Lecture 12: Containers

Hi!

I’m E! (just the letter yeah)

Third year EECS major

5th semester on course staff

Hobbies: music (guitar), video games, puzzles (especially crosswords), food, teaching!

Announcements

Cats!

Box-and-Pointer Notation

Box-and-Pointer Notation

Lists are represented as a row of index-labeled adjacent boxes, one per element Each box either contains a primitive value or points to a compound value

```
1  one_two = [1, 2]
2  nested = [[1, 2], [],
3            [[3, False, None],
4             [4, lambda: 5]]]
```

Edit code in Online Python Tutor

< Back End Forward >

line that has just executed
next line to execute

Box-and-Pointer Notation

What's the environment diagram? What gets printed here?

```
def f(s):
    x = s[0]
    return [x]

t = [3, [2+2, 5]]
u = [f(t[1]), t]
print(u)
```

pythontutor.com

Double Eights With a List

Implement double_eights, which takes a list `s` and returns whether two consecutive items are both 8.

using positions (indices)...

```
def double_eights(s):
    """Return whether two consecutive items
    of list s are 8.

    >>> double_eights([1, 2, 8, 8])
    True
    >>> double_eights([8, 8, 0])
    True
    >>> double_eights([5, 3, 8, 8, 3, 5])
    True
    >>> double_eights([2, 8, 4, 6, 8, 2])
    False
    """
    for ____________________________________:

        if ____________________________________:
            return True
    return False
```

using slices...

```
def double_eights(s):
    """Return whether two consecutive items
    of list s are 8.

    >>> double_eights([1, 2, 8, 8])
    True
    >>> double_eights([8, 8, 0])
    True
    >>> double_eights([5, 3, 8, 8, 3, 5])
    True
    >>> double_eights([2, 8, 4, 6, 8, 2])
    False
    """
    if _____________ == [ _____ ]:
        return True
    elif len(s) < 2:
        return False
    else:
        return ____________________________
```

Processing Container Values

Aggregation

Several built-in functions take iterable arguments and aggregate them into a value

- **sum**(iterable[, start]) -> value

  Return the sum of an iterable (not of strings) plus the value
  of parameter 'start' (which defaults to 0). When the iterable is
  empty, return start.

Summation

```
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence.

    >>> summation(5, cube)
    225
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

Built-in aggregations:

- **sum**(iterable[, start]) -> value

  Return the sum of an iterable plus the value of parameter 'start' (which defaults to 0).

- **max**(iterable[, key=func]) -> value
  max(a, b, c, ...[, key=func]) -> value

  With a single iterable argument, return its largest item.

- **all**(iterable) -> bool

  Return True if bool(x) is True for all values x in the iterable.

Strings (Demo)

Tree Recursion With Strings

(Modified) Spring 2023 Midterm 2 Question 5(a):

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **count_park**, which returns the number of ways that vehicles can be parked in n adjacent parking spots for positive integer n. Some or all spots can be empty.

```
def count_park(n):
    """Count the ways to park cars and motorcycles in n adjacent spots.
    >>> count_park(1)  # '.' or '%'
    2
    >>> count_park(2)  # '..', '.%', '%.', '%%', or '<>'
    5
    >>> count_park(4)  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
    if n < 0:
        return _____
    elif n == 0:
        return _____
    else:
        return _____ _____ _____
```

(Modified) Spring 2023 Midterm 2 Question 5(b):

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **park**, which returns a list of all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n. Spots can be empty.

```
def park(n):
    """Return the ways to park cars and motorcycles in n adjacent spots.
    >>> park(1)
    ['%', '.']
    >>> park(2)
    ['%%', '%.', '.%', '..', '<>']
    >>> len(park(4))  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
    if n < 0:
        return _______
    elif n == 0:
        return _______
    else:
        return ____________________________________________________________
```

Dictionaries

Dictionaries

A dictionary contains key-value pairs, where both the keys and values are objects

```
>>> numerals = {'I': 1.0, 'V': 5, 'X': 10}
```

We can index into a dictionary using the **keys,** which outputs the corresponding **value**

```
>>> numerals['X']
10
```

Like lists, we can update the values at an index. We can also add new key-value pairs

```
>>> numerals['I'] = 1
>>> numerals['L'] = 50
>>> numerals
{'I': 1, 'X': 10, 'L': 50, 'V': 5}
```

Dictionaries

We can create entire dictionaries using **dictionary comprehension** (similar to lists)

```
{<key exp>:<value exp> for <name> in <iter exp> if <filter exp>}
```

```
Short version: {<map exp> for <name> in <iter exp>}
```

Another way to index into a dictionary is by using **`.get()`**, where you can specify a default value if the key does not exist in the dictionary.

```
>>> d = {"I": 1, "V": 5, "X": 10}
>>> d.get("V", 0)
5
>>> d.get("M", 0)
0
```

Colorings - Inspired by this video!

Definition: A **coloring** is a mapping (dictionary) of the numbers 1...n to a fixed list of colors.

```
Example:
    n = 4, colors = ["red", "blue"]
    Coloring: {1: "red", 2: "blue", 3: "red", 4: "blue"}
```

A **Schur Coloring** is a coloring with a special property:

- For any triplet of numbers **a, b, and c,** if **a + b = c,** then **a, b, and c are not all the same color.**
- **a, b, and c do not need to be unique numbers!** For example, because 1 + 1 = 2, 1 and 2 must be different colors.

Colorings

Implement **`schur_colorings`**, a function takes an integer **`n`** and list of strings **`colors`**. It returns a list of all possible Schur Colorings for the numbers from `1` to **`n`** with the colors in **`colors`**

```
def schur_colorings(n: int, colors: list[str]):
    """
    >>> list(schur_colorings(4, ["Red", "Blue"]))
    [{1: 'Red', 2: 'Blue', 3: 'Blue', 4: 'Red'}, {1: 'Blue', 2: 'Red', 3: 'Red', 4: 'Blue'}]
    >>> len(list(schur_colorings(13, ["Red", "Blue", "Green"])))
    18
    >>> list(schur_colorings(14, ["Red", "Blue", "Green"]))
    []
    """
    if n == 0:
        return [{}]
    colorings = []
    for previous_coloring in _______:
        # What colors can n not be? (Hint: i and n-i always sums to n)
        banned_colors = [previous_coloring[i] for i in range(1, n // 2 + 1) if _______]
        # Create new colorings (Hint: use .get())
        new_colorings = [{i: _______ for i in range(1, n+1)} for color in colors if ________]
        colorings += new_colorings
    return colorings
```

Colorings

Implement **`schur_colorings`**, a function takes an integer **`n`** and list of strings **`colors`**. It returns a list of all possible Schur Colorings for the numbers from `1` to **`n`** with the colors in **`colors`**

```
def schur_colorings(n: int, colors: list[str]):
    """
    >>> list(schur_colorings(4, ["Red", "Blue"]))
    [{1: 'Red', 2: 'Blue', 3: 'Blue', 4: 'Red'}, {1: 'Blue', 2: 'Red', 3: 'Red', 4: 'Blue'}]
    """
    if n == 0:
        return [{}]
    colorings = []
    for previous_coloring in schur_colorings(n-1, colors):
        banned_colors = [
            previous_coloring[i] for i in range(1, n // 2 + 1)
            if previous_coloring[i] == previous_coloring[n - i]
        ]
        new_colorings = [
            {i: previous_coloring.get(i, color) for i in range(1, n+1)}
            for color in colors if color not in banned_colors
        ]
        colorings += new_colorings
    return colorings
```