MT2 Review Pt 2



Class outline:

- Lists & dicts
- Lists in environment diagrams
- Objects

Lists & dicts

Exercise: Element comparer

Does every element equal some other element in s?

```
[-4, -3, -2, 3, 2, 4] \rightarrow False

[4, 3, 2, 3, 2, 4] \rightarrow True
```

```
def all_have_an_equal(s):
    """Does every element equal some other element in s?

>>> all_have_an_equal([-4, -3, -2, 3, 2, 4])
    False
    >>> all_have_an_equal([4, 3, 2, 3, 2, 4])
    True
    """
```

Exercise: Element comparer (Solution)

Does every element equal some other element in s?

```
[-4, -3, -2, 3, 2, 4] \rightarrow False

[4, 3, 2, 3, 2, 4] \rightarrow True
```

```
def all_have_an_equal(s):
    """Does every element equal some other element in s?

>>> all_have_an_equal([-4, -3, -2, 3, 2, 4])
    False
>>> all_have_an_equal([4, 3, 2, 3, 2, 4])
    True
    """
    return min([sum([1 for y in s if x == y]) for x in s]) > 1
    # OR
    return all([s[i] in s[:i] + s[i+1:] for i in range(len(s))])
# OR
    return all(map(lambda x: s.count(x) > 1, s))
```

Exercise: Digits dictionary

Create a dictionary mapping each digit d to the lists of elements in s that end with d.

```
[5, 8, 13, 21, 34, 55, 89] → {1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}

def digit_dict(s):
    """Map each digit d to the lists of elements in s that end with d.

>>> digit_dict([5, 8, 13, 21, 34, 55, 89])
    {1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}
    """
```

Exercise: Digits dictionary (Solution)

Create a dictionary mapping each digit d to the lists of elements in s that end with d.

Lists in environment diagrams

Starting from:

Operation

Example

Result

append adds one element to a list

$$s.append(t)$$

 $t = 0$



extend adds all elements in one list to another list

Starting from:

Operation

Example

Result

append adds one element to a list

$$s.append(t)$$

 $t = 0$



$$s \rightarrow [2, 3, [5, 6]]$$

 $t \rightarrow 0$

extend adds all elements in one list to another list



Starting from:

$$s = [2, 3]$$

 $t = [5, 6]$

Operation

Example

Result

append adds one element to a list

$$s.append(t)$$

 $t = 0$



$$s \rightarrow [2, 3, [5, 6]]$$

 $t \rightarrow 0$

extend adds all elements in one list to another list

$$s.extend(t)$$

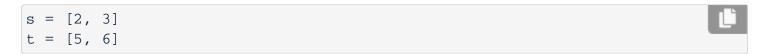
 $t[1] = 0$



$$s \rightarrow [2, 3, 5, 6]$$

t \rightarrow [5, 0]

Starting from:



Operation

Example

Result

append adds one element to a list



$$s \rightarrow [2, 3, [5, 6]]$$

 $t \rightarrow 0$

extend adds all elements in one list to another list

$$s.extend(t)$$

 $t[1] = 0$



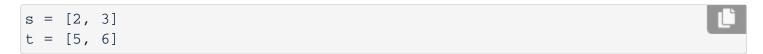
$$s \rightarrow [2, 3, 5, 6]$$

 $t \rightarrow [5, 0]$

$$s \rightarrow [2, 3]$$

 $t \rightarrow [5, 0]$
 $a \rightarrow [2, 9, [5, 0]]$
 $b \rightarrow [3, [5, 0]]$

Starting from:



Operation

Example

Result

The list constructor also creates a new list containing existing elements

t = list(s)s[1] = 0



slice assignment replaces a slice with new values

Starting from:

$$s = [2, 3]$$

 $t = [5, 6]$

Operation

Example

Result

The list constructor also creates a new list containing existing elements

$$t = list(s)$$

 $s[1] = 0$



$$s \rightarrow [2, 0]$$
$$t \rightarrow [2, 3]$$

slice assignment replaces a slice with new values

Starting from:

$$s = [2, 3]$$

 $t = [5, 6]$

Operation	Example	Result	
The list constructor also creates a new list containing existing elements	t = list(s) s[1] = 0	s → [2, 0] t → [2, 3]	
slice assignment replaces a slice with new values	s[0:0] = t s[3:] = t	s \rightarrow [5, 6, 2, 5, 6] t \rightarrow [5, 0]	

t[1] = 0

Lists in lists

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



/View in PythonTutor

```
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```



View in PythonTutor

00P

Matrix Representations

Fill in the class implementation to match the doctests.

```
class Matrix:
    0.00
   >>> m = Matrix(3, 3, [1, 0, 1, 1, 1, 1, 0, 0, 1])
   Matrix(3, 3, [1, 0, 1, 1, 1, 1, 0, 0, 1])
   >>> print(m)
   1 0 1
   1 1 1
   0 0 1
   >>> m2 = Matrix(3, 2, [124, 56, 254, 0, 100, 225])
   >>> m2
   Matrix(3, 2, [124, 56, 254, 0, 100, 225])
   >>> print(m2)
   124 56 254
    0 100 225
    def __init__(self, w, h, values):
    def __repr__(self):
    def str (self):
```

Matrix Representations (Solution)

```
class Matrix:
    0.00
   >>> m2 = Matrix(3, 2, [124, 56, 254, 0, 100, 225])
   >>> m2
   Matrix(3, 2, [124, 56, 254, 0, 100, 225])
   >>> print(m2)
   124 56 254
   0 100 225
   0.00
   def init (self, w, h, values):
       self.width = w
       self.height = h
       self.values = values
   def repr (self):
       return f"Matrix({self.width}, {self.height}, {self.values})"
   def __str__(self):
       grid_lines = []
       for h in range(self.height):
            grid_line = []
           for w in range(self.width):
                grid_line.append(str(self.values[(h * self.width) + w]))
            grid lines.append(' '.join(grid line))
       return '\n'.join(grid_lines)
```

Table Representations

```
class Table(Matrix):
   0.00
   >>> t = Table(2, 3, ['Ice Cream', 'Popularity'], ['Mint Chip', 2, 'Rocky Road', 1, 'Brownie Batter', 3
   >>> t.headers
   ['Ice Cream', 'Popularity']
   >>> t.
   Table(2, 3, ['Ice Cream', 'Popularity'], ['Mint Chip', 2, 'Rocky Road', 1, 'Brownie Batter', 3])
   >>> print(t)
   Ice Cream | Popularity
   Mint Chip 2
   Rocky Road 1
   Brownie Batter 3
   0.00
   def init (self, w, h, headers, values):
       self.headers = _____
   def __repr__(self):
   def str (self):
       header_line = _____
       divider = _____
       body = _____
       return
```

Table Representations (Solution)

```
class Table (Matrix):
    0.00
   >>> t = Table(2, 3, ['Ice Cream', 'Popularity'], ['Mint Chip', 2, 'Rocky Road', 1, 'Brownie Batt
   Table (2, 3, ['Ice Cream', 'Popularity'], ['Mint Chip', 2, 'Rocky Road', 1, 'Brownie Batter', 3])
   >>> print(t)
   Ice Cream | Popularity
   Mint Chip 2
   Rocky Road 1
    Brownie Batter 3
   def __init__(self, w, h, headers, values):
       super(). init (w, h, values)
       self.headers = headers
    def repr (self):
       return f"Table({self.width}, {self.height}, {self.headers}, {self.values})"
   def __str__(self):
       header_line = ' | '.join(self.headers)
       divider = '-' * sum([len(h) for h in self.headers])
       body = super().__str__()
       return header line + '\n' + divider + '\n' + body
```

Buttefly stages

Simulate the stages and instars of a butterfly using iterators.

```
class Butterfly:
   """ See: https://monarchwatch.org/biology/cycle1.htm
   >>> b = Butterfly()
   >>> b.stage
   'egg'
   >>> b.next_stage()
   >>> b.stage
   'larva'
   >>> b.instar
   >>> for in range(4): b.next instar()
   >>> b.instar
   >>> b.next_stage()
   >>> b.stage
   'pupa'
   >>> b.next_stage()
   >>> b.stage
   'adult'
   stages = ['egg', 'larva', 'pupa', 'adult']
   num instars = 5
   def ___init___(self):
       self.stage_iter = iter(_____)
   def next_stage(self):
       _____ = next(____)
       if _____ == 'larva':
           self.instar iter = iter( )
   def next instar(self):
       _____ = next(____)
```

Buttefly stages (Solution)

```
class Butterfly:
    """ See: https://monarchwatch.org/biology/cycle1.htm
   >>> b = Butterfly()
   >>> b.stage
    'eaa'
   >>> b.next_stage()
   >>> b.stage
   'larva'
   >>> b.instar
    >>> for _ in range(4): b.next_instar()
   >>> b.instar
   >>> b.next stage()
   >>> b.stage
    'pupa'
   >>> b.next_stage()
   >>> b.stage
    'adult'
    stages = ['egg', 'larva', 'pupa', 'adult']
    num instars = 5
    def init (self):
        self.stage iter = iter(self.stages)
        self.next_stage()
    def next_stage(self):
        self.stage = next(self.stage_iter, self.stages[-1])
        if self.stage == 'larva':
            self.instar_iter = iter(range(1, self.num_instars + 1))
            self.next instar()
    def next instar(self):
        self.instar = next(self.instar iter, self.num instars)
```