Presentation 16

# Nested Lists and Dictionaries

# Announcements for This Lecture

## Prelim and Regrades

- Prelim 1 is now graded
  - Solution posted in CMS
  - **Mean**: 70.5, **Median**: 74
- What are letter grades?
  - **A**: 80s (consultant level)
  - **B**: 60-79 (major level)
  - **C**: 30-55 (passing)
- Regrades are now open
  - But you can lose points!

## Other Announcements

- View the lesson videos
  - **Lesson 18** today
  - **Videos 19.1-16.7** today also
  - **Videos 20.1-20.8** next time
- Should be working on A4
  - Tasks 1-3 by Tomorrow
  - Task 4 by the Friday
  - Task 5 by Sunday

# Tables (As Nested Lists)

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3]]
```

- What is d[:1]?

| | | | |
|---|---|---|---|
| 5 | 4 | 7 | 3 |
| 4 | 8 | 9 | 7 |
| 5 | 1 | 2 | 3 |

A: 5
B: [5,4,7,3]
C: [[5,4,7,3]]
D: **ERROR**
E: I don't know

# Tables (As Nested Lists)

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3]]
```

- What is d[:1]?

| | | | |
|---|---|---|---|
| 5 | 4 | 7 | 3 |
| 4 | 8 | 9 | 7 |
| 5 | 1 | 2 | 3 |

A: 5

B: [5,4,7,3]

C: [[5,4,7,3]]

D: **ERROR**

E: I don't know

# Tables (As Nested Lists)

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3]]
```

| | | | |
|---|---|---|---|
| 5 | 4 | 7 | 3 |
| 4 | 8 | 9 | 7 |
| 5 | 1 | 2 | 3 |

- What is d[2][3]?

A: 3
B: [3]
C: 9
D: **ERROR**
E: I don't know

# Tables (As Nested Lists)

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3]]
```

- What is d[2][3]?

| | | | |
|---|---|---|---|
| 5 | 4 | 7 | 3 |
| 4 | 8 | 9 | 7 |
| 5 | 1 | 2 | 3 |

A: 3

B: [3]

C: 9

D: **ERROR**

E: I don't know

# Tables (As Nested Lists)

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3]]
```

| | | | |
|---|---|---|---|
| 5 | 4 | 7 | 3 |
| 4 | 8 | 9 | 7 |
| 5 | 1 | 2 | 3 |

- What is d[3][2]?

A: 3
B: [3]
C: 9
D: **ERROR**
E: I don't know

# Tables (As Nested Lists)

```
>>> d = [[5,4,7,3],[4,8,9,7],[5,1,2,3]]
```

| | | | |
|---|---|---|---|
| 5 | 4 | 7 | 3 |
| 4 | 8 | 9 | 7 |
| 5 | 1 | 2 | 3 |

- What is d[3][2]?

A: 3
B: [3]
C: 9
D: **ERROR**
E: I don't know

# Slices and Multidimensional Lists

- Create a nested list

```
>>> b = [[9,6],[4,5],[7,7]]
```

- Get a slice

```
>>> x = b[:2]
```

- Append to a row of x

```
>>> x[1].append(10)
```

- x now has nested list

```
[[9, 6], [4, 5, 10]]
```

- What are the contents of the list (with name) in b?

A: [[9,6],[4,5],[7,7]]
B: [[9,6],[4,5,10]]
C: [[9,6],[4,5,10],[7,7]]
D: [[9,6],[4,10],[7,7]]
E: I don't know

# Slices and Multidimensional Lists

- Create a nested list

```
>>> b = [[9,6],[4,5],[7,7]]
```

- Get a slice

```
>>> x = b[:2]
```

- Append to a row of x

```
>>> x[1].append(10)
```

- x now has nested list

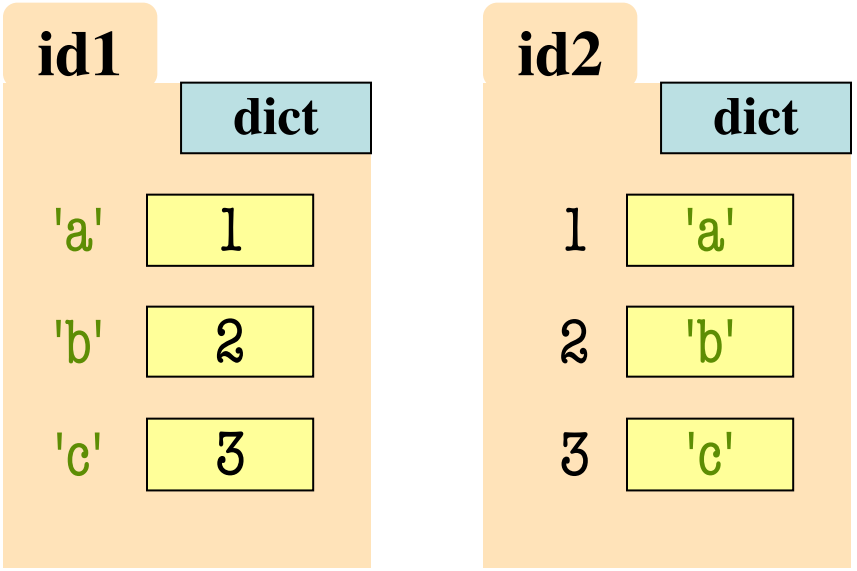
```
[[9, 6], [4, 5, 10]]
```

- What are the contents of the list (with name) in b?

A: [[9,6],[4,5],[7,7]]
B: [[9,6],[4,5,10]]
C: [[9,6],[4,5,10],[7,7]]
D: [[9,6],[4,10],[7,7]]
E: I don't know

# Dictionaries

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> g = {1:'a', 2:'b', 3:'c'}
```

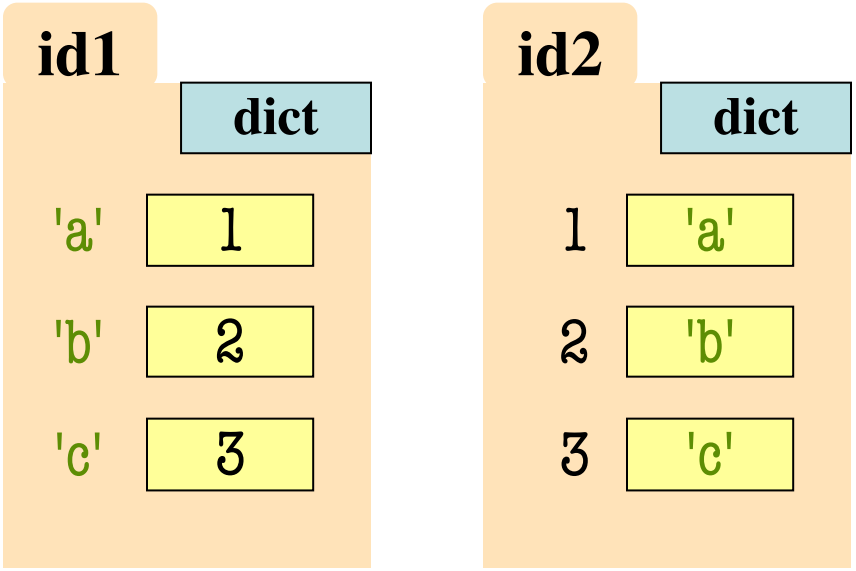



- What is g[1]?

A: 'a'
B: 'b'
C: 'c'
D: **ERROR**
E: I don't know

# Dictionaries

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> g = {1:'a', 2:'b', 3:'c'}
```

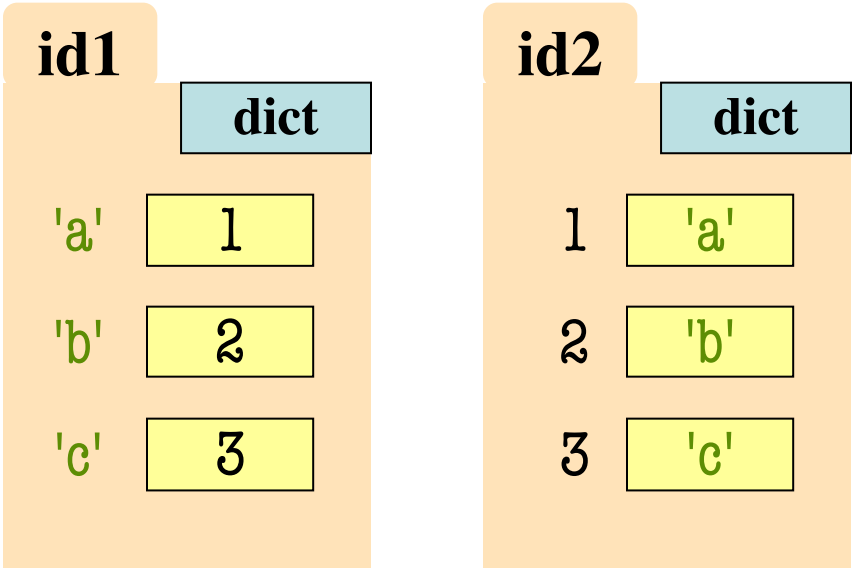



- What is g[1]?

A: 'a'
B: 'b'
C: 'c'
D: **ERROR**
E: I don't know

# Dictionaries

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> g = {1:'a', 2:'b', 3:'c'}
```

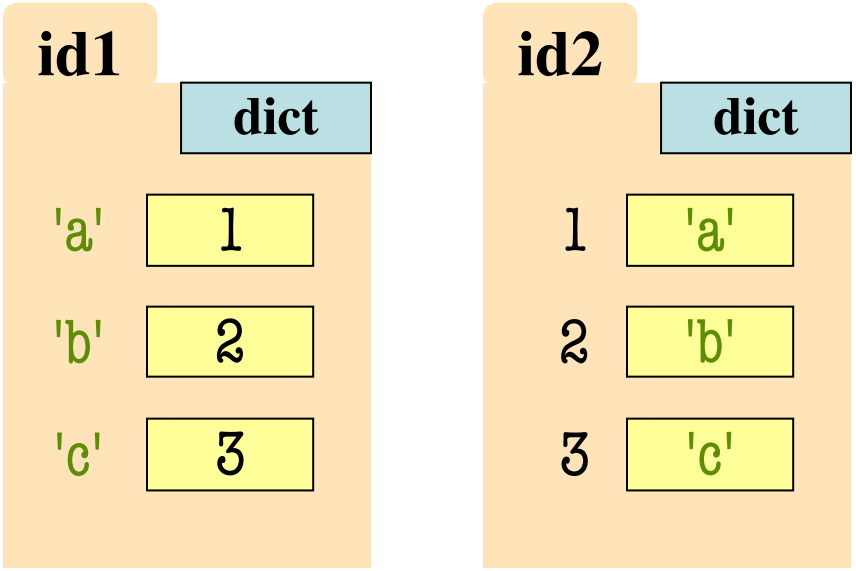



- What is g[1:3]?

A: 'a'
B: 'ab'
C: 'abc'
D: **ERROR**
E: I don't know

# Dictionaries

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> g = {1:'a', 2:'b', 3:'c'}
```

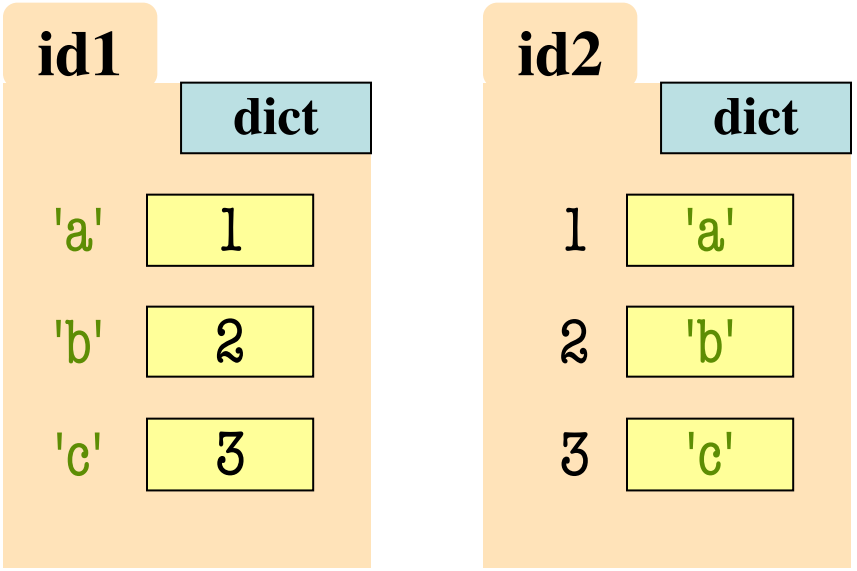



- What is g[1:3]?

A: 'a'
B: 'ab'
C: 'abc'
D: **ERROR**
E: I don't know

# Dictionaries

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> g = {1:'a', 2:'b', 3:'c'}
```

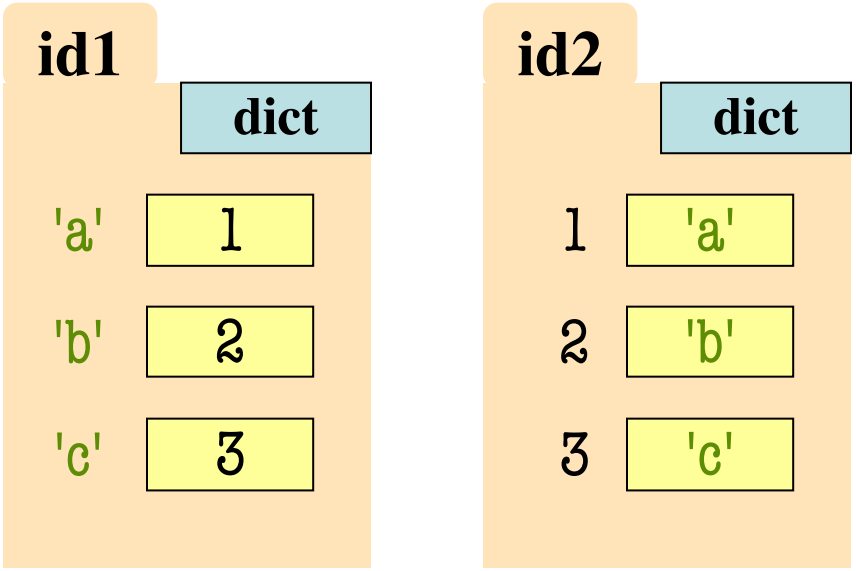



- What is (1 in d)?

A: True
B: False
C: 'a'
D: **ERROR**
E: I don't know

# Dictionaries

```
>>> d = {'a':1, 'b':2, 'c':3}
>>> g = {1:'a', 2:'b', 3:'c'}
```




- What is (1 in d)?

A: True
B: False
C: 'a'
D: **ERROR**
E: I don't know

# A Function on Nested Lists

```
def sum_columns(table):
    """
    Returns a (new) list that is the sum of all columns in table

    Example: sum_columns([[1, 2], [3, 4]) returns [4, 6]
             sum_columns([[1, 2], [3, 4], [5, 6]]) returns [9, 12]
             sum_columns([[1, 2, 3], [4, 5, 6]]) returns [5, 7, 9]

    Parameter table: the table to sum
    Precondition: table a non-empty 2d rectangular list of numbers
    """
    pass
```

# A Function on Nested Lists

```
def sum_columns(table):
    """
    Returns a (new) list that is the sum of all columns in table

    Example: sum_columns([[1, 2], [3, 4]) returns [4, 6]
             sum_columns([[1, 2],
             sum_columns([[1, 2, 3

    Parameter table: the table to su
    Precondition: table a non-empty
    """
    pass
```

**Loops over?**

A: Elements

B: Positions

C: Doesn't matter

D: Unsure

# A Function on Nested Lists

```
def sum_columns(table):
    """
    Returns a (new) list that is the sum of all columns in table

    Example: sum_columns([[1, 2], [3, 4]) returns [4, 6]
             sum_columns([[1, 2],
             sum_columns([[1, 2, 3

    Parameter table: the table to su
    Precondition: table a non-empty
    """
    pass
```

**How many loops?**

A: One

B: Two

C: Three

D: Unsure

# Another Nested List Function

```
def tablize(ragged):
    """MODIFIES ragged to convert it into a table (2d rectangular list)

    Ragged will be expanded to a table with the number of columns in
    its largest "row".  Rows that are missing elements will have 0s
    appended to the end to pad them out.

    Example:  Suppose a = [[1], [2,3,4], [5, 6]] then tablize(a)
    changes a to be [[1,0,0],[2,3,4],[5,6,0]]

    Parameter ragged: the ragged list to turn into a table
    Precondition: ragged is a non-empty 2d list of numbers"""
    pass
```

# Another Nested List Function

```
def tablize(ragged):
    """MODIFIES ragged to convert it into a table (2d rectangular list)

    Ragged will be expanded to a table with the number of columns in
    its largest "row".  Rows that are missing elements will have 0s
    appended to the end to pad them out.

    Example:  Suppose a = [[1], [2,3,4],
    changes a to be [[1,0,0],[2,3,4],[5,6,

    Parameter ragged: the ragged list t
    Precondition: ragged is a non-empt
    pass
```

**How many loops?**

A: One

B: Two

C: Three

D: Unsure

# A Function on Dictionaries

```
def merge(dict1,dict2):
    """
    Returns a new dictionary merging (joining keys) dict1 and dict2.

    If a key appears in only one of dict1 or dict2, the value is the value
    from that dictionary. If it is in both, the value is the sum of values.

    Example: merge({'a':1,'b':2},{'b':3,'c':4}) returns {'a':1,'b':5,'c':4}

    Precondition: dict1, dict2 are dictionaries with int or float values
    """
    pass
```

# A Function on Dictionaries

```
def merge(dict1,dict2):
    """
    Returns a new dictionary merging (joining keys) dict1 and dict2.

    If a key appears in only one of dict1 or dict2, the value is the value
    from that dictionary. If it is in b
    s.

    Example: merge({'a':1,'b':2},{'b'

    Precondition: dict1, dict2 are di
    """
    pass
```

**How many loops?**

A: One

B: Two

C: Three

D: Unsure

# Let's Do Recursion!

```
def histogram2(s):
    """Returns a histogram (dictionary) of the # of letters in string s.

    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.

    Example: histogram('') returns {},
             histogram('all') return {'a':1,'l':2}
             histogram('abracadabra') return {'a':5,'b':2,'c':1,'d':1,'r':2}

    Parameter s: The string to analyze
    Precondition: s is a string (possibly empty)."""
    pass
```

# Let's Do Recursion!

```
def histogram2(s):
    """Returns a histogram (dictionary) of the # of letters in string s.

    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.

    Example: histogram('') returns {},
             histogram('all') return {'a
             histogram('abracadabra')

    Parameter s: The string to analyze
    Precondition: s is a string (possibly
    pass
```

**How Divide?**

A: Cut in half

B: Pull off one elt.

C: Does not matter

D: Unsure

# Let's Do Recursion!

```
def histogram2(s):
    """Returns a histogram (dictionary) of the # of letters in string s.

    The letters in s are keys, and the count of each letter is the value. If
    the letter is not in s, then there is NO KEY for it in the histogram.

    Example: histogram('') returns {},
             histogram('all') return {'a
             histogram('abracadabra')

    Parameter s: The string to analyze
    Precondition: s is a string (possibly
    pass
```

**How Combine?**

A: Add left, right

B: Use merge fcn

C: Something trickier

D: Unsure

# Questions?