Review 7

Coroutines
Coroutines on the Exam

• We may ask you to **code a coroutine**
  ▪ This is *potentially* the easier question
  ▪ Need to know how to read specification
  ▪ Could be animation or something else

• **Way may ask you a call frames question**
  ▪ This is not *that* hard, actually
  ▪ Behaves like normal function 90% of time
  ▪ The hardest part is the **first step**
def animate(ball, dx, speed):
    """Coroutine to animate a ball
    Each call to send provides a time offset dt (float >= 0). Provided
    the sum of dt values is < speed, this moves the ball dx*dt/speed
    horizontally. If the sum is > speed, it snaps the ball to dx.
    Parameter ball: The ball object to move
    Precondition: ball is an instance of GImage
    Parameter dx: The amount to move the ball
    Precondition: dx is a number (int or float)
    Parameter speed: The number of seconds to animate the ball
    Precondition: speed is a number > 0"

pass
def animate(ball, dx, speed):
    """Coroutine to animate a ball

    Each call to send provides a time offset dt (float >= 0). Provided the sum of dt values is < speed, this moves the ball dx*dt/speed horizontally. If the sum is > speed, it snaps the ball to dx.

    Parameter ball: The ball object to move
    Precondition: ball is an instance of GImage

    Parameter dx: The amount to move the ball
    Precondition: dx is a number (int or float)

    Parameter speed: The number of seconds to animate the ball
    Precondition: speed is a number > 0"
"
    pass
def animate(ball, dx, speed):
    """Coroutine to animate a ball"""
    # Compute number of steps per second (dx/speed)
    # Create aggregator to track total time

    # Until we reach maximum time (speed)
    # Get the number of seconds passed
    # Add it to the total number of seconds
    # Use time to compute the correct position
    # Update the ball position

    # Snap the ball into place
def animate(ball, dx, speed):
    """Coroutine to animate a ball""
    # Compute number of steps per second (dx/speed)
    # Create aggregator to track total time

    # Until we reach maximum time (speed)
    # Get the number of seconds passed
    # Add it to the total number of seconds
    # Use time to compute the correct position
    # Update the ball position

    # Snap the ball into place

Requires a yield expr

Why??
def animate(ball,dx,speed):
    """Coroutine to animate a ball""
    step = dx/speed
    total = 0
    final = ball.x+dx

    while total < speed:
        dt = (yield)
        total = total + dt
        move = dt*step
        ball.x = ball.x + move

    ball.x = final

This is doable on an exam
def chunkify(input):
    """Coroutine to break a list into chunks.

    Each call to send is the number of elements to chunk. At each call, it yields a new list of the size of the number of elements in send.

    If the size (which is an int) sent is <= 0, it yields the empty list

    Parameter input: The data to process
    Precondition: input is a list"

    pass
def chunkify(input):
    """Coroutine to break a list into chunks. Each call to send is the number of elements to chunk. At each call, it yields a new list of the size of the number of elements in send. If the size (which is an int) sent is <= 0, it yields the empty list."
    Parameter input: The data to process
    Precondition: input is a list"

pass
def chunkify(input):
    """Coroutine to break a list into chunks.
    Each call to send is the number of elements to chunk. At each call, it yields a new list of the size of the number of elements in send.
    If the size (which is an int) sent is <= 0, it yields the empty list
    Parameter input: The data to process
    Precondition: input is a list"

    pass
def chunkify(input):
    
    '''Coroutine to break a list into chunks.'''
    
    # Get size of first chunk
    # Create an accumulator for chunks (output lists)
    
    # for each item in input:
        # check if number of items needed is 0
            # output the accumulator have so far
        # get the size of the next chunk
        # reset the accumulator to empty list
        # add item to accumulator
def chunkify(input):
    """Coroutine to break a list into chunks."
    # Get size of first chunk
    # Create an accumulator for chunks
    # for each item in input:
    # check if number of items needed is 0
    # output the accumulator have so far
    # get the size of the next chunk
    # reset the accumulator to empty list
    # add item to accumulator

Initial input

These are done by same yield expr
def chunkify(input):
    """Coroutine to break a list into chunks."""
    size = (yield)
    result = []
    for item in input:
        if size == 0:
            size = (yield result)
            result = []
            result.append(item)
        size = size - 1

Initial input

Output result

Input the size
Writing a Coroutine with Both Ways

```
def chunkify(input):
    """Coroutine to break a list into chunks."""
    size = (yield)
    result = []

    for item in input:
        if size == 0:
            size = (yield result)
            result = []
        result.append(item)
        size = size - 1
```
def chunkify(input):
    """Coroutine to break a list into chunks."""
    size = (yield)
    result = []

    for item in input:
        if size == 0:
            size = (yield result)
            result = []
            result.append(item)
        size = size - 1

    yield result

    Need one last output
Writing a Coroutine with Both Ways

def chunkify(input):
    """Coroutine to break a list into chunks.""
    size = (yield)
    result = []

    for item in input:
        if size == 0:
            size = (yield result)
            result = []
            result.append(item)
            size = size - 1

        yield result
Writing a Coroutine with Both Ways

def chunkify(input):
    """Coroutine to break a list into chunks."""
    size = (yield)
    result = []

    for item in input:
        if size == 0:
            size = (yield result)
            result = []
            result.append(item)
            size = size - 1

    yield result

What if size is 0?
def chunkify(input):
    '''Coroutine to break a list into chunks.'''
    size = (yield)
    result = []

    for item in input:
        while size == 0:
            size = (yield result)
            result = []
            result.append(item)
            size = size - 1

    yield result
def chunkify(input):
    """Coroutine to break a list into chunks."""
    size = (yield)
    result = []
    for item in input:
        while size == 0:
            size = (yield result)
            result = []
            result.append(item)
            size = size - 1
    yield result

Pseudocode makes this much easier
Coroutines and Call Frames

• Recall a generator has **three** steps
  ▪ Initial creation of coroutine (like constructor)
  ▪ Initial call to **next** to get started
  ▪ Subsequent call to **send** to keep going
• Cannot ask you a question about first!
• The second is just like a generator!
  ▪ Stops at the first yield expression/statement
• Only the third one is actually new
  ▪ And only challenge is input vs. output
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
Pre: nums is a list of ints""

    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Code to Execute

```python
23 a = [1, 2, 3]
24 b = addit(a)
25 next(b)
26 c = b.send(2)
```
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    '''Send added to next item
    Pre: nums is a list of ints'''
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Given at Line 24

```python
23  a = [1,2,3]
24  b = addit(a)
25  next(b)
26  c = b.send(2)
```
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Given at Line 24

```python
23 a = [1,2,3]
24 b = addit(a)
25 next(b)
26 c = b.send(2)
```

Just like a generator

Diagram Line 25
def addit(nums):
    
    """Send added to next item
    Pre: nums is a list of ints"
    
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Diagram Step 2

Will cheat and not show frame

```
12/3/20
Generators
```
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints""
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Diagram Step 3

```
1 2
id1
addit(a)

list
0 1 2 3
id2
corot

17
id1
addit

nums id1 pos 0
```
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Diagram Step 4

Return None is optional
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos]=nums[pos]+add
        pos = pos + 1
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1

Given at Line 25

23  a = [1,2,3]
24  b = addit(a)
25  next(b)
26  c = b.send(2)
```

Not much has changed

12/3/20

Generators
Coroutine with Input Only

def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1

Given at Line 24

23  a = [1,2,3]
24  b = addit(a)
25  next(b)
26  c = b.send(2)

Diagram
Line 26
Coroutine with Input Only

**Coroutine**

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints""
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
```

**Initial Diagram**

- Input goes to `add`
def addit(nums):
    '''Send added to next item
    Pre: nums is a list of ints'''
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
**Coroutine with Input Only**

**Coroutine**

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

**Diagram Step 4**

[Diagram showing the coroutine flow with indices and values]
Coroutine with Input Only

Coroutine

```python
def addit(nums):
    """Send added to next item
    Pre: nums is a list of ints"
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos] = nums[pos] + add
        pos = pos + 1
```

Diagram Step 5
Coroutine with Input Only

Coroutine

def addit(nums):
    
    """Send added to next item
    Pre: nums is a list of ints"
    
    pos = 0
    while pos < len(nums):
        add = (yield)
        nums[pos]=nums[pos]+add
        pos = pos + 1

Erase the Frame

12/3/20
Generators
36
Coroutine with Input and Output

Coroutine

```python
def skipit(nums):
    """Send tells positions to skip
    Pre: nums is a list of ints""

    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]
```

Code to Execute

```python
a = [9, 7, 4, 2]
b = skipit(a)
c = next(b)
d = b.send(2)
e = b.send(3)
```
Coroutine with Input and Output

Coroutine

```python
def skipit(nums):
    """Send tells positions to skip
    Pre: nums is a list of ints""
    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]
```

Given at Line 25

```python
22 a = [9, 7, 4, 2]
23 b = skipit(a)
24 c = next(b)
25 d = b.send(2)
26 e = b.send(3)
```

Generators
Coroutine with Input and Output

**Coroutine**

```python
def skipit(nums):
    """Send tells positions to skip
    Pre: nums is a list of ints""
    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]
```

**Given at Line 25**

```python
22  a = [9,7,4,2]
23  b = skipit(a)
24  c = next(b)
25  d = b.send(2)
26  e = b.send(3)
```

**Diagram Line 26**
Coroutine with Input and Output

Coroutine

def skipit(nums):
    '''Send tells positions to skip
    Pre: nums is a list of ints'''
    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]

Initial Diagram
Coroutine with Input and Output

Coroutine

```python
def skipit(nums):
    """Send tells positions to skip
    Pre: nums is a list of ints""
    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]
```

Diagram Step 2
Coroutine with Input and Output

Coroutine

```python
def skipit(nums):
    """Send tells positions to skip
    Pre: nums is a list of ints"""
    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]
```

Diagram Step 3

```
12/3/20

Generators
42
```
Coroutine with Input and Output

Coroutine

def skipit(nums):
    
    
    
    
    
    """Send tells positions to skip 
    Pre: nums is a list of ints""

    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]

Diagram Step 4

corot

skipit(a)

list

nums id1

pos 5

RETURN 2 skip 3
Coroutine with Input and Output

Coroutine

def skipit(nums):
    """Send tells positions to skip
    Pre: nums is a list of ints""
    pos = 0
    while pos < len(nums):
        skip = (yield nums[pos])
        pos = pos + skip
    yield nums[-1]

Erase the Frame
Coroutines and Functions

---

### Parent Function

```python
def sumfold(lst):
    """Returns list of sums""
    sum = []
g = pushsum(len(lst))
next(g)
for x in lst:
a = g.send(x)
sum.append(a)
return sum
```

### Coroutine

```python
def pushsum(n):
    """Yields sum of all sent""
    sum = 0
for x in range(n):
    val = (yield sum)
    sum = sum + val
yield sum
```
Coroutines and Functions

Parent Function

```python
def sumfold(lst):
    """Returns list of sums"""
    sum = []
g = pushsum(len(lst))
next(g)
for x in lst:
a = g.send(x)
    sum.append(a)
return sum
```

Function Call

```python
>>> x = sumfold([1,2])
```

Assume we are here:

```mermaid
diagram "Generators"
    sumfold - lst
    id3 - g
    id5 - 34
    sum - id4
    id3 - list
    id4 - list
    What is the next step?
```

What is the next step?
Which One is Closest to Your Answer?

A: \[
\text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 35 \\
\text{sum} \quad \text{id4}
\]

B: \[
\text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 34 \\
\text{sum} \quad \text{id4} \\
\text{pushsum} \quad n \quad 2 \quad 16 \\
\text{sum} \quad 0
\]

C: \[
\text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 34 \\
\text{sum} \quad \text{id4} \\
\text{pushsum} \quad n \quad 2 \quad 16
\]

D: \[
\text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 34 \\
\text{sum} \quad \text{id4} \\
\text{pushsum} \quad \text{16}
\]
Which One is Closest to Your Answer?

A: \[ \text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 35 \]
\[ \text{sum} \quad \text{id4} \]

B: \[ \text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 34 \]
\[ \text{sum} \quad \text{id4} \]
\[ \text{pushsum} \quad n \quad 2 \quad 16 \]

C: \[ \text{sumfold} \quad \text{lst} \quad \text{id3} \quad g \quad \text{id5} \quad 34 \]
\[ \text{sum} \quad \text{id4} \]
\[ \text{pushsum} \quad n \quad 2 \quad 16 \]

In all cases, the heap is unchanged.
Coroutines and Functions

Coroutine

def pushsum(n):
    """Yields sum of all sent"""
    sum = 0
    for x in range(n):
        val = (yield sum)
        sum = sum + val
    yield sum

Function Call

>>> x = sumfold([1,2])

What is the next step?

C: sumfold lst id3 g id5 34
   sum id4
   pushsum n 2 16
Which One is Closest to Your Answer?

A: \[
\text{sumfold} \hspace{5pt} \text{lst} \hspace{5pt} \text{id3} \hspace{5pt} \text{g} \hspace{5pt} \text{id5} \hspace{5pt} 35 \\
\text{sum} \hspace{5pt} \text{id4} \\
\text{pushsum} \hspace{5pt} n \hspace{5pt} 2 \hspace{5pt} 17 \\
\text{sum} \hspace{5pt} 0
\]

B: \[
\text{sumfold} \hspace{5pt} \text{lst} \hspace{5pt} \text{id3} \hspace{5pt} \text{g} \hspace{5pt} \text{id5} \hspace{5pt} 34 \\
\text{sum} \hspace{5pt} \text{id4} \\
\text{pushsum} \hspace{5pt} n \hspace{5pt} 2 \hspace{5pt} x \hspace{5pt} 1 \hspace{5pt} 17 \\
\text{sum} \hspace{5pt} 0
\]

C: \[
\text{sumfold} \hspace{5pt} \text{lst} \hspace{5pt} \text{id3} \hspace{5pt} \text{g} \hspace{5pt} \text{id5} \hspace{5pt} 34 \\
\text{sum} \hspace{5pt} \text{id4} \\
\text{pushsum} \hspace{5pt} n \hspace{5pt} 2 \hspace{5pt} x \hspace{5pt} 0 \hspace{5pt} 17 \\
\text{sum} \hspace{5pt} 0
\]

D: \[
\text{sumfold} \hspace{5pt} \text{lst} \hspace{5pt} \text{id3} \hspace{5pt} \text{g} \hspace{5pt} \text{id5} \hspace{5pt} 34 \\
\text{sum} \hspace{5pt} \text{id4} \\
\text{pushsum} \hspace{5pt} n \hspace{5pt} 2 \hspace{5pt} x \hspace{5pt} 0 \hspace{5pt} 18 \\
\text{sum} \hspace{5pt} 0
\]
Which One is Closest to Your Answer?

In all cases, the heap is unchanged

A: 
- `sumfold` lst `id3` g `id5` 35
- `sum` id4
- `pushsum` n 2 17
- `sum` 0

B: 
- `sumfold` lst `id3` g `id5` 34
- `sum` id4
- `pushsum` n 2 x 1 17
- `sum` 0

C: 
- `sumfold` lst `id3` g `id5` 34
- `sum` id4
- `pushsum` n 2 x 0 17
- `sum` 0

D: 
- `sumfold` lst `id3` g `id5` 34
- `sum` id4
- `pushsum` n 2 x 0 18
- `sum` 0

12/3/20

Generators
Coroutines and Functions

Coroutine

def pushsum(n):
    """Yields sum of all sent""
    sum = 0
    for x in range(n):
        val = (yield sum)
        sum = sum + val
    yield sum

Function Call

>>> x = sumfold([1,2])

What is the next step?

A: sumfold  lst  id3  g  id5  35
    sum  id4
    pushsum  n  2  17
    sum  0
Which One is Closest to Your Answer?

A:

\[
\begin{array}{c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id} & \text{g} & \text{id} \\
\hline
\text{sum} & \text{id} & 35 \\
\hline
\text{pushsum} & \text{n} & 2 & x & 0 \\
\hline
\text{sum} & 0 \\
\end{array}
\]

B:

\[
\begin{array}{c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id} & \text{g} & \text{id} \\
\hline
\text{sum} & \text{id} & 34 \\
\hline
\text{pushsum} & \text{n} & 2 & x & 1 \\
\hline
\text{sum} & 0 \\
\end{array}
\]

C:

\[
\begin{array}{c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id} & \text{g} & \text{id} \\
\hline
\text{sum} & \text{id} & 34 \\
\hline
\text{pushsum} & \text{n} & 2 & x & 0 \\
\hline
\text{sum} & 1 \\
\end{array}
\]

D:

\[
\begin{array}{c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id} & \text{g} & \text{id} \\
\hline
\text{sum} & \text{id} & 34 \\
\hline
\text{pushsum} & \text{n} & 2 & x & 1 \\
\hline
\text{sum} & 1 \\
\end{array}
\]
Coroutines and Functions

Coroutine

```python
def pushsum(n):
    """Yields sum of all sent"""
    sum = 0
    for x in range(n):
        val = (yield sum)
        sum = sum + val
    yield sum
```

Function Call

```python
>>> x = sumfold([1,2])
```

What is the next step?

A:

<table>
<thead>
<tr>
<th>sumfold</th>
<th>lst</th>
<th>id3</th>
<th>g</th>
<th>id5</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>id4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushsum</td>
<td>n</td>
<td>2</td>
<td>x</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>sum</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12/3/20
Which One is Closest to Your Answer?

<table>
<thead>
<tr>
<th>A:</th>
<th>sumfold</th>
<th>lst</th>
<th>id3</th>
<th>g</th>
<th>id5</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td></td>
<td></td>
<td>id4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pushsum</td>
<td>n</td>
<td>2</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td>0</td>
<td>YIELD</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B:</th>
<th>sumfold</th>
<th>lst</th>
<th>id3</th>
<th>g</th>
<th>id5</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td></td>
<td></td>
<td>id4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pushsum</td>
<td>n</td>
<td>2</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td>0</td>
<td>YIELD</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C:</th>
<th>sumfold</th>
<th>lst</th>
<th>id3</th>
<th>g</th>
<th>id5</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td></td>
<td></td>
<td>id4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pushsum</td>
<td>n</td>
<td>2</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td>0</td>
<td>RETURN</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D:</th>
<th>sumfold</th>
<th>lst</th>
<th>id3</th>
<th>g</th>
<th>id5</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td></td>
<td></td>
<td>id4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pushsum</td>
<td>n</td>
<td>2</td>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td></td>
<td>0</td>
<td>RETURN</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
def sumfold(lst):
    """Returns list of sums""
    sum = []
    g = pushsum(len(lst))
    next(g)
    for x in lst:
        a = g.send(x)
        sum.append(a)
    return sum

>>> x = sumfold([1,2])
Coroutines and Functions

Parent Function

def sumfold(lst):
    """Returns list of sums"""
    sum = []
    g = pushsum(len(lst))
    next(g)
    for x in lst:
        a = g.send(x)
        sum.append(a)
    return sum

Function Call

>>> x = sumfold([1,2])

What is the next step?

Generators
Which One is Closest to Your Answer?

A:
\[
\begin{array}{c|cc|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} \\
\text{sum} & \text{id4} & \text{x} & 0 \\
\end{array}
\]

B:
\[
\begin{array}{c|cc|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} \\
\text{sum} & \text{id4} & \text{x} & 0 \\
\end{array}
\]

C:
\[
\begin{array}{c|cc|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} \\
\text{sum} & \text{id4} & \text{x} & 1 \\
\end{array}
\]

D:
\[
\begin{array}{c|cc|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} \\
\text{sum} & \text{id4} & \text{x} & 1 \\
\end{array}
\]
Coroutines and Functions

Parent Function

```python
def sumfold(lst):
    """Returns list of sums"""
    sum = []
    g = pushsum(len(lst))
    next(g)
    for x in lst:
        a = g.send(x)
        sum.append(a)
    return sum
```

Function Call

```python
>>> x = sumfold([1,2])
```

What is the next step?
Which One is Closest to Your Answer?

A:

\[
\begin{array}{c|c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} & \text{id5} & 37 \\
\hline
\text{sum} & \text{id4} & x & 1 & a & 1 \\
\end{array}
\]

B:

\[
\begin{array}{c|c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} & \text{id5} & 36 \\
\hline
\text{sum} & \text{id4} & x & 1 & & \\
\text{pushsum} & n & 2 & & 16 \\
\end{array}
\]

C:

\[
\begin{array}{c|c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} & \text{id5} & 36 \\
\hline
\text{sum} & \text{id4} & x & 1 & & \\
\text{pushsum} & n & 2 & x & 0 & 18 \\
\text{sum} & 0 & \text{RETURN} & 0 & & \\
\end{array}
\]

D:

\[
\begin{array}{c|c|c|c|c|c}
\text{sumfold} & \text{lst} & \text{id3} & \text{g} & \text{id5} & 36 \\
\hline
\text{sum} & \text{id4} & x & 1 & & \\
\text{pushsum} & n & 2 & x & 0 & 19 \\
\text{sum} & 0 & & \text{val} & 1 & \\
\end{array}
\]
Coroutines and Functions

Coroutine

```python
def pushsum(n):
    """Yields sum of all sent"""
    sum = 0
    for x in range(n):
        val = (yield sum)
        sum = sum + val
    yield sum
```

Function Call

```python
>>> x = sumfold([1, 2])
```

```
D:

<table>
<thead>
<tr>
<th>sumfold</th>
<th>lst</th>
<th>id3</th>
<th>g</th>
<th>id5</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>id4</td>
<td>x</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushsum</td>
<td>n 2</td>
<td>x 0</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>0</td>
<td>val</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Try the rest on your own

12/3/20

Generators
Questions?