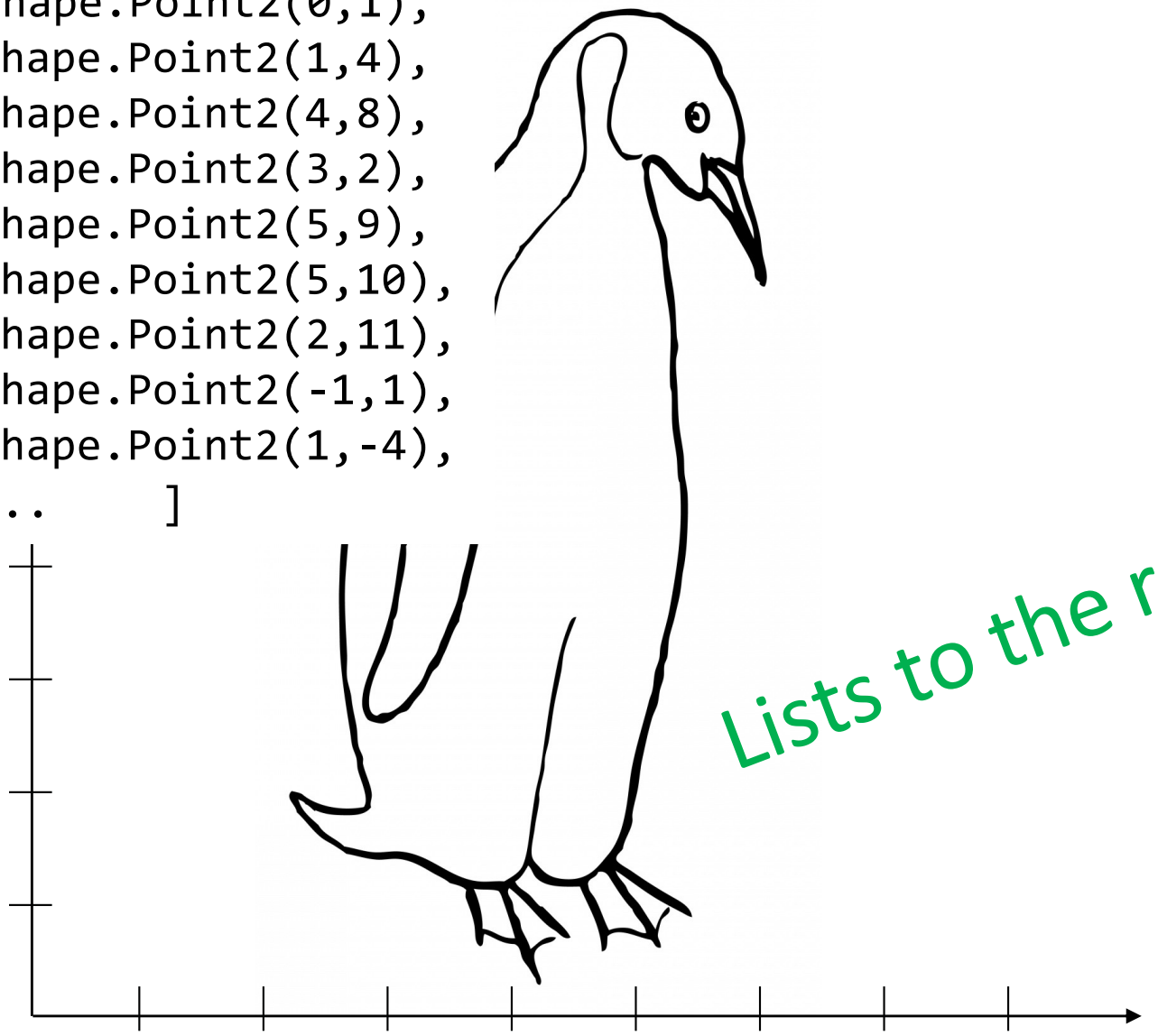
```
a = shape.Point2(0,1)
b = shape.Point2(1,4)
c = shape.Point2(4,8)
d = shape.Point2(3,2)
e = shape.Point2(5,9)
f = shape.Point2(5,10)
g = shape.Point2(2,11)
h = shape.Point2(-1,1)
i = shape.Point2(1,-4)
...
```



Lists to the rescue!

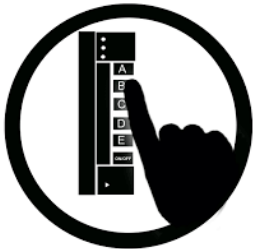
What if we need many points?

```
penguin = [shape.Point2(0,1),
           shape.Point2(1,4),
           shape.Point2(4,8),
           shape.Point2(3,2),
           shape.Point2(5,9),
           shape.Point2(5,10),
           shape.Point2(2,11),
           shape.Point2(-1,1),
           shape.Point2(1,-4),
           ...     ]
```



Lists to the rescue!

Basic List Operations (1)



Create a list of ints:

```
x = [5, 6, 5, 9, 15, 23]
```

type of **x** is **list**

0	1	2	3	4	5
5	6	5	9	15	23

Access a list element using its index:

```
y = x[3]
```

type of **y** is **int**

What is the value of y?

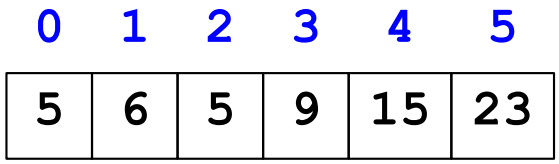
A: 5
B: 6
C: 9
D: 3
E: I don't know

Basic List Operations (2)

Create a list of ints:

```
x = [5, 6, 5, 9, 15, 23]
```

type of **x** is **list**



Access a list element using its index:

```
y = x[3]
```

type of **y** is **int**

What is the value of y? ***9***

Create a list of strings:

```
s = ['hi', 'world']
t = s[0]
```

type of **s** is **list**

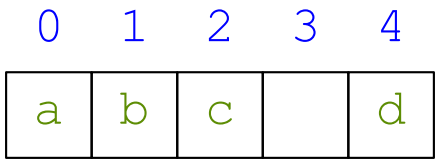
type of **t** is **str**

Does any of this look familiar?

Sequences: Umbrella term for Strings and Lists

String

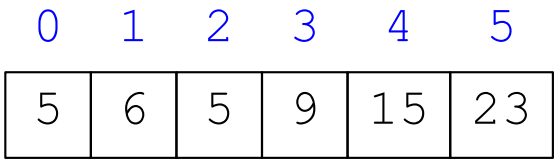
- s = 'abc d'



- Put characters in quotes
  - Use \' for quote character
- Access characters with []
  - s[0] is 'a'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d‘
- len(s) → 5, length of string

List

- x = [5, 6, 5, 9, 15, 23]



- Put values inside [ ]
  - Separate by commas
- Access values with []
  - x[0] is 5
  - x[6] causes an error
  - x[0:2] is [5, 6] (excludes 2nd 5)
  - x[3:] is [9, 15, 23]
- len(x) → 6, length of list

Things that Work for All Sequences

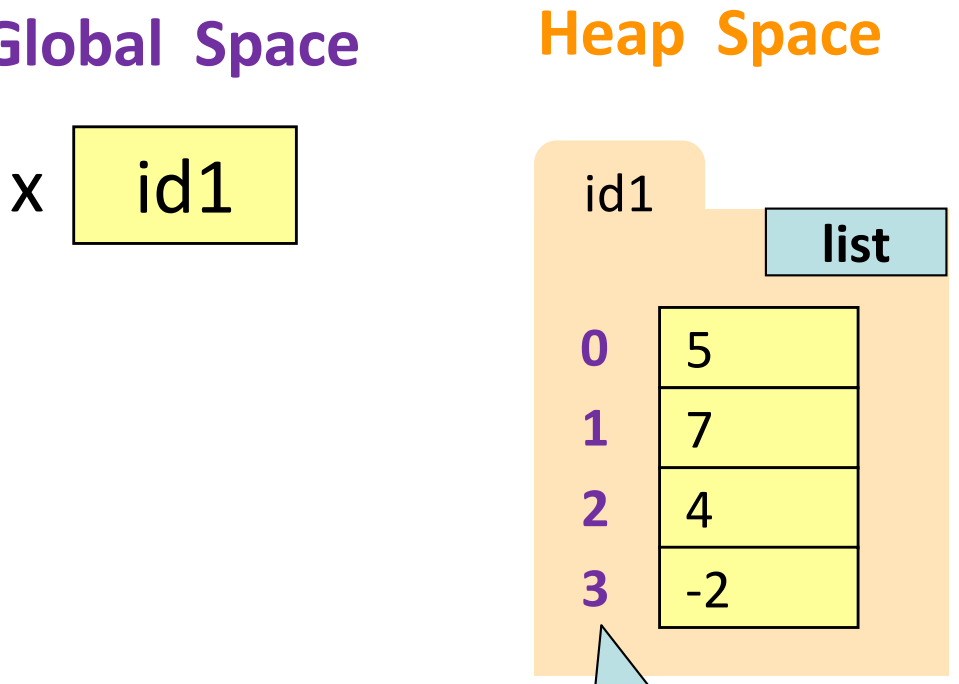
s = 'slithy'		x = [5, 6, 9, 6, 15, 5]
s.index('s') → 0	methods	x.index(5) → 0
s.count('t') → 1		x.count(6) → 2
len(s) → 6	built-in fns	len(x) → 6
s[4] → "h"	slicing	x[4] → 15
s[1:3] → "li"		x[1:3] → [6, 9]
s[3:] → "thy"		x[3:] → [6, 15, 5]
s[–2] → "h"		x[–2] → 15
s + ' toves' → "slithy toves"	operators	x + [1, 2] → [5, 6, 9, 6, 15, 5, 1, 2]
s * 2 → "slithyslithy"		x * 2 → [5, 6, 9, 6, 15, 5, 5, 6, 9, 6, 15, 5]
't' in s → True		15 in x → True

See Prelim 1 Reference Sheet on Canvas

Lists in Memory

Wrong:

Correct:



x = [5, 7, 4,-2]

Lists: objects with "string-like" syntax

Objects

- Attributes are named
  - Example: p.x

Global Space

p | id3

Heap Space

id3 — Point3

x	1
y	2
z	3

List*

- Attributes are indexed
  - Example: x[2]

Global Space

x | id2

Heap Space

id2 — list

0	5
1	7
2	4
3	-2

**also an object, but special*

List Methods Can Alter the List

x = [5, 6, 5, 9]

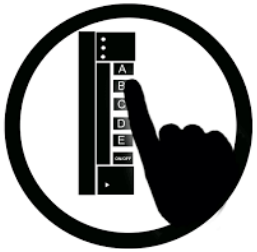
y = [15, 16, 15, 19]

See Python API for more

- <list>.append(<value>)
  - Adds a new value to the end of list
  - x.append(-1) changes the list to [5, 6, 5, 9, -1]
- <list>.insert(<index>,<value>)
  - Puts value into list at index; shifts rest of list right
  - **y**.insert(2,-1) changes the list to [15, 16, -1, 15, 19]
- <list>.sort()

What do you think this does?

Q: Insert into list



- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> x[3] = -1
>>> x.insert(1, 2)
```

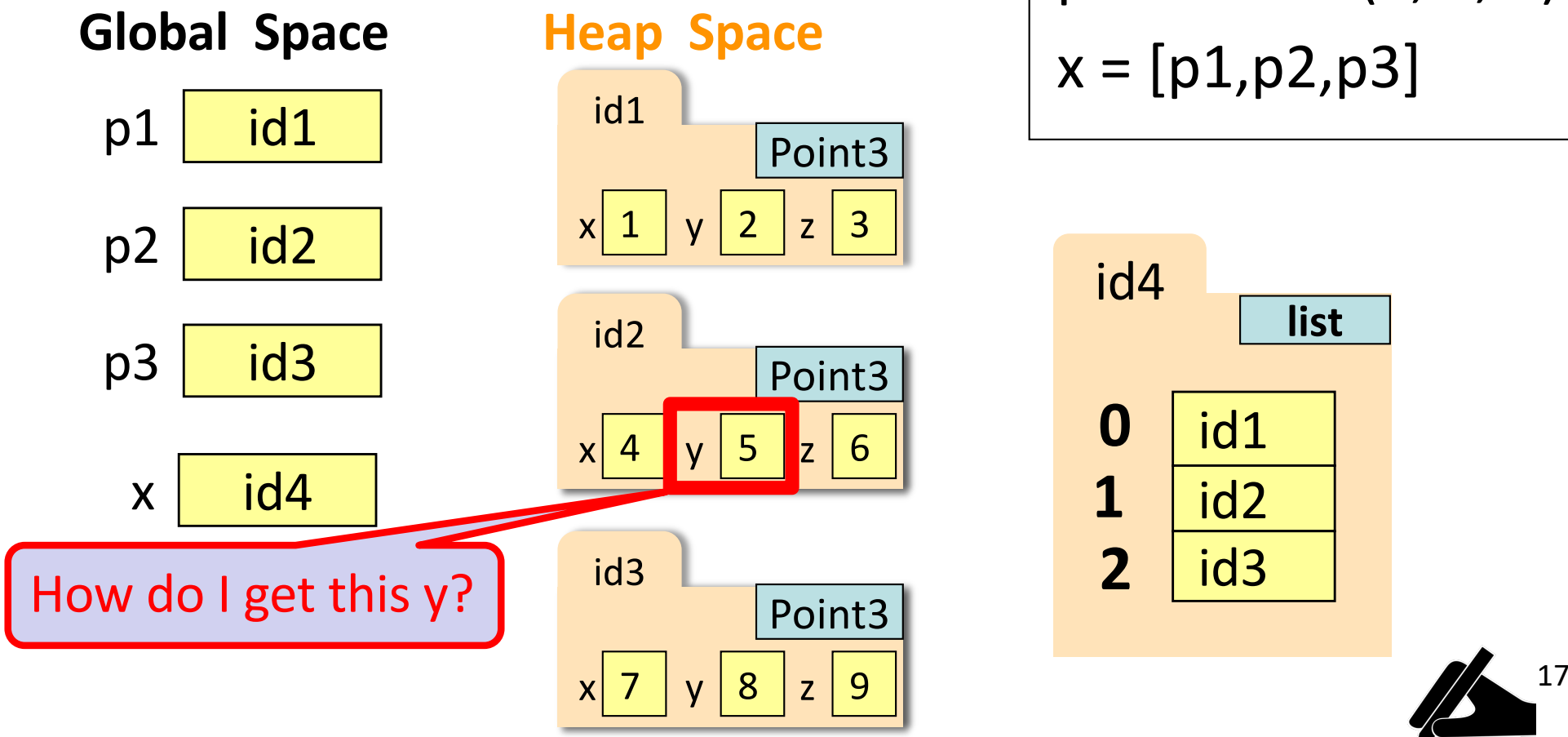
- What is x[4]?

A: 10
B: 9
C: -1
D: ERROR
E: I don't know

Lists of Objects

- List elements are variables
  - Can store base types and ids
  - Cannot store folders

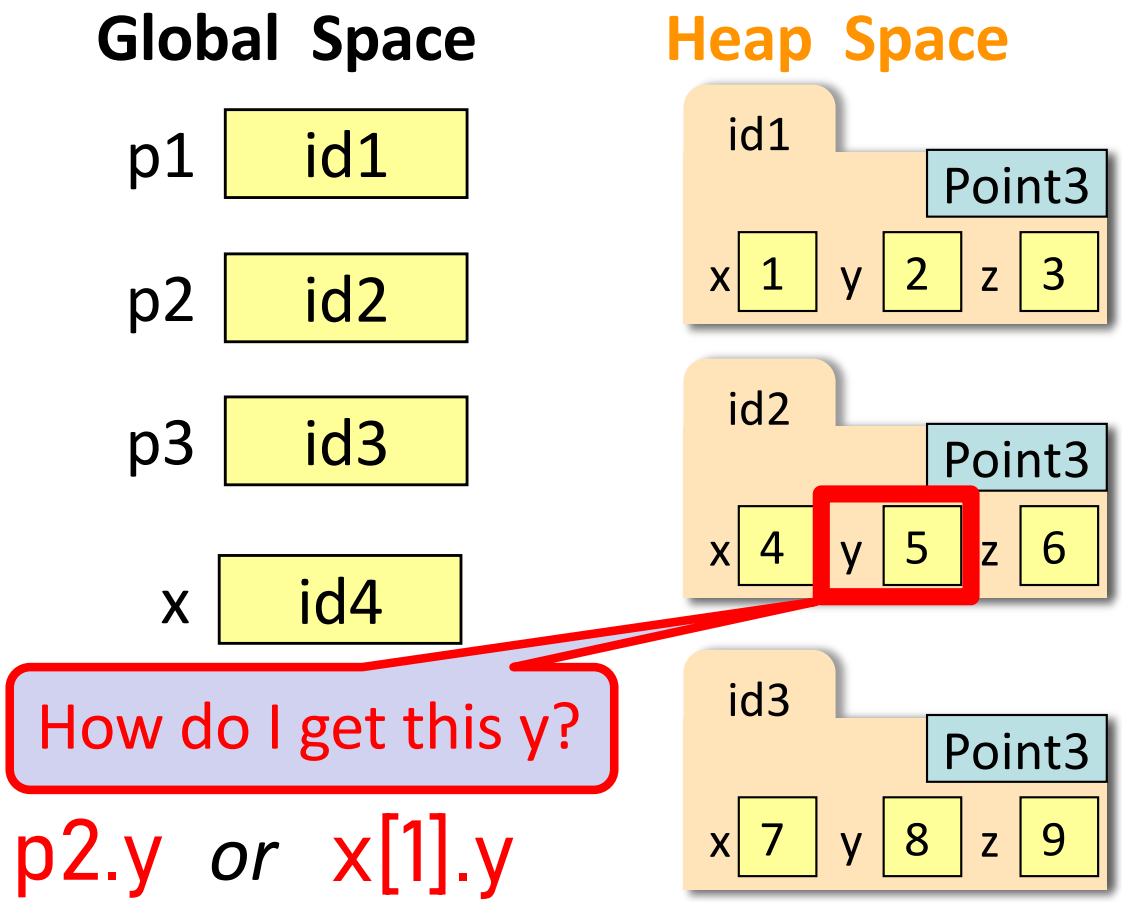
```
p1 = Point3(1, 2, 3)
p2 = Point3(4, 5, 6)
p3 = Point3(7, 8, 9)
x = [p1,p2,p3]
```

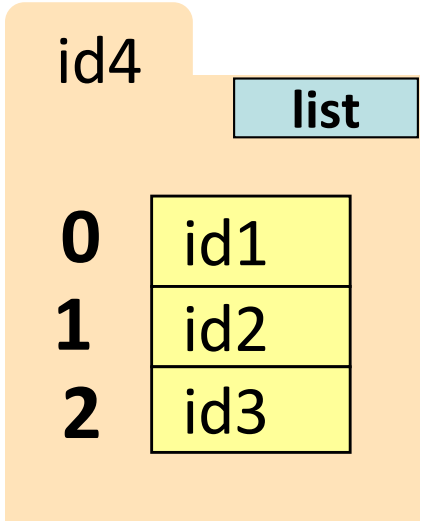


Lists of Objects

- List elements are variables
  - Can store base types and ids
  - Cannot store folders

```
p1 = Point3(1, 2, 3)
p2 = Point3(4, 5, 6)
p3 = Point3(7, 8, 9)
x = [p1,p2,p3]
```

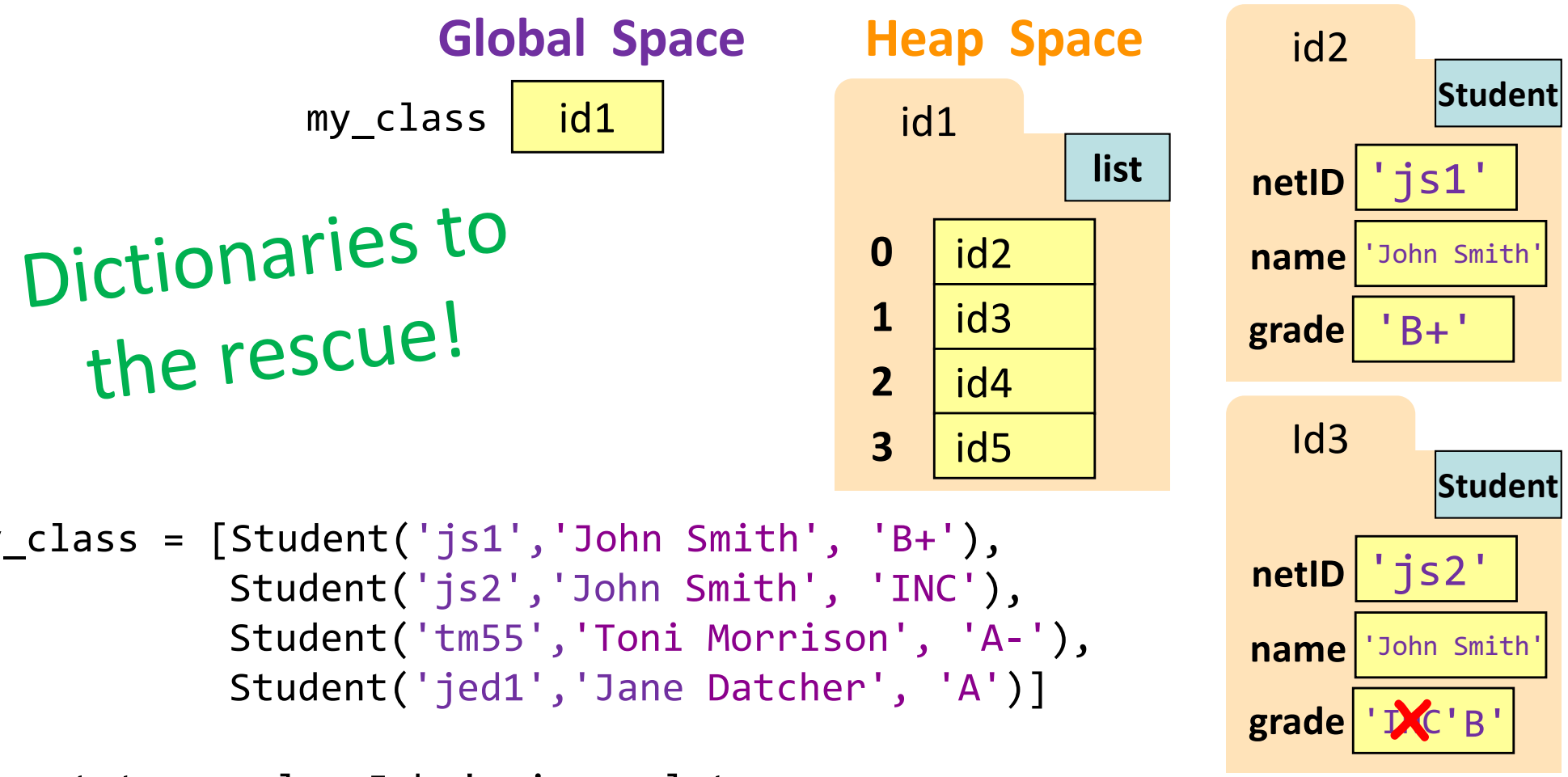




Prelim 1 Material Stops here.
(There are list practice questions at the end of the lecture that are also in scope for the Prelim.)

But the rest of this lecture is necessary for A3, so listen up!

What if indices seem cumbersome?



Dictionaries to the rescue!

```
my_class = [Student('js1','John Smith', 'B+'),
            Student('js2','John Smith', 'INC'),
            Student('tm55','Toni Morrison', 'A-'),
            Student('jed1','Jane Datcher', 'A')]

# want to resolve John's incomplete
my_class[1].grade = 'B'
```

So we just remember that this student is index 1?
Or search through the list looking for the right student?

Dictionaries (Type `dict`)

Description

- List of key-value pairs
  - Keys are unique
  - Values need not be
- Example: net-ids
  - net-ids are **unique** (a key)
  - names need not be (values)
  - js1 is John Smith (class '13)
  - js2 is John Smith (class '16)

Python Syntax

- Create with format:

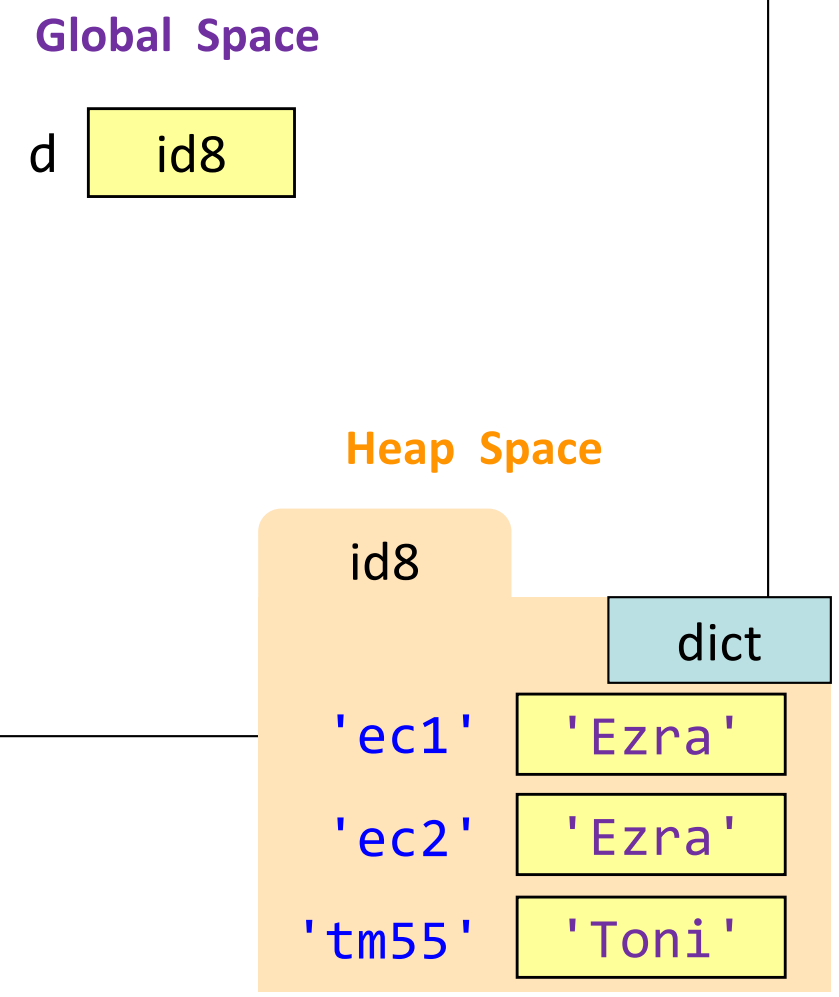
```
{key1:value1,
 key2:value2, …}
```

- Keys must be **immutable**
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- Example:

```
d = {'js1':'John Smith',
     'js2':'John Smith',
   'tm55':'Toni Morrison'}
```

Using Dictionaries (Type `dict`)

```
>>> d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
>>> d['ec1']
'Ezra'
>>> d[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> d[:1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'slice'
>>>
```



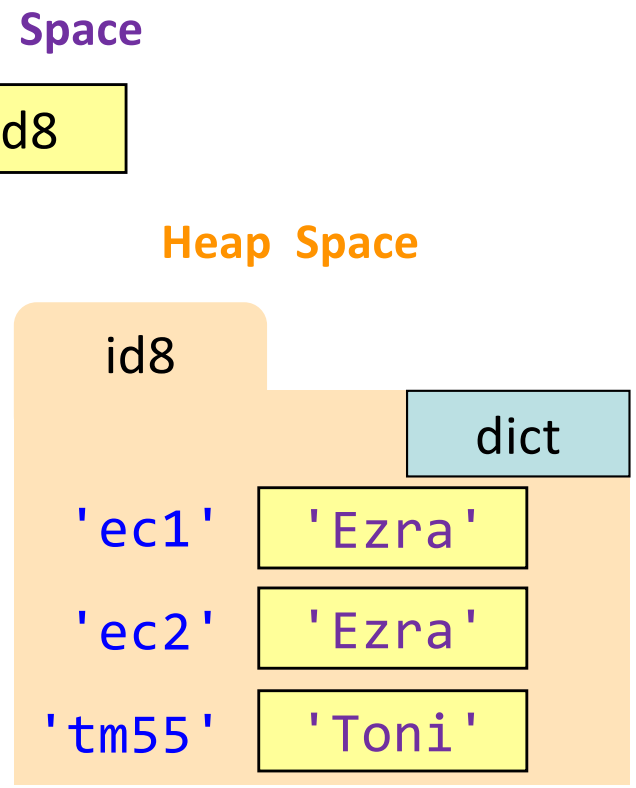
- Can access elements like a list, but…
  - Must use the key, not an index
  - Cannot slice ranges

Basic Dictionary Operations **(1-a)**

```
d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
```

1. Can reassign values

```
d['ec1'] = 'Ellis'
```

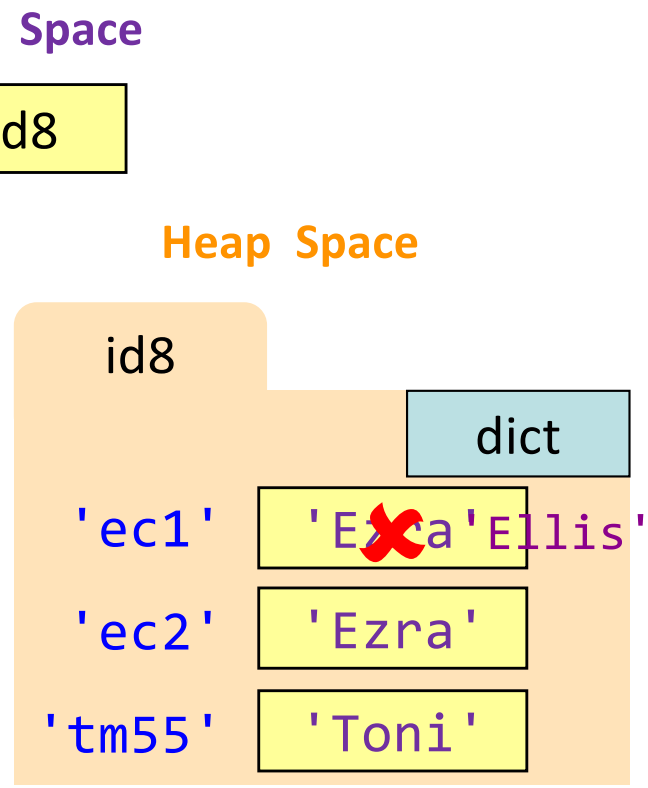


Basic Dictionary Operations **(1-b)**

```
d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
```

1. Can reassign values

```
d['ec1'] = 'Ellis'
```



Basic Dictionary Operations **(2-a)**

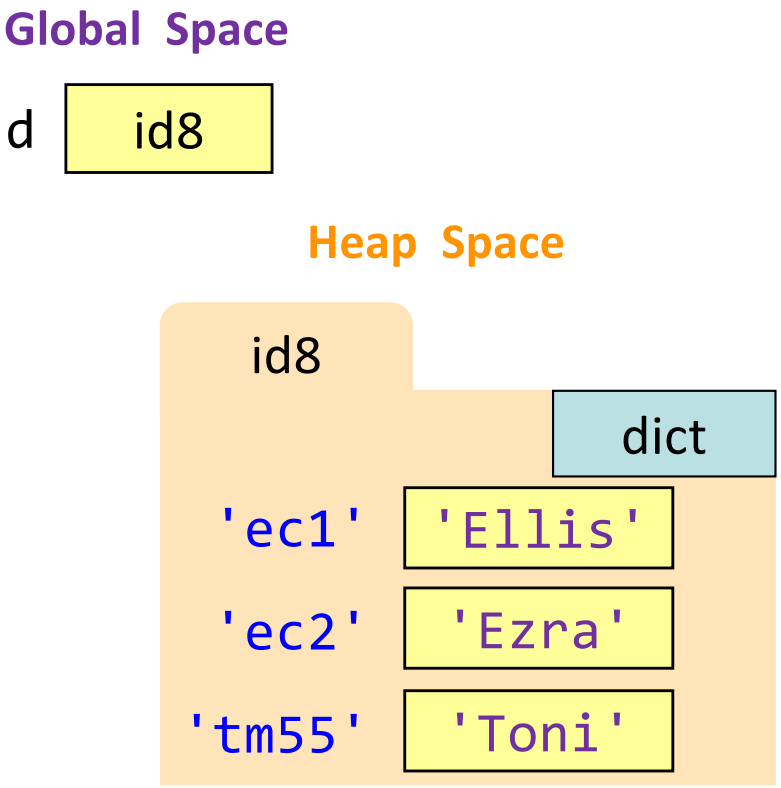
```
d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
```

1. Can reassign values

   ```
   d['ec1'] = 'Ellis'
   ```

2. Can add new keys

   ```
   d['psb26'] = 'Pearl'
   ```



Basic Dictionary Operations **(2-b)**

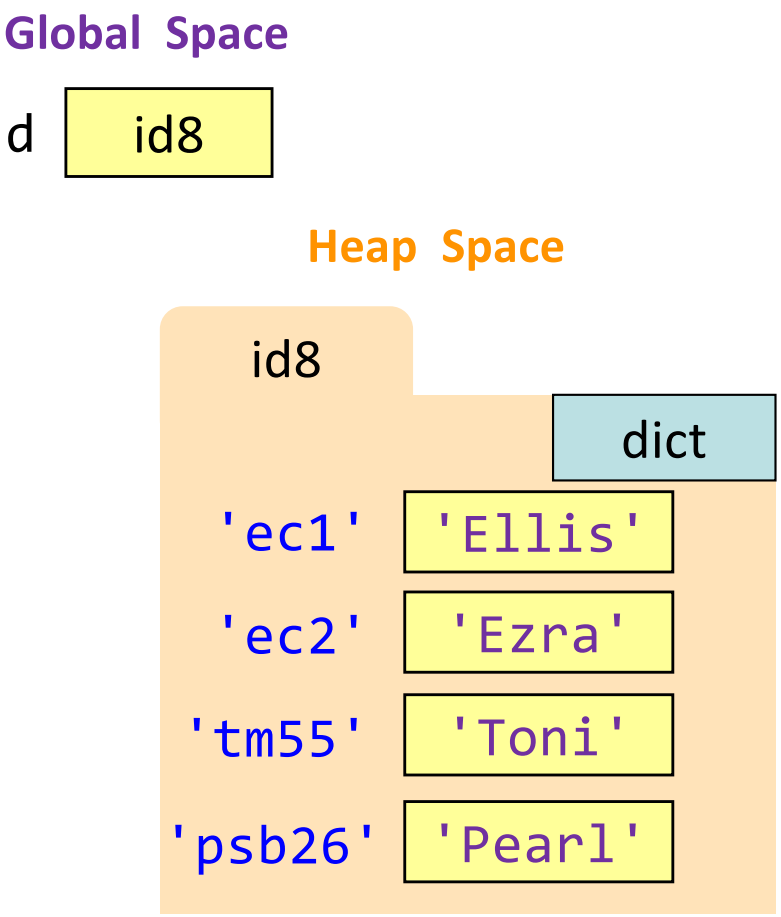
```
d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
```

1. Can reassign values

   ```
   d['ec1'] = 'Ellis'
   ```

2. Can add new keys

   ```
   d['psb26'] = 'Pearl'
   ```



Basic Dictionary Operations **(3-a)**

```
d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
```

1. Can reassign values

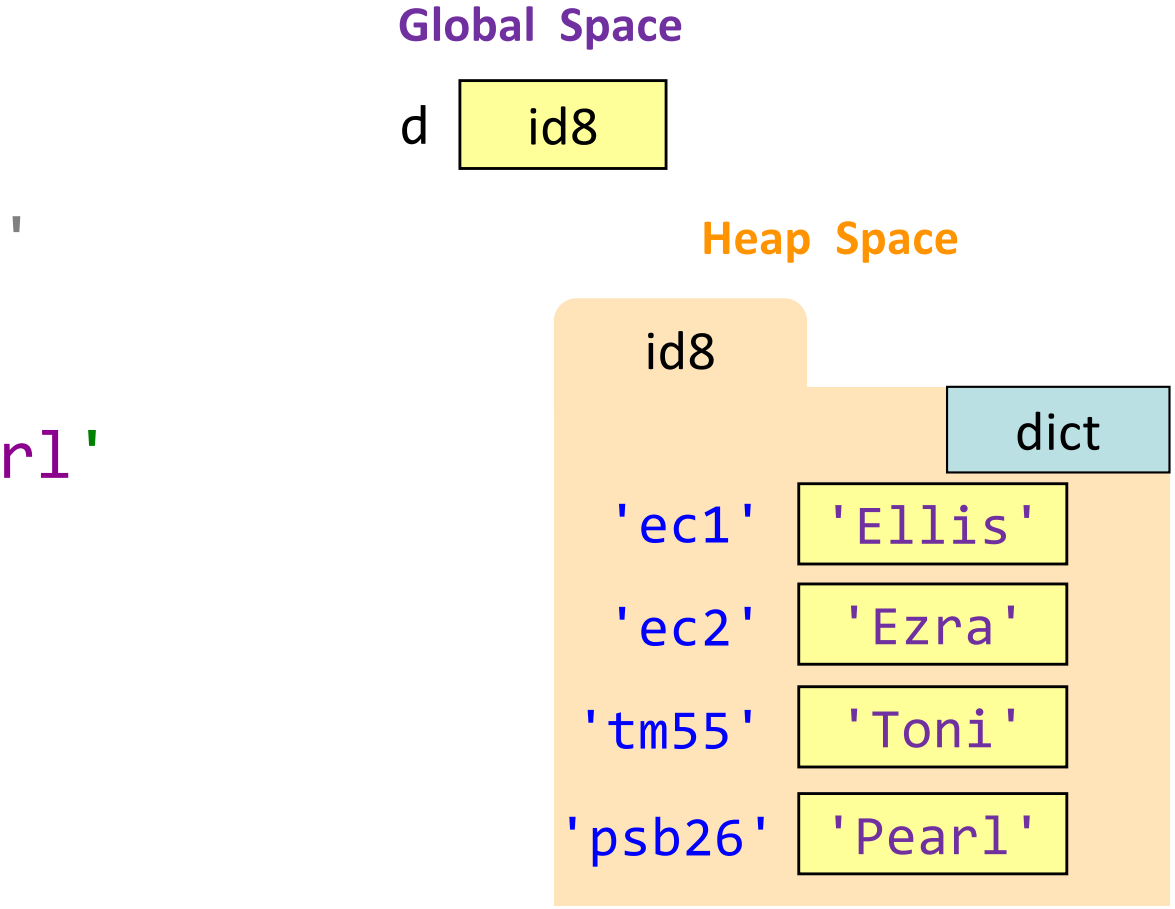
   ```
   d['ec1'] = 'Ellis'
   ```

2. Can add new keys

   ```
   d['psb26'] = 'Pearl'
   ```

3. Can delete keys

   ```
   del d['tm55']
   ```



Basic Dictionary Operations (3-b)

```
d = {'ec1':'Ezra', 'ec2':'Ezra', 'tm55':'Toni'}
```

1. Can reassign values

   ```
   d['ec1'] = 'Ellis'
   ```

2. Can add new keys

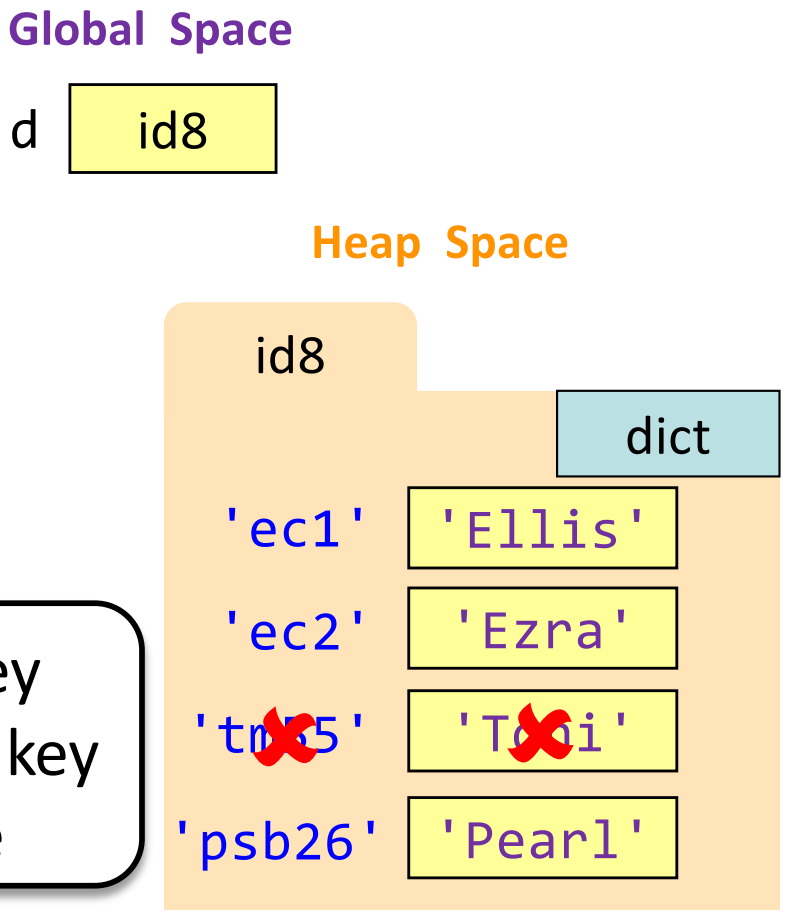
   ```
   d['psb26'] = 'Pearl'
   ```

3. Can delete keys

   ```
   del d['tm55']
   ```

Be sure to read Textbook 11.1-11.5 for additional examples!

Deleting key deletes both key and value



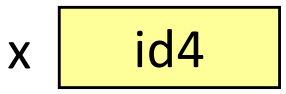
Here are the list practice questions that are in scope for the Prelim.

Q: Swap List Values?

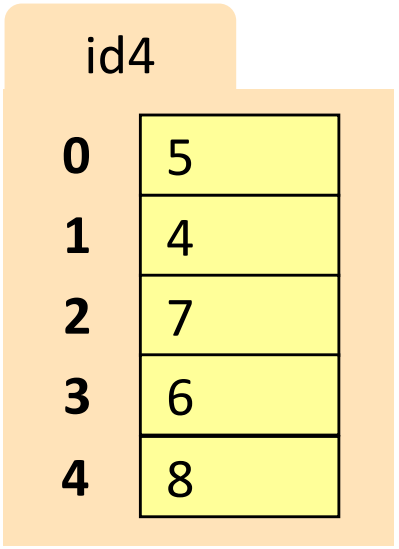
```
def swap(b, h, k):
    """Procedure swaps b[h] and b[k] in b
       Precondition: b is a mutable list, h
       and k are valid positions in the list"""”
1       temp= b[h]
2       b[h]= b[k]
3       b[k]= temp
```

```
x = [5,4,7,6,8]
swap(x, 3, 4)
print(x[3])
```

Global Space



Heap Space



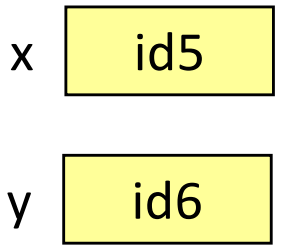
What gets printed?

A: 8
B: 6
C: Something else
D: I don’t know

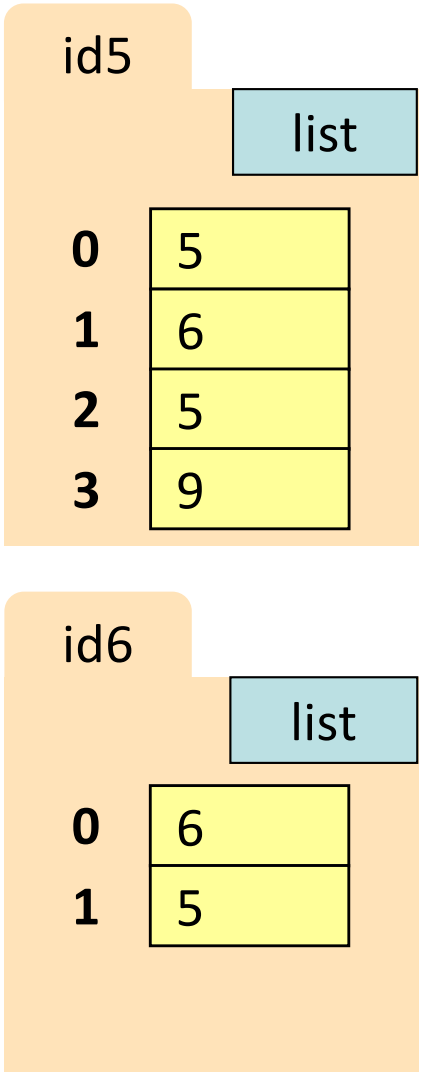
List Slices Make Copies: a slice of a list is a new list

```
x = [5, 6, 5, 9]
y = x[1:3]
```

Global Space



Heap Space



copy means new folder

Q: List Slicing

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> y = x[1:]
>>> y[0] = 7
```

- What is x[1]?

A: 7
B: 5
C: 6
D: ERROR
E: I don't know

Q4

- Execute the following:

```
>>> x = [5, 6, 5, 9, 10]
>>> y = x
>>> y[1] = 7
```

- What is x[1]?

A: 7
B: 5
C: 6
D: ERROR
E: I don't know