Generators
Generators on the Exam

- We may ask you to **code a generator**
  - This is actually the easier question
  - Just need to know how to read specification
  - Similar to a traditional for-loop question
- 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 emit_alpha(string):
    """
    Generates the letters in string, in the order given

    This generator only outputs one letter at a time.

    Example: emit_alpha('ab12c!') yields 'a', 'b', and 'c', in that order

    Parameter string: The string to process
    Precondition: string is a str
    """
    pass
def emit_alpha(string):
    """
    Generates the letters in string, in the order given
    This generator only outputs one letter at a time.
    Example: emit_alpha('ab12c!') yields 'a', 'b', and 'c', in that order
    Parameter string: The string to process
    Precondition: string is a str
    """
    pass
def emit_alpha(string):
    """Generates the letters in string, in the order given
    Precondition: string is a str"""
    # for each element of the string
    # check if the element is a letter
    # output (yield) it if so
Implementing the Generator

def emit_alpha(string):
    """Generates the letters in string, in the order given
    Precondition: string is a str"""

    for x in string:
        if x.isalpha():
            yield x
def sumfold(input):
    
    """
    Generates the sums of the numbers seen so far in input
    
    Example: sumfold([1,2,3]) yields the numbers 1, 3, and 6
    
    Parameter input: The input data to sum
    Precondition: input is a iterable of numbers (int or float)
    """

    pass
def sumfold(input):

    """
    Generates the sums of the numbers seen so far in input
    
    Example: sumfold([1,2,3]) yields the numbers 1, 3, and 6
    
    Parameter input: The input data to sum
    
    Precondition: input is a iterable of numbers (int or float)
    """

    pass

This is not a sequence!
Not sliceable! No len()!
Can only use loops!
def sumfold(input):
    """Generates the sums of the numbers seen so far in input
    Precondition: input is a iterable of numbers (int or float)""
    
    # for item in input
    # output (yield) sum of data so far
def sumfold(input):
    """Generates the sums of the numbers seen so far in input
    Precondition: input is a iterable of numbers (int or float)"""

    # for item in input
    # add item to sum of data so far
    # output (yield) sum
def sumfold(input):
    """Generates the sums of the numbers seen so far in input
    Precondition: input is a iterable of numbers (int or float)""
    # create variable for sum so far

    # for item in input
        # add item to sum of data so far
        # output (yield) sum
def sumfold(input):
    """Generates the sums of the numbers seen so far in input
    Precondition: input is a sequence of numbers (int or float)"""
    sum = 0
    for item in input:
        sum = sum + item
        yield sum
def filterdiv(input, n):
    """Generates all elements of input evenly divisible by n
    The elements are generated in the order they appear in input.
    Example: filterdiv([1,2,3,4],2) generates the numbers 2 and 4
    Parameter input: The input data to filter
    Precondition: input an iterable of int
    Parameter n: The number to divide by
    Precondition: n an int > 0""
    pass
def filterdiv(input, n):
    """Generates all elements of input evenly divisible by n
    The elements are generated in the order they appear in input.
    Example: filterdiv([1,2,3,4],2) generates the numbers 2 and 4
    Parameter input: The input data to filter
    Precondition: input an iterable of int
    Parameter n: The number to divide by
    Precondition: n an int > 0"""
    pass
More than One Parameter

```python
def filterdiv(input, n):
    """Generates all elements of input evenly divisible by n
    Precondition: input an iterable of int
    Precondition: n an int > 0"""
    # for each item in input
    # check if item is divisible by n
    # output (yield) it if so
```

12/3/20
Generators 15
def filterdiv(input, n):
    """Generates all elements of input evenly divisible by n
    Precondition: input an iterable of int
    Precondition: n an int > 0"""

    for item in input:
        if item % n == 0:
            yield item
def pair_swap(input):
    
    Generates output consisting of input, all adjacent pairs swapped

Example: pair_swap((1,2,3,4,5)) yields 2, 1, 4, 3, and 5, in that order.

Parameter input: The input to process
Precondition: input is an iterable or iterator

pass
The Optional Lab Problem

```python
def pair_swap(input):
    """
    Generates output consisting of input, all adjacent pairs swapped
    Example: pair_swap((1, 2, 3, 4, 5)) yields 2, 1, 4, 3, and 5, in that order.
    Parameter input: The input to process
    Precondition: input is an iterable or iterator
    """
    pass
```

This is not a sequence! Not sliceable! No len()! Can only use loops!
**The Optional Lab Problem**

```python
def pair_swap(input):
    """Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator"
    
    # for each two elements (a,b) of input
    # yield b
    # yield a
```

How to this this? Only get one at time
The Optional Lab Problem

```python
def pair_swap(input):
    """Generates outputing contests of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator""

    # for each element a in input
    # check if a is SECOND item
    # yield a
    # yield first item
```

12/3/20
Generators 20
def pair_swap(input):
    """Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator"""
    # create variable for first item

    # for each element a in input
    # check if first item is not None
    # yield a
    # yield first item
    # set first item to None
    # else assign a to first item
The Optional Lab Problem

def pair_swap(input):
    """Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator"
    first = None

    for a in input:
        if not first is None:
            yield a
            yield first
            first = None
        else:
            first = a

Are we done?
def pair_swap(input):
    """Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator""
    first = None

    for a in input:
        if not first is None:
            yield a
            yield first
            first = None
        else:
            first = a
        if not first is None:
            yield first

    Need one last output if odd
The Optional Lab Problem

```python
def pair_swap(input):
    """Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator"
    first = None

    for a in input:
        if not first is None:
            yield a
            yield first
            first = None
        else:
            first = a
        if not first is None:
            yield first
```

Are we done?
def pair_swap(input):
    '''Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator'''
    first = None
    for a in input:
        if not first is None:
            yield a
            yield first
            first = None
        else:
            first = a
        if not first is None:
            yield first

What if input contains None?

Var first does two things:
* remembers prev value
* checks if even position
The Optional Lab Problem

```python
def pair_swap(input):
    """Generates output consisting of input, all adjacent pairs swapped
    Precondition: input is an iterable or iterator""
    first = None
    issec = False
    for a in input:
        if issec:
            yield a
            yield first
            first = None; issec = False
        else:
            first = a; issec = True
        if issec:
            yield first

12/3/20 Generators 26
```
Generators and Call Frames

• Recall a generator has two steps
  ▪ Initial creation of generator (like constructor)
  ▪ Subsequent calls to function next

• Cannot ask you a question about first!
  ▪ You don’t know how that function works
  ▪ You do not know contents of generator folder

• Can only ask you about next
  ▪ But this is like a normal function call
  ▪ The only hard part is the start of the call
Generator Call Frames: At the Start

Generator and Code

```python
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_even([1,2,3])
b = next(a)
```

Given After Line 26

```
a = emit_even([1,2,3])
```

Do not know the contents

Where is this remembered?

12/3/20 Generators
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_even([1,2,3])
b = next(a)
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)

This is all given!
Will never be asked to draw this step!
Generator Call Frames: At the Start

Generator and Code

```python
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints""
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)
```

Given After Line 26

Diagram

Line 27
Generator Call Frames: At the Start

Generator and Code

```python
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1, 2, 3])
b = next(a)
```

Initial Diagram

Generator unchanged

```
0 1
1 2
2 3
```

---

12/3/20

Generators
def emit_even(input):
    
    """Gens all even #s in input
    Prec: input list of ints"""
    
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)
Generator Call Frames: At the Start

Generator and Code

def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"""
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)

Diagram Step 3

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Generator unchanged
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"""
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)
def emit_even(input):
    
    """Gens all even #s in input
    Prec: input list of ints"
    
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints""
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1, 2, 3])
b = next(a)
**Generator Call Frames: At the Start**

---

**Generator and Code**

```python
def emit_even(input):
    """Gens all even #s in input
    Prec: input list of ints"
    for x in input:
        if x % 2 == 0:
            yield x

# Code to execute
a = emit_evens([1,2,3])
b = next(a)
```

---

**Erase the Frame**

```
\[
\begin{array}{c}
a \quad \text{id1} \\
b \quad \text{id2} \\
0 \quad \text{list} \\
1 \quad 1 \\
2 \quad 2 \\
3 \quad 3 \\
\end{array}
\]
```

```
\[
\begin{array}{c}
\text{emit_even} \\
\text{input} \\
\text{id2} \\
\text{x} \\
\text{RETURN} \\
2 \\
\end{array}
\]
```
def scatter(input):
    """Gens input as 1-elt lists"""
    for x in input:
        item = [x]
        yield item

# Code to execute
a = scatter([1,2,3,4])
b = next(a)
c = next(a)
```python
def scatter(input):
    """Gens input as 1-elt lists"""
    for x in input:
        item = [x]
        yield item

# Code to execute
a = scatter([1,2,3,4])
b = next(a)
c = next(a)
```

Initial Step:

```
scatter
input
id2 x 1
item
id3
```

12/3/20
Generators
def scatter(input):
    """Gens input as 1-elt lists"""
    for x in input:
        item = [x]
        yield item

# Code to execute
a = scatter([1,2,3,4])
b = next(a)
c = next(a)
def scatter(input):
    """Gens input as 1-elt lists"""
    for x in input:
        item = [x]
        yield item

# Code to execute
a = scatter([1,2,3,4])
b = next(a)
c = next(a)
def scatter(input):
    """Gens input as 1-elt lists"""
    for x in input:
        item = [x]
        yield item

# Code to execute
a = scatter([1,2,3,4])
b = next(a)
c = next(a)

Diagram Step 4

12/3/20
Generators
def scatter(input):
    """Gens input as 1-elt lists"""
    for x in input:
        item = [x]
        yield item

# Code to execute
a = scatter([1,2,3,4])
b = next(a)
c = next(a)
Generators and Functions

Function Definitions

def rnginv(n):    #Inverse range
    for x in range(1,n):
        yield 1/x

def harmonic(n):  #Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum+x
    return x

Function Call

>>> x = harmonic(3)

Assume we are here:

Ignoring the heap, what is the next step?
Which One is Closest to Your Answer?

A:  
- harmonic  
- sum 0  
- rnginv

B:  
- harmonic  
- sum 0  
- rnginv

C:  
- harmonic  
- sum 0  
- x  

D:  
- harmonic  
- sum 0  
- rnginv

12/3/20  
Generators
Generators and Functions

Function Definitions

```python
def rnginv(n):   #Inverse range
    for x in range(1,n):
        yield 1/x

def harmonic(n):  #Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum+x
    return x
```

Function Call

```python
>>> x = harmonic(3)
```

A:

<table>
<thead>
<tr>
<th>harmonic</th>
<th>n</th>
<th>3</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>0</td>
<td>g</td>
<td>id3</td>
</tr>
<tr>
<td>rnginv</td>
<td>n</td>
<td>3</td>
<td>19</td>
</tr>
</tbody>
</table>

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

A:

harmonic  n  3  34
sum   0  g  id3  x  1

B:

harmonic  n  3  34
sum   0  g  id3
rnginv  n  2  20
x  1

C:

harmonic  n  3  34
sum   0  g  id3
rnginv  n  3  20
x  1  YIELD  1

D:

harmonic  n  3  34
sum   0  g  id3
rnginv  n  3  21
x  1  YIELD  1
Generators and Functions

Function Definitions

def rnginv(n):     #Inverse range
    for x in range(1,n):
        yield 1/x

def harmonic(n):  #Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum+x
    return x

Function Call

>>> x = harmonic(3)

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

A:

harmonic  n 3  34
sum 0  g id3 x 1

B:

harmonic  n 3  34
sum 0  g id3 x 1
rnginv  n 3  19
x 1  YIELD 1

C:

harmonic  n 3  34
sum 0  g id3
rnginv  n 3
x 1  YIELD 1

D:

harmonic  n 3  34
sum 0  g id3
rnginv  n 3
x 1  RETURN 1
Generators and Functions

Function Definitions

```python
def rnginv(n):  # Inverse range
    for x in range(1, n):
        yield 1/x

def harmonic(n):  # Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum + x
    return x
```

Function Call

```python
>>> x = harmonic(3)
```

D:

```
harmonic   n  3  34
sum       0  g  id3
rnginv    n  3
x         1  RETURN 1
```

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

A: 
```
harmonic 3 35
sum 0 g id3 x 1
```

B: 
```
harmonic 3 35
sum 1 g id3 x 1
```

C: 
```
rnginv 3 35
sum 0 g id3 x 1
```

D: 
```
rnginv 3 35
sum 1 g id3 x 1
```

12/3/20
Generators
Generators and Functions

---

### Function Definitions

```python
def rnginv(n):    # Inverse range
    for x in range(1, n):
        yield 1/x

def harmonic(n): # Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum + x
    return x
```
### Which One is Closest to Your Answer?

<table>
<thead>
<tr>
<th>A:</th>
<th>harmonic</th>
<th>n</th>
<th>3</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td>g</td>
<td>id3</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B:</th>
<th>harmonic</th>
<th>n</th>
<th>3</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td>g</td>
<td>id3</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C:</th>
<th>harmonic</th>
<th>n</th>
<th>3</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td>g</td>
<td>id3</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>rnginv</td>
<td>n</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D:</th>
<th>harmonic</th>
<th>n</th>
<th>3</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sum</td>
<td>g</td>
<td>id3</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>rnginv</td>
<td>n</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generators and Functions

**Function Definitions**

```python
def rnginv(n):  # Inverse range
    for x in range(1,n):
        yield 1/x

def harmonic(n):  # Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum + x
    return x
```

**Function Call**

```python
>>> x = harmonic(2)
```

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

A: harmonic n 3 34
sum 1 g id3 x 1
rnginv n 3 19

B: harmonic n 3 34
sum 1 g id3 x 1
rnginv n 3 19
x 1

C: harmonic n 3 35
sum 1 g id3 x 0.5

D: harmonic n 3 34
sum 0 g id3 x 1
rnginv n 3 20
x 2

12/3/20
Generators 57
Generators and Functions

Function Definitions

```python
def rnginv(n):
    # Inverse range
    for x in range(1, n):
        yield 1/x

def harmonic(n):
    # Harmonic sum
    sum = 0
    g = rnginv(n)
    for x in g:
        sum = sum + x
    return x
```

Function Call

```python
>>> x = harmonic(2)
```

Try the rest on your own
Questions?