



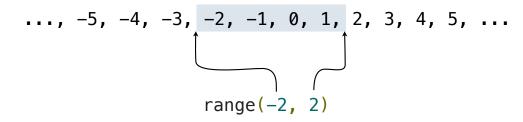
Lists

['Demo']



The Range Type

A range is a sequence of consecutive integers.*

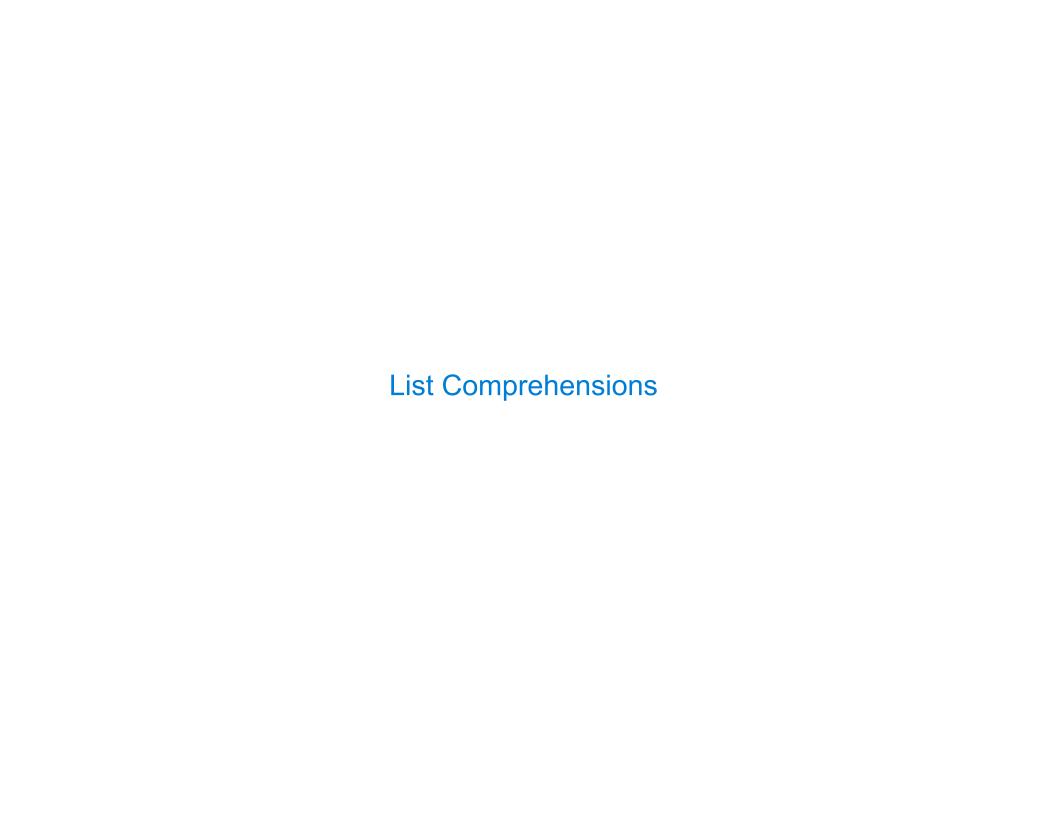


Length: ending value - starting value

(Demo)

Element selection: starting value + index

* Ranges can actually 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: Two Lists

```
Given these two related lists of the same length:

xs = 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]
```

8

Example: Promoted

First in Line

Implement **promoted**, which takes a sequence \mathbf{s} and a one-argument function \mathbf{f} . It returns a list with the same elements as \mathbf{s} , but with all elements \mathbf{e} for which $\mathbf{f}(\mathbf{e})$ 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)]
```

Lists, Slices, & Recursion

A List is a First Element and the Rest of the List

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:].

Recursion Example: Sum

Implement **sum_list**, which takes a list of numbers s and returns their sum. If a list is empty, the sum of its elements is 0.

```
def sum_list(s):
    """Sum the elements of list s.

>>> sum([2, 4, 1, 3])
    10
    """

if len(s) == 0:
    return 0

else:
    return __s[0] _ + __sum_list(s[1:])
```

Recursive idea: The sum of the elements of a list is the result of adding the first element to the sum of the rest of the elements

Recursion Example: Large Sums

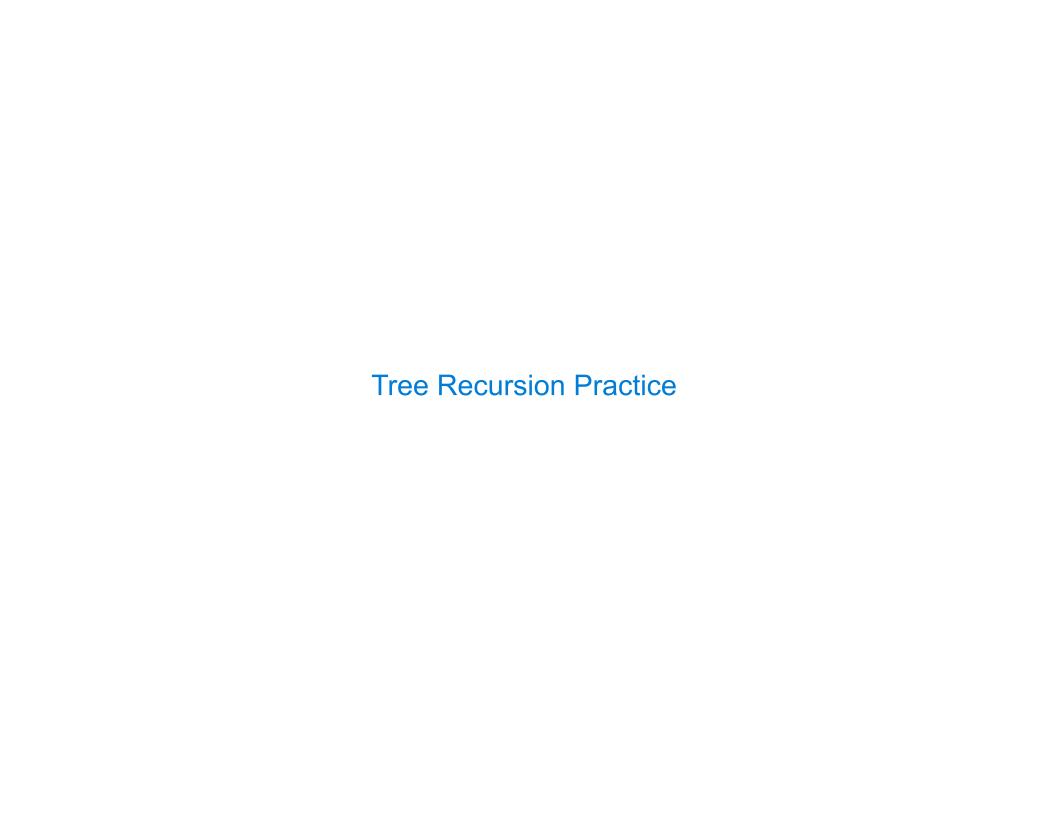
Definition: A sublist of a list s is a list with some (or none or all) of the elements of s.

Implement large, which takes a list of positive numbers s and a non-negative number n.

It returns the sublist of **s** with the largest sum that is less than or equal to **n**.

You may call **sum_list**, which takes a list and returns the sum of its elements.

```
def large(s, n):
    """Return the sublist of positive numbers s with the
    largest sum that is less than or equal to n.
    >>> large([4, 2, 5, 6, 7], 3)
    [2]
    >>> large([4, 2, 5, 6, 7], 8)
    [2, 6]
    >>> large([4, 2, 5, 6, 7], 19)
    [4, 2, 6, 7]
    >>> large([4, 2, 5, 6, 7], 20)
    [2, 5, 6, 7]
    if s == []:
        return []
    elif s[0] > n:
        return large(s[1:], n)
    else:
        first = s[0]
                     [first] + large(s[1:], n - first)
        with s0 =
                              large(s[1:], n)
        without_s0 =
        if sum list(with s0) > sum list(without s0):
            return with s0
        else:
            return without s0
```



Spring 2023 Midterm 2 Question 5

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.