Sequences

Announcements

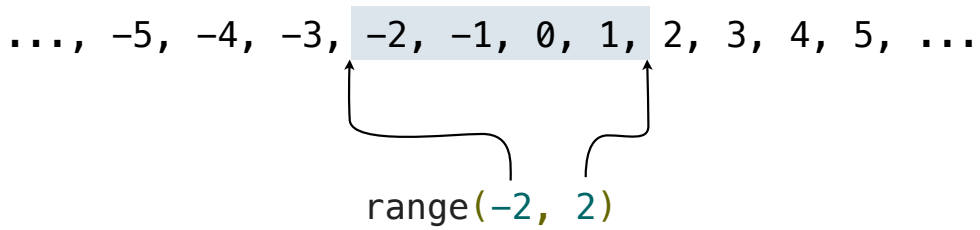
Lists

```
['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

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]
```

Reference: Sequence Operations

Operation	Result
`x in s`	True if an item of *s* is equal to *x*, else False
`x not in s`	False if an item of *s* is equal to *x*, else True
`s + t`	the concatenation of *s* and *t*
`s * n or n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

Strings as Sequences

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 [e for e in s if f(e)] + [e for e in s if not f(e)]
```

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)]
```

Example: Twenty-One

Twenty-One Rules

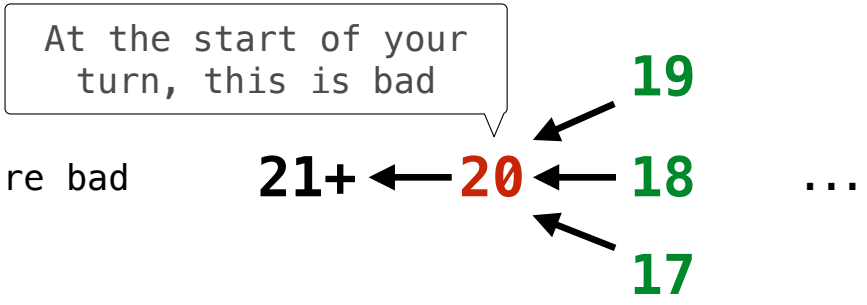
Two players alternate turns, on which they can add 1, 2, or 3 to the current total

The total starts at 0

The game end whenever the total is 21 or more

The last player to add to the total loses

Some states are good; some are bad



(Demo)