6.101 Quiz 2

Fall 2022

Name: **Answers**

Kerberos/Athena Username:

5 questions   1 hour and 50 minutes

- Please **WAIT** until we tell you to begin.

- This quiz is closed-book, but you may use one 8.5×11 sheet of paper (both sides) as a reference.

- You may **NOT** use any electronic devices (including computers, calculators, phones, etc.).

- If you have questions, please **come to us at the front** to ask them.

- Enter all answers in the boxes provided. Work on other pages with QR codes may be taken into account when assigning partial credit. **Please do not write on the QR codes.**

- If you finish the exam more than 10 minutes before the end time, please quietly bring your exam to us at the front of the room. If you finish within 10 minutes of the end time, please remain seated so as not to disturb those who are still finishing their quizzes.

- You may not discuss the details of the quiz with anyone other than course staff until final quiz grades have been assigned and released.

Worksheet (intentionally blank)

1 Getting With the Times

In the week 8 readings, we discussed the problem of filling an audio cassette tape with songs to match a given duration. Audio cassettes, though, fell out of fashion starting in the early 1990's, being replaced by *compact discs* (CDs). So here we'll explore a slightly-more-modern take on the mixtape problem, where our goal is to fill a CD's duration (typically 80 minutes) with songs. Unlike in our original formulation, however, the order of the songs will matter in this problem.

In this problem, we'll write two functions involving these "mix CDs". One of these functions is `mix_cd`, which should take as input a dictionary mapping song titles to their respective durations (represented as positive integers) and a target duration for the whole CD (also an integer). It should return a list of song names (with no repeated songs) whose durations add up to the target duration if such a list exists (and `None` otherwise). For example:

```
songs = {"A": 5, "B": 10, "C": 6, "D": 4, "E": 7}
>>> mix_cd(songs, 21)
["A", "B", "C"]  # or any other valid list, like ["D", "E", "B"]
>>> mix_cd(songs, 1000)
None
>>> mix_cd(songs, 10)
["B"]  # or any other valid list, like ["D", "C"]
```

1.1 Helper Function

First, let's write a helper function that may come in handy while working on this problem. `dict_without_key(d, k)` should take a dictionary and a key as input, and it should return a *new* dictionary that contains all of the key-value pairs from the given dictionary except the one with key `k` (if it exists). This function should always return a new dictionary, and it should not mutate its input.

Fill in the definition of this function in the box below:

```
def dict_without_key(d, k):
    return {key: val for key, val in d.items() if key != k}
```

1.2 Single Result

Next, fill in the blanks below to complete the definition of `mix_cd(songs, duration)` as described on the previous page. If any box should be left empty, cross it out clearly. You may assume that you have access to a working version of `dict_without_key`.

```
def mix_cd(songs, duration):
    if duration < 0:

        return    None

    if duration == 0:

        return    []

    for song, length in songs.items():

        arg1 =    dict_without_key(songs, song)

        arg2 =    duration - length

        rec_result = mix_cd(arg1, arg2)

        # if the recursive call produced a valid CD as output, return an appropriate value

        if rec_result    is not None    :

            return    [song] + rec_result

        else:

            pass

    return None   # not necessary, but OK
```

1.3 All Possibilities

Next, fill in the blanks below to complete the definition of `all_cds(songs, duration)`. This function takes the same inputs as `mix_cd`, but it should make a generator that produces *all* valid CDs of the given duration. If any box should be left empty, cross it out clearly. You may assume that you have access to a working version of `dict_without_key`. Note that lists containing the same elements in different orders should be considered as distinct CDs (and so your generator should produce all of them).

```
def all_cds(songs, duration):
    if duration < 0:
```

```
        return
```

```
    if duration == 0:
```

```
        yield []
        return
```

```
    for song, length in songs.items():
```

arg1 =
```
        dict_without_key(songs, song)
```

arg2 =
```
        duration - length
```

```
        rec_result = all_cds(arg1, arg2)
```

```
        for cd in rec_result:
            yield [song] + cd
```

2   String Repeater

In this problem, we will consider a small language designed for specifying repeating patterns within text. In this language, a digit within a string specifies how many times the preceding part of the string should be repeated. These parts are either single characters or groups of characters surrounded by curly braces.

We would like to take inputs in this language and output the corresponding strings, using a function called expand that we'll implement in this problem. Consider the examples below:

- expand("a") → "a"

- expand("a5") → "aaaaa"

- expand("{cat}3dog") → "catcatcatdog"

- expand("{{a2c}2d}3e4") → "aacaacdaacaacdaacaacdeeee"

Each number of repetitions will be represented by a single digit, and groupings surrounded by curly braces will always be followed by a digit.

Similar to the parser for the LISP lab, the expand function can be written to make use of a helper function that takes both a string and an index as input and returns only a small piece of the output (a single character or group, possibly repeated some number of times) as well as an index corresponding to the end of that piece, such that the definition of expand looks like:

```
def expand(s)
    out, ix = "", 0
    while ix < len(s):
        c, ix = expand_helper(s, ix)
        out += c
    return out
```

Fill in the definition of expand_helper on the facing page so that this function will work as expected. Your code only needs to work for well-formed inputs, and you may assume that digits and curly braces are never part of the text that should be repeated (i.e., they only exist to specify what should be repeated and how many times, rather than as part of a string to be repeated).

Note also that we can check whether a string consists of only digits via the isdigit method, for example:

```
>>> "a".isdigit()
False
>>> "2".isdigit()
True
>>> "abc1".isdigit()
False
>>> "1234".isdigit()
True
```

```
def expand_helper(s, ix):
    char = s[ix]
    if char == "{":
```

```
        inner = ''
        ix = ix + 1
        while s[ix] != '}':
            subexpr, ix = expand_helper(ix)
            inner += subexpr
        return inner * (int(s[ix+1])), ix + 2
```

```
    else:
```

```
        if ix < len(s)-1 and s[ix+1].isdigit():
            return token * int(s[ix+1]), ix+2
        return token, ix+1
```

3 Out of the Loop

Your friend R.E. Cursion (the "R" stands for R.E. Cursion...) was drinking coffee while working on some code, and they accidentally spilled the coffee on their keyboard! After some testing, they discovered that the coffee had broken their keyboard in a strange way, such that typing the word "`for`" is now impossible, as is typing the word "`while`." With their keyboard broken in this way they are unable to make use of Python loops; but they know that all hope is not lost, and they are determined to carry on and implement some functions anyway! Help them out by filling in the code boxes below *without using* `for` *or* `while` *loops* (including comprehensions).

3.1 Function Cascade

Given a list of functions (each of which takes a single input and returns an arbitrary output) called `funcs`, `cascade(funcs)` should produce a new function that, when called, applies each function from `funcs` to its input, in the order they're specified. For example, with a list like $[f_0, f_1, \ldots, f_N]$, `cascade(funcs)` should return a new function f' such that, for all x, the result of $f'(x)$ is the same as $f_N(f_{N-1}(\ldots f_1(f_0(x))\ldots))$. For full credit, your code should also handle the case where the input list is empty. Implement this function in the box below without using `for` or `while`.

```python
def cascade(funcs):
    if not funcs:
        return lambda x: x
    else:
        return lambda x: cascade(funcs[1:])(funcs[0](x))
```

3.2 Cleaning Zeros From the Ends of a List

Given a list L containing arbitrary values, `cleaned(L)` should create a new list containing those same values but with any leading and trailing zeros removed, without mutating the input. For example, calling `cleaned([0, 1, 0, 2, 3, 4, 0, 0, 0])` should result in `[1, 0, 2, 3, 4]`. Implement this function in the box below without using `for` or `while`.

```python
def cleaned(L):
    if not L:
        return []
    if L[0] == 0:
        return cleaned(L[1:])
    if L[-1] == 0:
        return cleaned(L[:-1])
    return L[:]
```

3.3 Generator: All Elements in a Tree

Given a tree represented as shown below, `elts(tree)` should be a generator that yields all of the elements from the tree in an arbitrary order.

A node in the tree is represented as a list with the first element being the node's value and the rest of the list being the node's children (each of which is itself a tree). That is to say, our tree structure is a nested list structure.

For example, the following code (`tree1`) and picture represent the same tree.

```
tree1 = [13,
        [7],
        [8,
          [99],
          [16,
            [77]],
          [42]]]
```
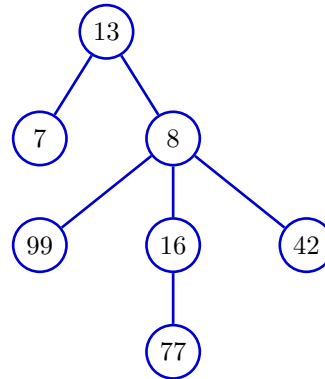

Note also that, while this example tree has integers as its elements, tree elements can be arbitrary objects. Partial credit will be given for solutions that work with integer values only, but for full credit, your code should handle the case where the elements in the tree are *arbitrary Python objects* (even lists).

Implement this function in the box on the facing page without using `for` or `while`.

```python
def elts(tree):
    if len(tree) == 1:
        yield tree[0]
        return

    yield from elts(tree[-1])
    yield from elts(tree[:-1])


# alternative solution
def elts(tree)
    def elts_from_multiple_trees(list_of_trees):
        yield from elts(list_of_trees[0])
        yield from elts_from_multiple_trees(list_of_trees[1:])

    yield tree[0]
    yield from elts_from_multiple_trees(tree[1:])
```

Worksheet (intentionally blank)

4 Game of the Year

This holiday season, everyone is scrambling to buy copies of Sorny's new smash-hit video game, in which the goal is to move a player to collect orbs of some kind. As with the Snekoban game from week 4, this game is played on a 2-D grid, and on each timestep, the player can move in one of four directions: "left", "right", "up", or "down". In this game, an orb is collected when the player moves to the same spot as an orb.

The programmers at Sorny chose to represent the game as an instance of a Game class. Each instance has two attributes:

- grid: a list of lists of lists of strings (similar to lab 4) representing the locations of the items in the game

- orbs: a single integer tracking the number of orbs the player has collected so far

Here is an example game in this representation:

```
class Game:
    def __init__(self, grid, orbs):
        self.grid = grid
        self.orbs = orbs


b = [
    [[],       [],          [],       ["orb"],   []],
    [["orb"],  ["player"],  [],       [],        []],
    [["orb"],  [],          [],       [],        []]
]


game = Game(b, 0)
```

On the following pages, we will see several pieces of code intended to create a new object called new_game, representing the game as updated after moving the player to the left and collecting the orb there. For each, we would like to predict what both the game and new_game objects will look like after that piece of code is run, starting with game defined as above.

For each attempt below, indicate the resulting values of game.grid and new_game.grid by specifying a single letter from the last two pages of this quiz (pages 25 and 27, which you may remove), or **other** if none of the grids matches, or **exception** if the code will not run to completion but will raise an exception instead.

Also indicate the number of orbs collected by specifying an integer.

4.1   Attempt 1

```
new_game = game
new_game.grid[1][0] = ["player"]
new_game.grid[1][1] = []
new_game.orbs += 1
```

game.grid (A-L, other, or exception):   [E]   game.orbs: [1]

new_game.grid (A-L, other, or exception):   [E]   new_game.orbs: [1]

4.2   Attempt 2

```
new_game = Game(game.grid.copy(), game.orbs + 1)
new_game.grid[1][0].append("player")
new_game.grid[1][1].remove("player")
```

game.grid (A-L, other, or exception):   [B]   game.orbs: [0]

new_game.grid (A-L, other, or exception):   [B]   new_game.orbs: [1]

4.3 Attempt 3

```
game_row = [[] for i in range(len(game.grid[0]))]
grid = [game_row for i in range(len(game.grid))]
grid[0][3] = ["orb"]
grid[2][0] = ["orb"]
grid[1][0] = ["player"]

new_game = Game(grid, game.orbs + 1)
```

game.grid (A-L, other, or exception): A   game.orbs: 0

new_game.grid (A-L, other, or exception): J   new_game.orbs: 1

4.4 Attempt 4

```
grid = []
for row in game.grid[:][:][:]:
    grid.append(row)
grid[0][3] = ["orb"]
grid[2][0] = ["orb"]
grid[1][0] = ["player"]

new_game = Game(None, None)
new_game.grid = grid
new_game.orbs = game.orbs + 1
```

game.grid (A-L, other, or exception): D   game.orbs: 0

new_game.grid (A-L, other, or exception): D   new_game.orbs: 1

Worksheet (intentionally blank)

5   Funcadelic

In this problem, we'll implement a special kind of dictionary-like structure built for storing integers. One feature that distinguishes these structures from dictionaries, though, is that, in addition to regular keys, a user can specify keys as *conditions*: functions that map integers to Booleans. We'll define a class called `ConditionalDict` to represent these structures. Your `ConditionalDict` should behave according to the following specification:

Your class should optionally take a dictionary as input, containing some initial mappings (which will only have integers as keys):

```
>>> x = ConditionalDict({7: 20})
>>> y = ConditionalDict()
```

Instances of the class should behave like dictionaries in the sense that we can associate integer keys with integer values:

```
>>> x[8] = 30
>>> x[7]
20
>>> x[8]
30
```

Additionally, we can specify the key as a function:

```
>>> x[lambda n: n > 5] = 10
```

After having done this kind of association, any key that is not explicitly represented in `x` but for which this function returns `True` should implicitly be associated with the value `10`. Values associated with a particular key should take precedence over values implicitly associated with that key by way of a function.

```
>>> x[6]
10
>>> x[7]
20
>>> x[8]
30
>>> x[10]
10
```

If a key is not explicitly bound to a value but it satisfies multiple conditions, the associated value should be the one from the condition that was added most recently:

```
>>> x[lambda x: 0 < x < 10] = 17
>>> x[9]
17
>>> x[11]
10
```

If a key is not explicitly associated with a value and does not satisfy any of the given conditions, trying to look it up should raise a `KeyError`.

5.1 Implementation

In the box below (and optionally on the facing page), write the code for the `ConditionalDict` class so that it behaves according to the examples on the previous page. Recall that the `x[k]` and `x[k] = v` operations can be implemented using methods called `__getitem__` and `__setitem__`, respectively. You can use `callable(f)` (which returns a Boolean) to check whether `f` is *callable*, and for this problem, it is fine to use that as your check for whether `f` is a function or not.

```python
class ConditionalDict:
    def __init__(self, vals=None):
        self.vals = dict(vals) if vals is not None else {}
        self.funcs = []

    def __setitem__(self, key, val):
        if callable(key):
            self.funcs.append((key, val))
        else:
            self.vals[key] = val

    def __getitem__(self, key):
        if key in self.vals:
            return self.vals.get(key)
        for func, val in self.funcs[::-1]:
            if func(key):
                return val
        raise KeyError
```

```
# additional space for code for ConditionalDict (if necessary)
```

5.2   Extension

Next, we would like to add support for keys to be able to be associated with values that map their inputs to other numbers. We'll implement this behavior in a new class `ConditionalMap`, which is a subclass of `ConditionalDict`.

For example, if we do the following:

```
>>> x = ConditionalMap()
>>> x[lambda n: n % 2 == 1] = lambda i: i**2
```

then any odd key that isn't explicitly mapped to a value should be associated with the square of that number. This behavior should also work for keys that are specified explicitly rather than through a condition. For example, if we do the following:

```
>>> x[7] = lambda n: n+a
>>> a = 2
```

then looking up `x[7]` should call the associated function and give the result (in this case, 9).

On the facing page, fill in the definition of `ConditionalMap`. Note that it is a subclass of `ConditionalDict`. For full credit, implement only the subset of methods that need to be implemented in `ConditionalMap`, inheriting the others from `ConditionalDict`, and try to do so while taking full advantage of code from the `ConditionalDict` class rather than reimplementing behaviors.

```
class ConditionalMap(ConditionalDict):
    def __getitem__(self, key):
        val = ConditionalDict.__getitem__(self, key)
        return val(key) if callable(val) else val
```

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Options for Problem 4 ("Game of the Year")

Option A:
```
# this is identical to the original grid
[[[],         [],          [],          ["orb"],    []],
 [["orb"],    ["player"],  [],          [],         []],
 [["orb"],    [],          [],          [],         []]]
```

Option B:
```
[[[],                  [],          [],          ["orb"],    []],
 [["orb", "player"],   [],          [],          [],         []],
 [["orb"],             [],          [],          [],         []]]
```

Option C:
```
[[[],         [],          [],          ["orb"],    []],
 [["orb"],    [],          [],          [],         []],
 [["orb"],    [],          [],          [],         []]]
```

Option D:
```
[[[],          [],          [],          ["orb"],    []],
 [["player"],  ["player"],  [],          [],         []],
 [["orb"],     [],          [],          [],         []]]
```

Option E:
```
[[[],          [],          [],          ["orb"],    []],
 [["player"],  [],          [],          [],         []],
 [["orb"],     [],          [],          [],         []]]
```

Option F:
```
[[[],                  [],          [],          ["orb"],    []],
 [["orb", "player"],   ["player"],  [],          [],         []],
 [[],                  [],          [],          [],         []]]
```

Worksheet (intentionally blank)

Worksheet (intentionally blank)

More Options for Problem 4 ("Game of the Year")

Option G:
```
[[[],                   [],         [],         ["orb"],    []],
 [["orb", "player"],  ["player"], [],         [],         []],
 [["orb"],            [],         [],         [],         []]]
```

Option H:
```
[[["orb", "player"], ["player"], [],         [],         []],
 [["orb", "player"], ["player"], [],         [],         []],
 [["orb", "player"], ["player"], [],         [],         []]]
```

Option I:
```
[[["orb", "player"], ["player"], [],       ["orb"],        []],
 [["orb", "player"], ["player"], [],       ["orb"],        []],
 [["orb", "player"], ["player"], [],       ["orb"],        []]]
```

Option J:
```
[[["player"], [], [], ["orb"], []],
 [["player"], [], [], ["orb"], []],
 [["player"], [], [], ["orb"], []]]
```

Option K:
```
[[["player"], ["player"], ["player"], ["player"], ["player"]],
 [["player"], ["player"], ["player"], ["player"], ["player"]],
 [["player"], ["player"], ["player"], ["player"], ["player"]]]
```

Option L:
```
[[["player"],          ["player"], ["player"], ["orb", "player"],  ["player"]],
 [["orb", "player"],   ["player"], ["player"], ["player"],         ["player"]],
 [["orb", "player"],   ["player"], ["player"], ["player"],         ["player"]]]
```

Worksheet (intentionally blank)