Lecture 11: Sequences

hi, i’m sriya :)

Announcements

Lab 3 is due TONIGHT

Homework 3 is due tomorrow

Resources

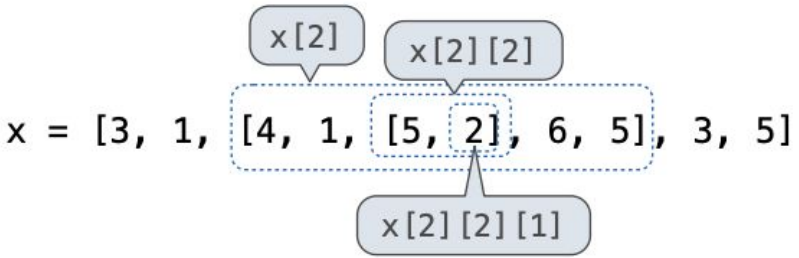
- Sign up for Quest Exam Support Meetings
- Sign up for an Exam Prep Tutoring Section on sections.cs61a.org
- OH today and tomorrow in Warren from 5-8 PM

Lists

```
digits = [8, 0, 8, 1]
Index              0  1  2  3
Index (negative)  -4 -3 -2 -1
```

```
>>> digits[2]
8

>>> digits[-1]
1
```



Write an expression to get the 2: `x[2][2][1]`

For Loops

demo!

Ranges

A range is a sequences of consecutive integers.*

…, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, …

`range(-2, 2)`

Interesting Properties:

- **Length**: ending value - starting value
- **Element Selection**: starting value + index

```
>>> list(range(-2, 2))
[-2, -1, 0, 1]
>>> list(range(4))
[0, 1, 2, 3]
```

list constructor

range with a 0 starting value

* Ranges can be represent more general integer sequences.

List Comprehensions

```
[<map exp> for <name> in <iter exp> if <filter exp>]
```

```
Short version: [<map exp> for <name> in <iter exp>]
```

Example: Evens

```
def evens(n: int) -> list[int]:
    """Return a list of the first n even numbers.

    >>> evens(0)
    []
    >>> evens(3)
    [0, 2, 4]
    """
    return ____________________________
```

Example: Evens

```
def evens(n: int) -> list[int]:
    """Return a list of the first n even numbers.

    >>> evens(0)
    []
    >>> evens(3)
    [0, 2, 4]
    """
    return [2 * x for x in range(n)]
```

Example: Two Lists

Given these two related lists of the same length:

```
xs = list(range(-10, 11))
ys = [x*x - 2*x + 1 for x in xs]
```

Write a list comprehension that evaluates to: A list of all the x values (from `xs`) for which the corresponding y (from `ys`) is below 10.

```
>>> list(xs)
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10]
>>> ys
[121, 100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49,
64, 81]
>>> xs_where_y_is_below_10
[-2, -1, 0, 1, 2, 3, 4]
```

Example: Promoted

Implement `promoted`, which takes a sequence `s` and a one-argument function `f`. It returns a list with the same elements as `s`, but with all elements `e` for which `f(e)` is a true value ordered first. Among those placed first and those placed after, the order stays the same.

```
def promoted(s, f):
    """Return a list with the same elements as s, but with all
    elements e for which f(e) is a true value placed first.

    >>> promoted(range(10), odd)  # odds in front
    [1, 3, 5, 7, 9, 0, 2, 4, 6, 8]
    """
    return _____
```

Example: Promoted

Implement `promoted`, which takes a sequence `s` and a one-argument function `f`. It returns a list with the same elements as `s`, but with all elements `e` for which `f(e)` is a true value ordered first. Among those placed first and those placed after, the order stays the same.

```
def promoted(s, f):
    """Return a list with the same elements as s, but with all
    elements e for which f(e) is a true value placed first.

    >>> promoted(range(10), odd)  # odds in front
    [1, 3, 5, 7, 9, 0, 2, 4, 6, 8]
    """
    return [e for e in s if f(e)] + [e for e in s if not f(e)]
```

List, Slices, & Recursion

For any list `s`, the expression `s[1:]` is called a slice from index 1 to the end (or 1 onward)

- The value of `s[1:]` is a list whose length is one less than the length of `s`
- It contains all of the elements of `s` except `s[0]`
- Slicing `s` doesn't affect `s`

```
>>> s = [2, 3, 6, 4]
>>> s[1:]
[3, 6, 4]
>>> s
[2, 3, 6, 4]
```

In a list `s`, the first element is `s[0]` and the rest of the elements are `s[1:]`.

Example: Reverse

```
def reverse(s):
    """Return s in reverse order.

    >>> reverse([4, 6, 2])
    [2, 6, 4]
    """
    if not s:
        return []
    return _____
```

there are multiple ways to do this!

Example: Reverse

```
def reverse(s):
    """Return s in reverse order.

    >>> reverse([4, 6, 2])
    [2, 6, 4]
    """
    if not s:
        return []
    return reverse(s[1:]) + [s[0]]
```

there are multiple ways to do this!

More Examples!

```
def multiplication_table(n):
    """
    Return a 2D list (a list of lists) representing a multiplication
    table for digits 1 through n.
    >>> multiplication_table(3)
    [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
    """
    # YOUR CODE HERE
```

it helps to write the list in a different way!

```
[[1,  2,  3],   row 1
 [2,  4,  6],   row 2
 [3,  6,  9]]   row 3

col 1  col 2  col 3
```

More Examples!

```
def multiplication_table(n):
    """
    Return a 2D list (a list of lists) representing a multiplication
    table for digits 1 through n.
    >>> multiplication_table(3)
    [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
    """
    my_table = [[0] * n for _ in range(n)]
    for i in range(n):          ← outer loop
        for j in range(n):      ← inner loop
            my_table[i][j] = (i + 1) * (j + 1)
    return my_table
```

```
think of it like this:
my_table = [[0] * n for _ in range(n)]
for row in range(rows):
    for col in range(columns):
        my_table[row][col] = row * col
return my_table
```

there are multiple ways to do this! can you think of another?

More Examples!

```
def multiplication_table(n):
    """
    Return a 2D list (a list of lists) representing a multiplication
    table for digits 1 through n.
    >>> multiplication_table(3)
    [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
    """
    return ____________________________________________________________
```

More Examples!

```
def multiplication_table(n):
    """
    Return a 2D list (a list of lists) representing a multiplication
    table for digits 1 through n.
    >>> multiplication_table(3)
    [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
    """
    return [[i * j for j in range(1, n + 1)] for i in range(1, n + 1)]
```

More Examples!

```
def shuffle(deck1, deck2):
    """
    Interleave two decks of any length together.
    >>> odds = [1, 3, 5]
    >>> evens = [2, 4, 6, 8]
    >>> shuffle(odds, evens)
    [1, 2, 3, 4, 5, 6, 8]
    """
    # YOUR CODE HERE
```

how do we solve this with a for loop?

what do we need to consider?

1) deck1 is longer
2) deck2 is longer
3) both decks are the same length

More Examples!

how do we solve this with a for loop?

what do we need to consider?

1) deck1 is longer
2) deck2 is longer
3) both decks are the same length

```
def shuffle(deck1, deck2):
    """
    Interleave two decks of any length together.
    >>> odds = [1, 3, 5]
    >>> evens = [2, 4, 6, 8]
    >>> shuffle(odds, evens)
    [1, 2, 3, 4, 5, 6, 8]
    """
    shuffled_deck = []
    min_len = min(len(deck1), len(deck2))
    for i in range(min_len):
        shuffled_deck += [deck1[i], deck2[i]]
    shuffled_deck += deck1[min_len:]
    shuffled_deck += deck2[min_len:]
    return shuffled_deck
```

← why do we need to use min?

More Examples!

```
def shuffle(deck1, deck2):
    """
    Interleave two decks of any length together.
    >>> odds = [1, 3, 5]
    >>> evens = [2, 4, 6, 8]
    >>> shuffle(odds, evens)
    [1, 2, 3, 4, 5, 6, 8]
    """
    # YOUR CODE HERE
```

now can we solve this with recursion?

what do we need to consider?

1) deck1 is longer
2) deck2 is longer
3) both decks are the same length

More Examples!

```
def shuffle(deck1, deck2):
    """
    Interleave two decks of any length together.
    >>> odds = [1, 3, 5]
    >>> evens = [2, 4, 6, 8]
    >>> shuffle(odds, evens)
    [1, 2, 3, 4, 5, 6, 8]
    """
    # base cases
    if not deck1:
        return deck2
    if not deck2:
        return deck1
    # recursive call
    return [deck1[0], deck2[0]] + shuffle(deck1[1:], deck2[1:])
```

shuffle([], [3, 4]) → [3, 4]

shuffle([1, 2], []) → [1, 2]

now can we solve this with recursion?

what do we need to consider?

1) deck1 is longer
2) deck2 is longer
3) both decks are the same length

Feedback :)

go.cs61a.org/sriya-lecture