Containers

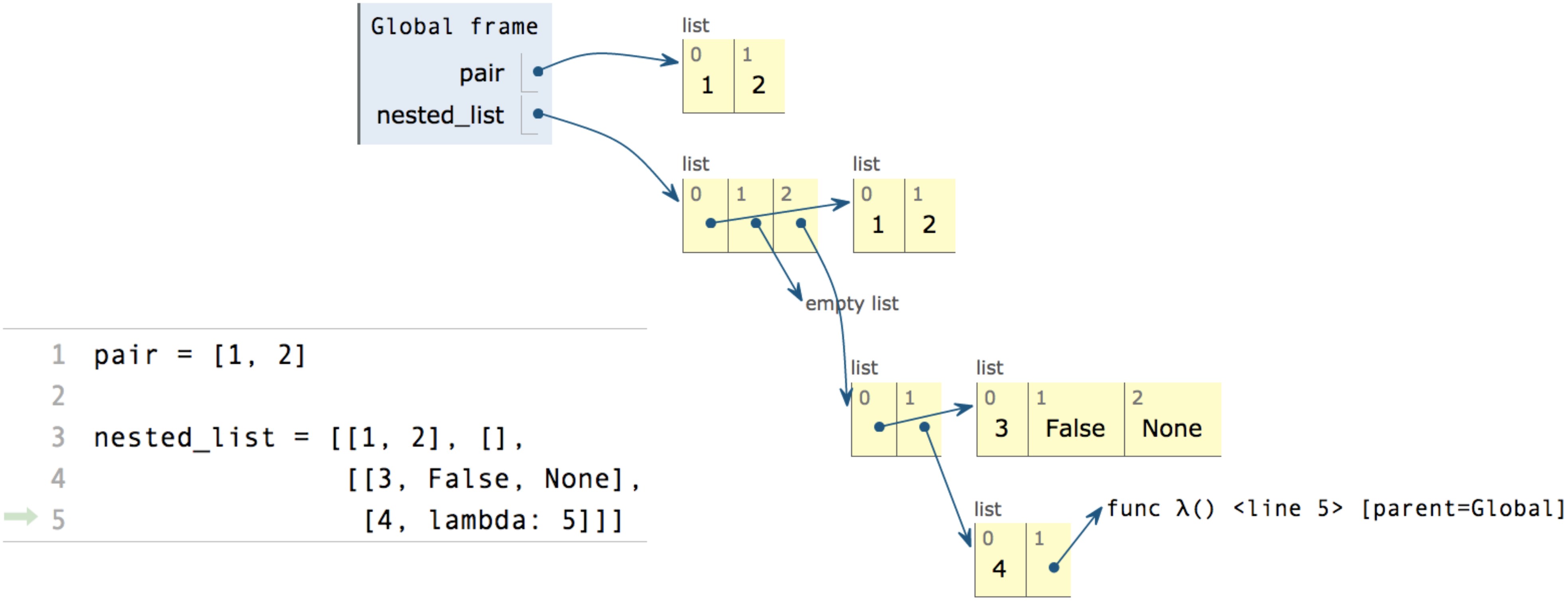
Announcements

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



```
pair = [1, 2]

nested_list = [[1, 2], [],
               [[3, False, None],
                [4, lambda: 5]]]
```

pythontutor.com/composingprograms.html#code=pair%20%3D%20[1,%202]%0A%0Anested_list%20%3D%20[[1,%202],%20[],%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[[3,%20False,%20None],
%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20[4,%20lambda%3A%205]]]&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=4

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)
```

https://pythontutor.com/cp/composingprograms.html#code=def%20f%28s%29%3A%0A%20%20%20%20x%20%3D%20s%5B0%5D%0A%20%20%20%20return%20%5Bx%5D%0A%0At%20%3D%20%5B3,%20%5B2%2B2,%205%5D%5D%0Au%20%3D%20%5Bf%28t%5B1%5D%29,%20t%5D%0Aprint%28u%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Slicing

(Demo)

Double-Eights with a List

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

using positions (indices)...

```
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
```

using slices...

```
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:])
```

https://pythontutor.com/cp/composingprograms.html#code=def%20double_eights%28s%29%3A%0A%20%20%20%20if%20s%5B%3A2%5D%20%3D%3D%20%5B8,%208%5D%3A%0A%20%20%20%20%20%20%20%20return%20True%0A%20%20%20%20elif%20len%28s%29%20%3C%202%3A%0A%20%20%20%20%20%20%20%20return%20False%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20return%20double_eights%28s%5B1%3A%5D%29%0A%20%20%20%20%20%20%20%20%0Adouble_eights%28%5B5,%203,%208,%208,%203,%205%5D%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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

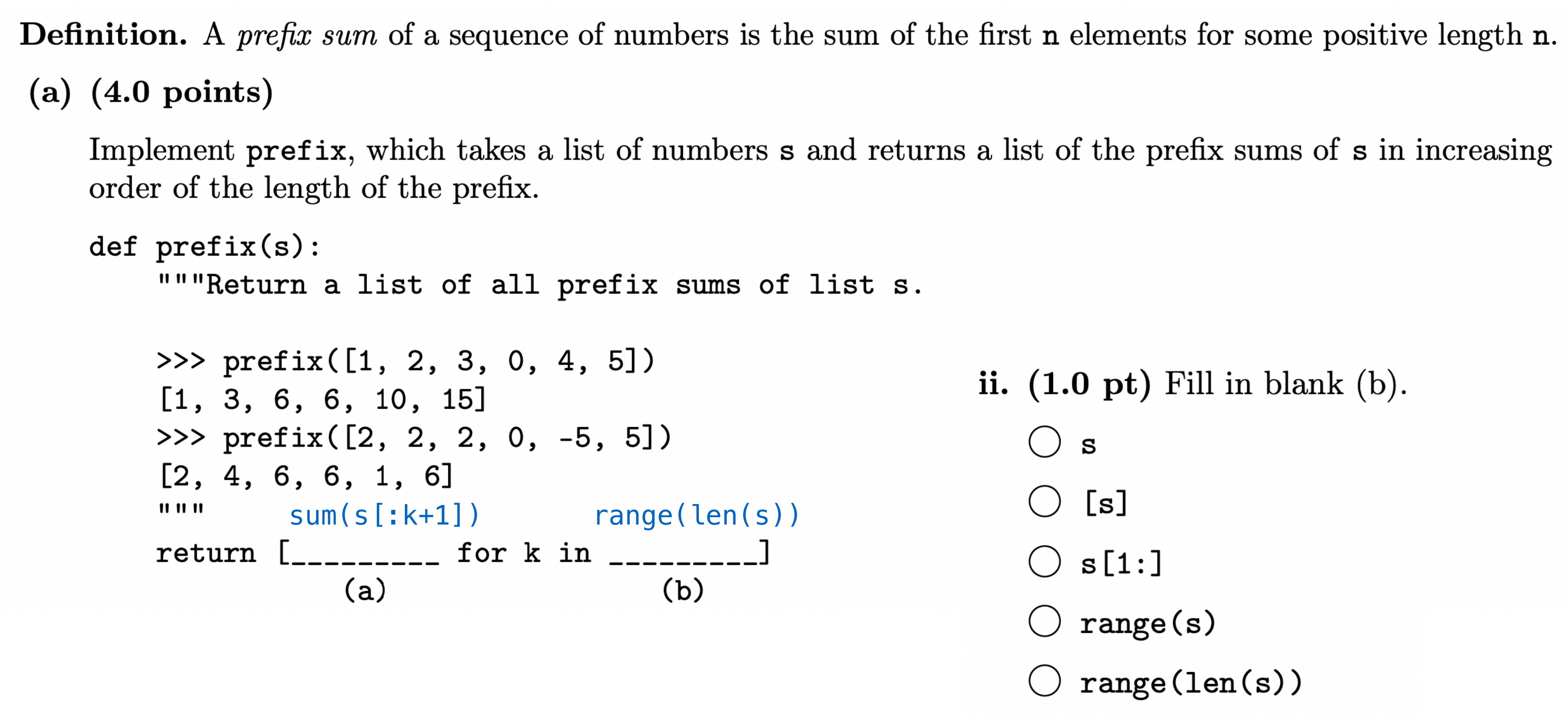
  With a single iterable argument, return its largest item.
  With two or more arguments, return the largest argument.

- **all**(iterable) -> bool

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

(Demo)

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))

Strings

```
'Demo'
```

Tree Recursion (with Strings)

Spring 2023 Midterm 2 Question 5(a) [modified a bit]

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.

```
def count_park(n):
    """Count the ways to park cars and motorcycles in n adjacent spots.
    >>> count_park(1)  # '.' or '%'
    2
    >>> count_park(2)  # '..', '.%', '%.', '%%', or '<>'
    5
    >>> count_park(4)  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
    if n < 0:
        return ______0______
    elif n == 0:
        return ______1______
    else:
        return count_park(n-2) + count_park(n-1) + count_park(n-1)
```

Spring 2023 Midterm 2 Question 5(b) [modified a lot]

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 a list of all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n. 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 ['%'+s for s in park(n-1)] + ['.'+s for s in park(n-1)] + ['<>'+s for s in park(n-2)]
```

```
park(3):
  %%%
  %%.
  %.%
  %..
  %<>
  .%%
  .%.
  ..%
  ...
  .<>
  <>%
  <>.
```