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] → False
[4, 3, 2, 3, 2, 4]    → 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] → False
[4, 3, 2, 3, 2, 4]    → 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.

```
[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]}
    """
    return {i: [x for x in s if x % 10 == i]
            for i in range(10) if any([x % 10 == i for x in s])}
    # OR
    last_digits = list(map(lambda x: x % 10, s))
    return {i: [x for x in s if x % 10 == i]
            for i in range(10) if i in last_digits}
```

Lists in environment diagrams

List operations

Starting from:

```
s = [2, 3]
t = [5, 6]
```

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	`s.extend(t)` `t[1] = 0`	

addition & slicing create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	

List operations

Starting from:

```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result
`append` adds one element to a list	`s.append(t)` `t = 0`	`s → [2, 3, [5, 6]]` `t → 0`
`extend` adds all elements in one list to another list	`s.extend(t)` `t[1] = 0`	

addition & slicing create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	

List operations

Starting from:

```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result
`append` adds one element to a list	`s.append(t)` `t = 0`	`s → [2, 3, [5, 6]]` `t → 0`
`extend` adds all elements in one list to another list	`s.extend(t)` `t[1] = 0`	`s → [2, 3, 5, 6]` `t → [5, 0]`

addition & slicing create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	

List operations

Starting from:

```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result
`append` adds one element to a list	`s.append(t)` `t = 0`	`s → [2, 3, [5, 6]]` `t → 0`
`extend` adds all elements in one list to another list	`s.extend(t)` `t[1] = 0`	`s → [2, 3, 5, 6]` `t → [5, 0]`

addition & slicing create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		`s → [2, 3]`
		`t → [5, 0]`
		`a → [2, 9, [5, 0]]`
		`b → [3, [5, 0]]`

List operations

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`	

slice assignment replaces a slice with new values	`s[0:0] = t`
	`s[3:] = t`
	`t[1] = 0`	

List operations

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`
	`t[1] = 0`	

List operations

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`
	`t[1] = 0`
		`s → [5, 6, 2, 5, 6]`
		`t → [5, 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

OOP

Matrix Representations

Fill in the class implementation to match the doctests.

```
class Matrix:
    """
    >>> m = Matrix(3, 3, [1, 0, 1, 1, 1, 1, 0, 0, 1])
    >>> m
    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:
    """
    >>> 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):
        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):
    """
    >>> 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
    """

    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):
    """
    >>> t = Table(2, 3, ['Ice Cream', 'Popularity'], ['Mint Chip', 2, 'Rocky Road', 1, 'Brownie Batt
    >>> 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
    """

    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
    1
    >>> for _ in range(4): b.next_instar()
    >>> b.instar
    5
    >>> 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
    'egg'
    >>> b.next_stage()
    >>> b.stage
    'larva'
    >>> b.instar
    1
    >>> for _ in range(4): b.next_instar()
    >>> b.instar
    5
    >>> 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)
```