

Containers

Announcements

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.

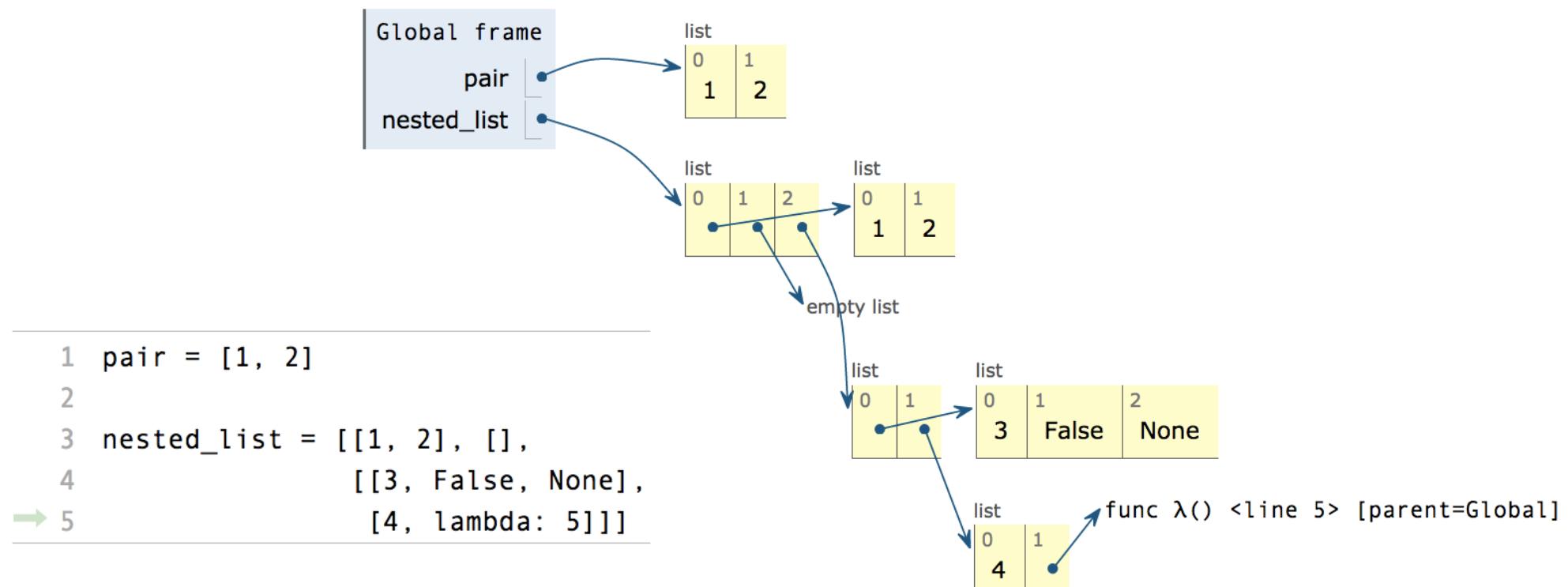
>>> promoted(range(10), odd) # odds in front
[1, 3, 5, 7, 9, 0, 2, 4, 6, 8]
"""
return [e for e in s if f(e)] + [e for e in s if not f(e)]
```

Box-and-Pointer Notation

Box-and-Pointer Notation in Environment Diagrams

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



Discussion Question

What's the environment diagram? What gets printed?

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

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

Slicing

Double-Eights with a List

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

```
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 i in range(len(s)-1):
        if s[i] == 8 and s[i+1] == 8:
            return True
    return False
```

```
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 s[:2] == [8, 8]:
        return True
    elif len(s) < 2:
        return False
    else:
        return double_eights(s[1:])
```

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.

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

With a single iterable argument, return its largest item.
With two or more arguments, return the largest argument.

- **all(iterable)** → bool

Return True if `bool(x)` is True for all values x in the iterable.
If the iterable is empty, return True.

Spring 2023 Midterm 2 Question

Definition. A *prefix sum* of a sequence of numbers is the sum of the first n elements for some positive length n .

(a) (4.0 points)

Implement `prefix`, which takes a list of numbers `s` and returns a list of the prefix sums of `s` in increasing order of the length of the prefix.

```
def prefix(s):
    """Return a list of all prefix sums of list s.

>>> prefix([1, 2, 3, 0, 4, 5])
[1, 3, 6, 6, 10, 15]
>>> prefix([2, 2, 2, 0, -5, 5])
[2, 4, 6, 6, 1, 6]
    """      sum(s[:k+1])      range(len(s))
return [_____ for k in _____]
                (a)                  (b)
```

ii. (1.0 pt) Fill in blank (b).

- `s`
- `[s]`
- `s[1:]`
- `range(s)`
- `range(len(s))`

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 an expression that evaluates to the x for which the corresponding y is smallest:

```
>>> 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]
```

```
>>> x_corresponding_to_min_y
```

```
1
```

Strings

'Demo'

Parking

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 all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n . Some or all 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

{'Dem': 0}

Dictionary Comprehensions

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

Short version: {<key exp>: <value exp> for <name> in <iter exp>}