Sequences

Announcements

Exam Prep Sections!

Lists

```
['Demo']
```

List Indexing

```
digits = [8, 0, 8, 1]

         Index   0  1  2  3
Index (negative) -4 -3 -2 -1
```

```
>>> digits[2]
8

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

```
x = [3, 1, [4, 1, [5, 2], 6, 5], 3, 5]
```

```
Write an expression to get the 2: _________________
```

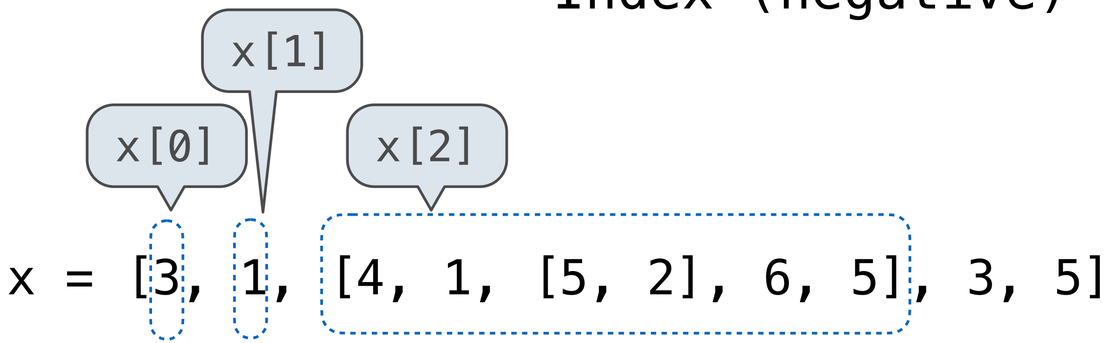
List Indexing

```
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: _____

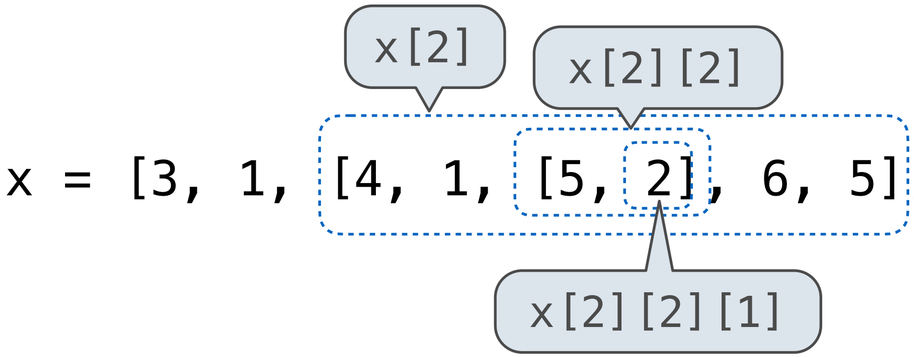
List Indexing

```
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]

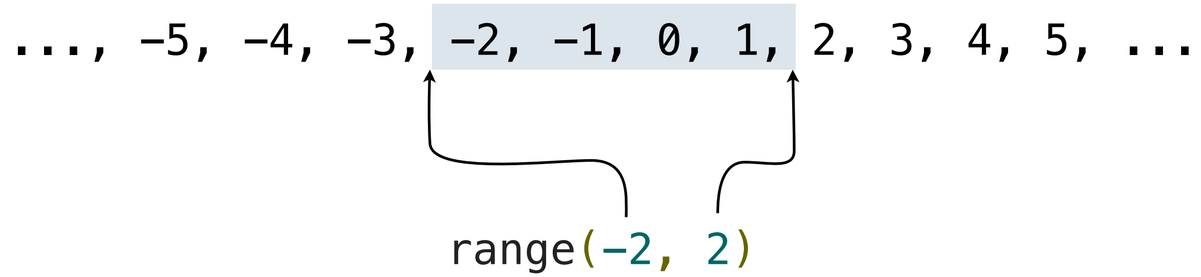
For Loops

(Demo)

Ranges

The Range Type

A range is a sequence of consecutive integers.*



Length: ending value - starting value

(Demo)

Element selection: starting value + index

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

List constructor

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

Range with a 0 starting value

* Ranges can actually represent more general integer sequences.

List Comprehensions

(Demo)

List Comprehensions

A name you choose

```
[<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 _____________________________
```

Write down an example!

evens(3):

list(range(3)): [0, 1, 2]

Return value: [0, 2, 4]

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)]
```

Write down an example!

evens(3):

list(range(3)): [0, 1, 2]

Return value: [0, 2, 4]

Example: Promoted

First in Line

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 _____________________________________________________
```

Write down an example!

Started with: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Result: [1, 3, 5, 7, 9, 0, 2, 4, 6, 8]

First in Line

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)]
```

Write down an example!

Started with: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Result: [1, 3, 5, 7, 9, 0, 2, 4, 6, 8]

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: Reverse

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

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

Write down an example!

reverse([4, 6, 2])

What happens if you call the function on everything except the first element?

reverse([6, 2]) + [4]

[2, 6]

reverse([6, 2])
returns [2, 6]

returns [2, 6, 4]

Recursion 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]]
```

Write down an example!

reverse([4, 6, 2])

What happens if you call the function on everything except the first element?

reverse([6, 2]) + [4]

[2, 6]

reverse([6, 2])
returns [2, 6]

returns [2, 6, 4]

Recursion Example: All Possible Sums

```
def sums(n: int) -> list[list[int]]:
    """Return a list of all of the possible lists of
    positive integers whose elements add up to n.

    >>> sums(3)
    [[1, 1, 1], [1, 2], [2, 1], [3]]
    """
    result = []
    for first in range(1, n):  1, 2

        result = result + _____

    return result + [[n]]
```

Write down an example!

result: []
first: 1

Make some return values where 1 is first?

[[1, 1, 1], [1, 2]]

sums(2): [[1, 1], [2]]

What expression can we write that results in [1, 1, 1], [1, 2]? sums(2)

Recursion Example: All Possible Sums

```
def sums(n: int) -> list[list[int]]:
    """Return a list of all of the possible lists of
    positive integers whose elements add up to n.

    >>> sums(3)
    [[1, 1, 1], [1, 2], [2, 1], [3]]
    """
    result = []
    for first in range(1, n):  1, 2
        result = result + [[first] + rest for rest in sums(n - first)]
    return result + [[n]]
```

Write down an example!

result: []
first: 1

Make some return values where 1 is first?

[[1, 1, 1], [1, 2]]

sums(2): [[1, 1], [2]]

What expression can we write that results in [1, 1, 1], [1, 2]?

[[1] + rest for rest in sums(2)]