CS61A Exam Prep 6

Spring 21 Midterm 2 Q5

```
class LearnableContent:
    """A base class for specific kinds of learnable content.
    All kinds have title and author attributes,
    but each kind may have additional attributes.
    """

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"{self.title} by {self.author}"

class Exercise(LearnableContent):
    """
    >>> lambda_calc = Exercise("Lambda Calculus", "Rosie F", 5)
    >>> lambda_calc.title
    'Lambda Calculus'
    >>> lambda_calc.author
    'Rosie F'
    >>> lambda_calc.num_questions
    5
    >>> str(lambda_calc)
    'Lambda Calculus by Rosie F'
    """

    def __init__(self, title, author, num_questions):
        ------------------------
        #          (a)
        ------------------------
        #          (b)
```

```
class Video(LearnableContent):
    """
    >>> vid = Video("The Golden Ratio", "Sal Khan", 881)
    >>> Video.license
    'CC-BY-NC-SA'
    >>> vid.title
    'The Golden Ratio'
    >>> vid.author
    'Sal Khan'
    >>> vid.num_seconds
    881
    >>> str(vid)
    'The Golden Ratio by Sal Khan (881 seconds)'
    """

    --------------------
    #       (c)

    def __init__(self, title, author, num_seconds):
        ------------------------
        #          (d)
        ------------------------
        #          (e)

    def __str__(self):
        ------------------------
        #          (f)
```

Fall 18 Midterm 2 Q5a

(a) (6 pt) Implement the `Poll` class and the `tally` function, which takes a choice `c` and returns a list describing the number of votes for `c`. This list contains pairs, each with a name and the number of times `vote` was called on that `choice` at the `Poll` with that name. Pairs can be in any order. Assume all `Poll` instances have distinct names. *Hint*: the dictionary `get(key, default)` method (MT 2 guide, page 1 top-right) returns the value for a `key` if it appears in the dictionary and `default` otherwise.

```
class Poll:
    s = []
    def __init__(self, n):

        self.name = ____________________________________________________________________________

        self.votes = {}

        ____________________________________________________________________________________________

    def vote(self, choice):

        self._________________________ = ____________________________________________________

def tally(c):
    """Tally all votes for a choice c as a list of (poll name, vote count) pairs.

    >>> a, b, c = Poll('A'), Poll('B'), Poll('C')
    >>> c.vote('dog')
    >>> a.vote('dog')
    >>> a.vote('cat')
    >>> b.vote('cat')
    >>> a.vote('dog')
    >>> tally('dog')
    [('A', 2), ('C', 1)]
    >>> tally('cat')
    [('A', 1), ('B', 1)]
    """

    return ____________________________________________________________________________________
```

Fall 2018 Midterm 2 Q5b

(b) (2 pt) Implement the `vote` method of the `Crooked` class, which only records every other `vote` call for each `Crooked` instance. Only odd numbered calls to `vote` are recorded, e.g., first, third, fifth, etc.

```
class Crooked(Poll):
    """A poll that ignores every other call to vote.

    >>> d = Crooked('D')
    >>> for s in ['dog', 'cat', 'dog', 'cat', 'cat']:
    ...     d.vote(s)
    >>> d.votes
    {'dog': 2, 'cat': 1}
    """
    record = True
    def vote(self, choice):
        if self.record:

            _________________ . ____________________________(___________________________________)

        self._________________ = ____________________________________________________________________
```

Fall 2019
Final Q5

Implement the `TodoList` and `Todo` classes. When a `Todo` is `complete`, it is removed from all the `TodoList` instances to which it was ever added. Track both the number of completed `Todo` instances in each list and overall so that printing a `TodoList` instance matches the behavior of the doctests below. Assume the `complete` method of a `Todo` instance is never invoked more than once.

```
class TodoList:
    """A to-do list that tracks the number of completed items in the list and overall.

    >>> a, b = TodoList(), TodoList()
    >>> a.add(Todo('Laundry'))
    >>> t = Todo('Shopping')
    >>> a.add(t)
    >>> b.add(t)
    >>> print(a)
    Remaining: ['Laundry', 'Shopping'] ; Completed in list: 0 ; Completed overall: 0
    >>> print(b)
    Remaining: ['Shopping'] ; Completed in list: 0 ; Completed overall: 0
    >>> t.complete()
    >>> print(a)
    Remaining: ['Laundry'] ; Completed in list: 1 ; Completed overall: 1
    >>> print(b)
    Remaining: [] ; Completed in list: 1 ; Completed overall: 1
    >>> Todo('Homework').complete()
    >>> print(a)
    Remaining: ['Laundry'] ; Completed in list: 1 ; Completed overall: 2
    """
    def __init__(self):
        self.items, self.complete = [], 0
    def add(self, item):
        self.items.append(item)

        ____________________________________________________________________________________
    def remove(self, item):


        ____________________________________________________________________________ += 1


        self.items.remove(_____________________________________________________________________)

    def __str__(self):
        return ('Remaining: ' + str(_____________________________________________________) +
                ' ; Completed in list: ' + str(self.complete) +


                ' ; Completed overall: ' + str(_____________________________________________))
class Todo:
    done = 0
    def __init__(self, task):
        self.task, self.lists = task, []
    def complete(self):

        ____________________________________________________________________________ += 1
        for t in self.lists:
            t.remove(self)
```

Fall 2021 Midterm 2 Q5

Definition. A *twig* is a tree that is not a leaf but whose branches are all leaves.

The `Tree` and `Link` classes appear on your midterm 2 study guide. Assume they are defined.

(a) (4.0 points)

Implement `twig`, which takes a `Tree` instance t. It returns `True` if `t` is a twig and `False` otherwise.

```
def twig(t):
    """Return True if Tree t is a twig and False otherwise.

    >>> twig(Tree(1))
    False
    >>> twig(Tree(1, [Tree(2), Tree(3)]))
    True
    >>> twig(Tree(1, [Tree(2), Tree(3, [Tree(4)])]))
    False
    """
    return _________ and _________
              (a)           (b)
```

Fall 2018 Final Q4a

(a) **(7 pt)** Implement `runts`, which takes a `Tree` instance `t` in which *every label is different* and returns a list of the labels of all runt nodes in `t`, in any order. Also implement `apply_to_nodes`, which returns nothing and is part of the implementation. Do **not** mutate any tree. The `Tree` class is on the Midterm 2 Guide.

```
def runts(t):
    """Return a list in any order of the labels of all runt nodes in t.

    >>> sorted(runts(Tree(9, [Tree(3), Tree(4, [Tree(5, [Tree(6)]), Tree(7)]), Tree(2)])))
    [2, 5, 6, 9]
    """
    result = []
    def g(node):

        if ____________________________________________________________________________:


            result.append(_____________________________________________________________)


    apply_to_nodes(_____________________________________________________________________)


    return ____________________________________________________________________________

def apply_to_nodes(f, t):
    """Apply a function f to each node in a Tree instance t."""


    ____________________________________________________________________________________

    for b in t.branches:


        ____________________________________________________________________________
```

Fall 2018 Final Q4b

(b) (4 pt) Implement `max_label`, which takes a `Tree t` and returns its largest label. Do **not** mutate any tree.

```
def max_label(t):
    """Return the largest label in t.

    >>> max_label(Tree(4, [Tree(5), Tree(3, [Tree(6, [Tree(1), Tree(2)])])]))
    6
    """
    def f(node):

        ------------------------------------------------------------------------------


        -------- max(------------, ------------, key=lambda n: -------------------------)

    apply_to_nodes(f, t)  # Assume that apply_to_nodes above is implemented correctly.

    return t.label
```