Review 6

# 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**

## Generator Specifications

```
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
```

# Generator Specifications

```
def emit_alp
    """
    Generates the letters in string, in the order given

    This generator only outputs one lette

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

    Parameter string: The string to process
    Precondition: string is a str
    """
    pass
```

Indication it is a generator

Indication of what it outputs and in what order

Precondition for same reason as a function

# Implementing the Generator

```
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
```

# Another Exercise

```
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
```

# Another Exercise

```
def sumfold(input):
    """
    Generates the sums of the numbers seen so far in input

    Example: sumfold([1,2,3                                nd 6

    Parameter input: The input              o 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!

## Another Exercise

```
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
```

## Another Exercise

```
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
```

## Another Exercise

```
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
```

## Another Exercise

```
def sumfold(input):
    """Gene                       umbers seen so far in input
    Precon                        e of numbers (int or float)"""
    sum = 0

    for item in input:
        sum = sum + item
        yield sum
```

Some generators need accumulators

# More than One Parameter

```
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

```
def filterdiv(input,n):
    """Generates all elements of input evenly divisible by n

    The elements are                    r they appear in input.

    Example: filterdiv                  es 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
```

Don’t need an accumulator

# More than One Parameter

```
def filterdiv(input,n):
    """Generates all elements of input evenly divisble 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
```

## More than One Parameter

```
def filterdiv(input,n):
    """Generates all elements of input evenly divisble by n
    Precondition: input an iterable of int
    Precondition: n an int > 0"""

    for item in input:
        if item % n == 0:
            yield item
```

## The Optional Lab Problem

```
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

```
def pair_swap(input):
    """
    Generates output consisting of input, all adjacent pairs swapped

    Example: pair_swap((1,                                    d 5, in
    that order.

    Parameter input: The input         ccess
    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

```
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

```
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
```

# The Optional Lab Problem

```
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?

# 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
    if not first is None:
        yield first
```

Need one last output if odd

## 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
    if not first is None:
        yield first
```

Are we done?

# 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
    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

```
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
```

## Generators and Call Frames

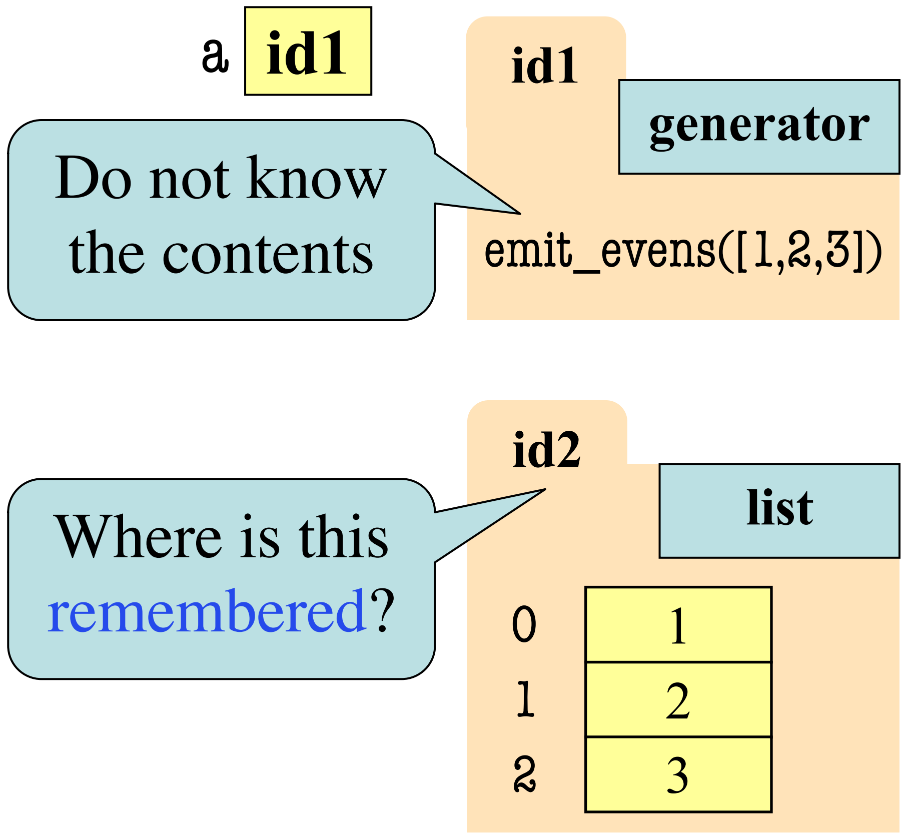
- Recall a generator has two steps
  - Initial creation of generator (like constructor)
  - Subsequent calls to function `next`
- Cannot ask you a question about first!
  - You don't know how that function works
  - You do not know contents of generator folder
- Can only ask you about `next`
  - But 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

```
   def emit_even(input):
       """Gens all even #s in input
          Prec: input list of ints"""
21     for x in input:
22         if x % 2 == 0:
23             yield x
24
25 # Code to execute
26 a = emit_evens([1,2,3])
27 b = next(a)
```

## Given After Line 26

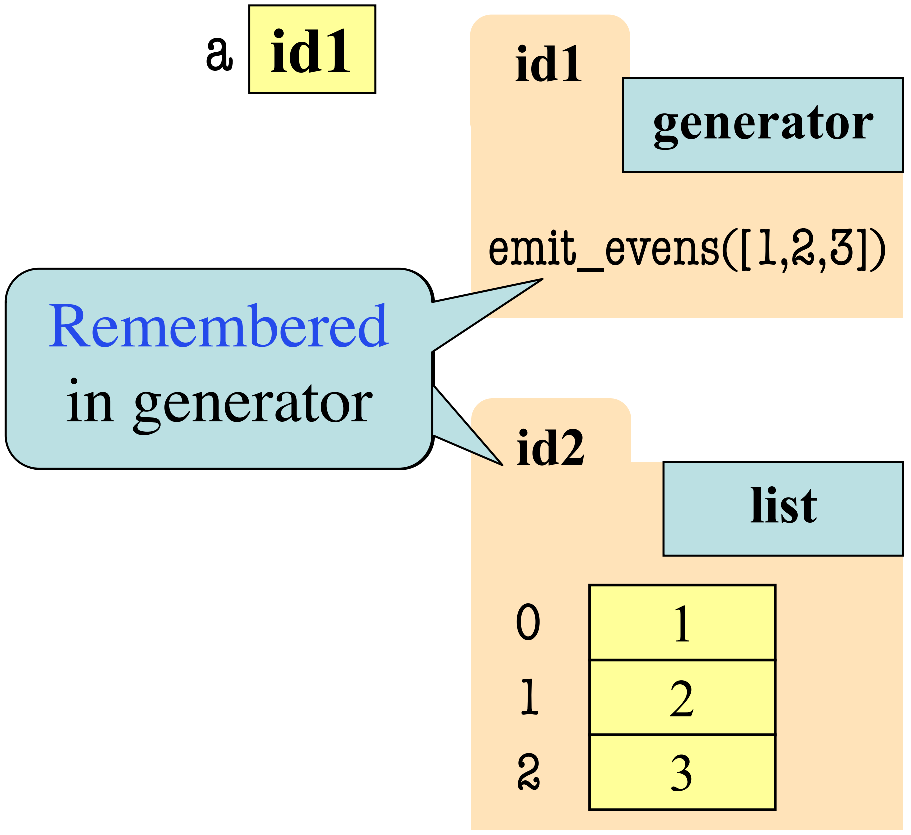


# Generator Call Frames: At the Start

## Generator and Code

```
   def emit_even(input):
       """Gens all even #s in input
          Prec: input list of ints"""
21     for x in input:
22         if x % 2 == 0:
23             yield x
24
25 # Code to execute
26 a = emit_evens([1,2,3])
27 b = next(a)
```

## Given After Line 26

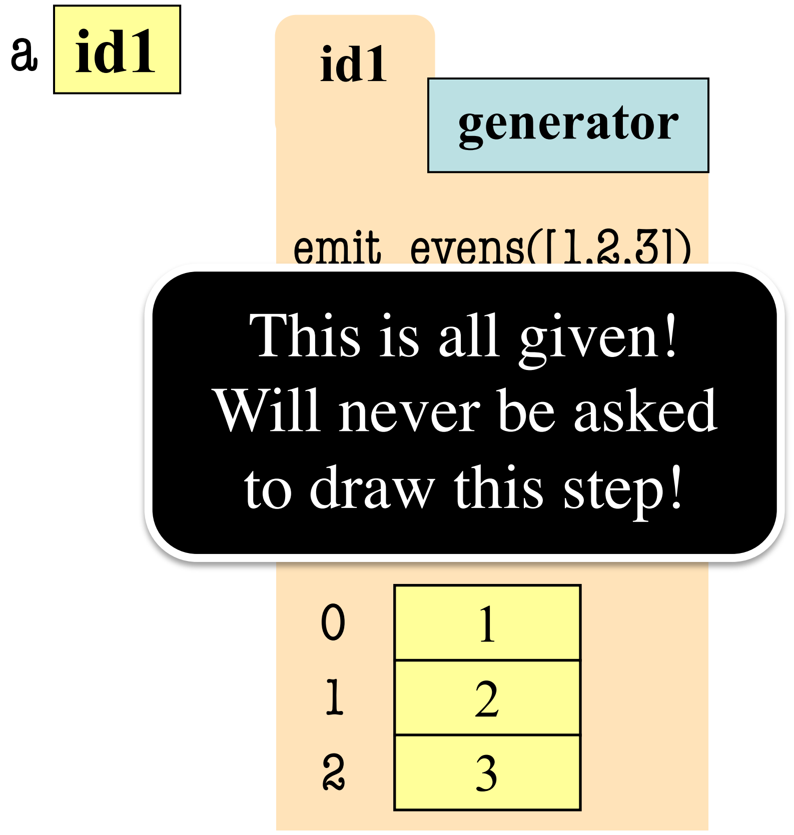


# Generator Call Frames: At the Start

## Generator and Code

```
   def emit_even(input):
       """Gens all even #s in input
          Prec: input list of ints"""
21     for x in input:
22         if x % 2 == 0:
23             yield x
24
25 # Code to execute
26 a = emit_evens([1,2,3])
27 b = next(a)
```

## Given After Line 26



This is all given!
Will never be asked
to draw this step!

# Generator Call Frames: At the Start

## Generator and Code

```
    def emit_even(input):
        """Gens all even #s in input
           Prec: input list of ints"""
21      for x in input:
22          if x % 2 == 0:
23              yield x
24
25  # Code to execute
26  a = emit_evens([1,2,3])
27  b = next(a)
```

## Given After Line 26

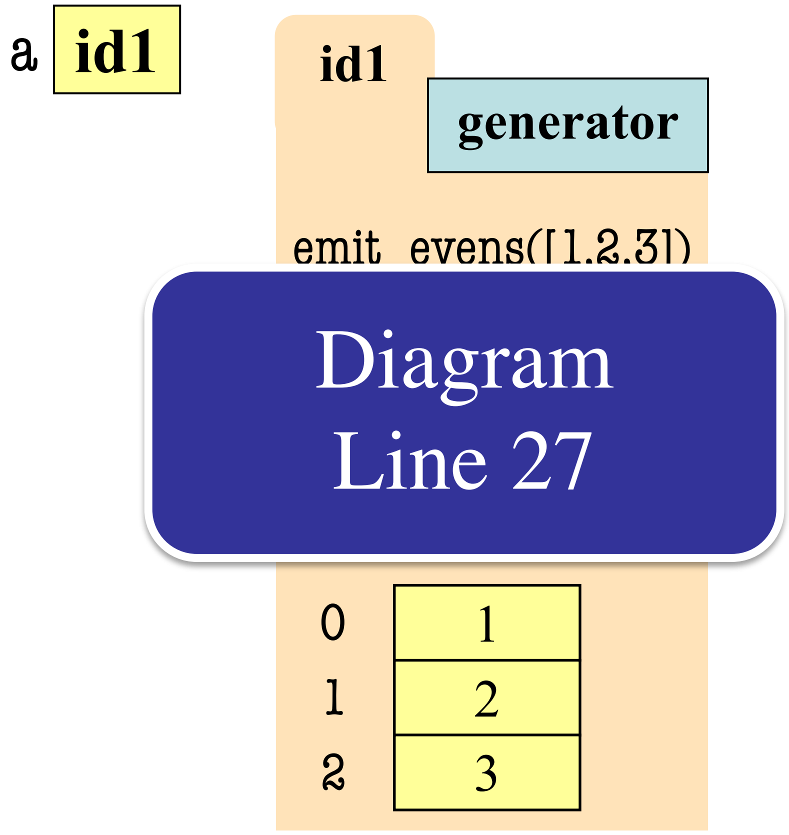


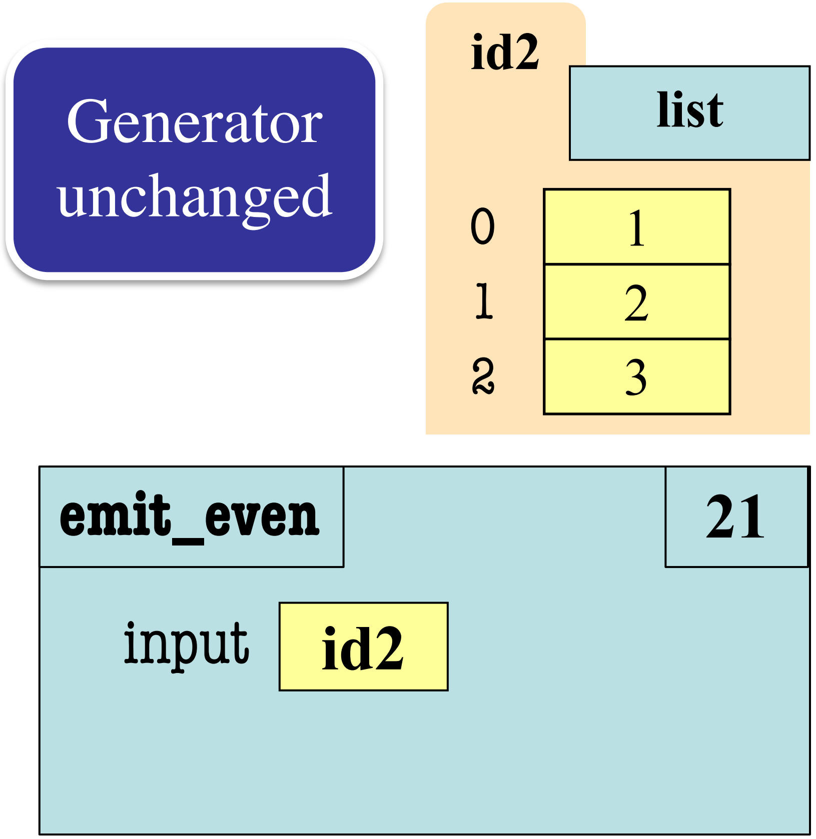
Diagram
Line 27

# Generator Call Frames: At the Start

## Generator and Code

```
   def emit_even(input):
       """Gens all even #s in input
          Prec: input list of ints"""
21     for x in input:
22         if x % 2 == 0:
23             yield x
24
25 # Code to execute
26 a = emit_evens([1,2,3])
27 b = next(a)
```

## Initial Diagram

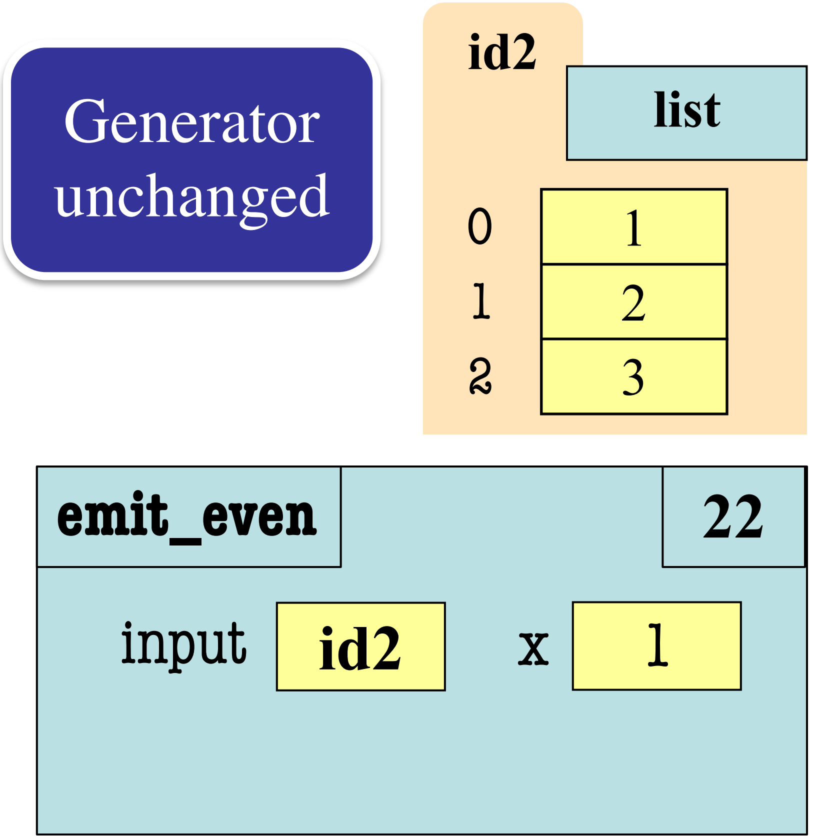


# Generator Call Frames: At the Start

## Generator and Code

```
   def emit_even(input):
       """Gens all even #s in input
          Prec: input list of ints"""
21     for x in input:
22         if x % 2 == 0:
23             yield x
24
25 # Code to execute
26 a = emit_evens([1,2,3])
27 b = next(a)
```

## Diagram Step 2

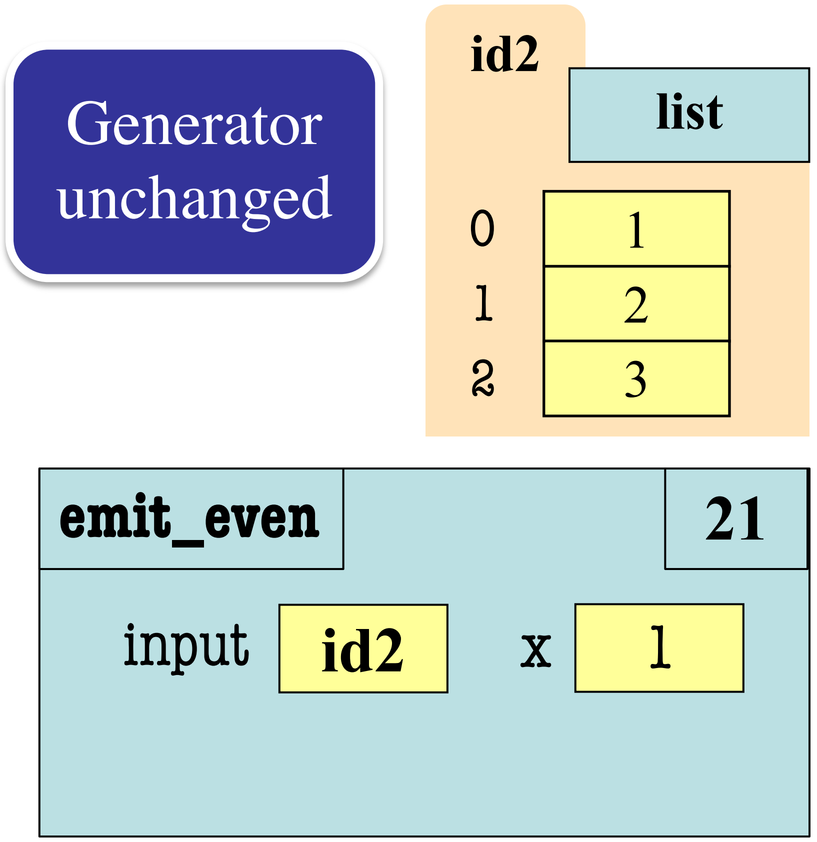


# Generator Call Frames: At the Start

## Generator and Code

```
     def emit_even(input):
         """Gens all even #s in input
            Prec: input list of ints"""
21       for x in input:
22           if x % 2 == 0:
23               yield x
24
25   # Code to execute
26   a = emit_evens([1,2,3])
27   b = next(a)
```

## Diagram Step 3

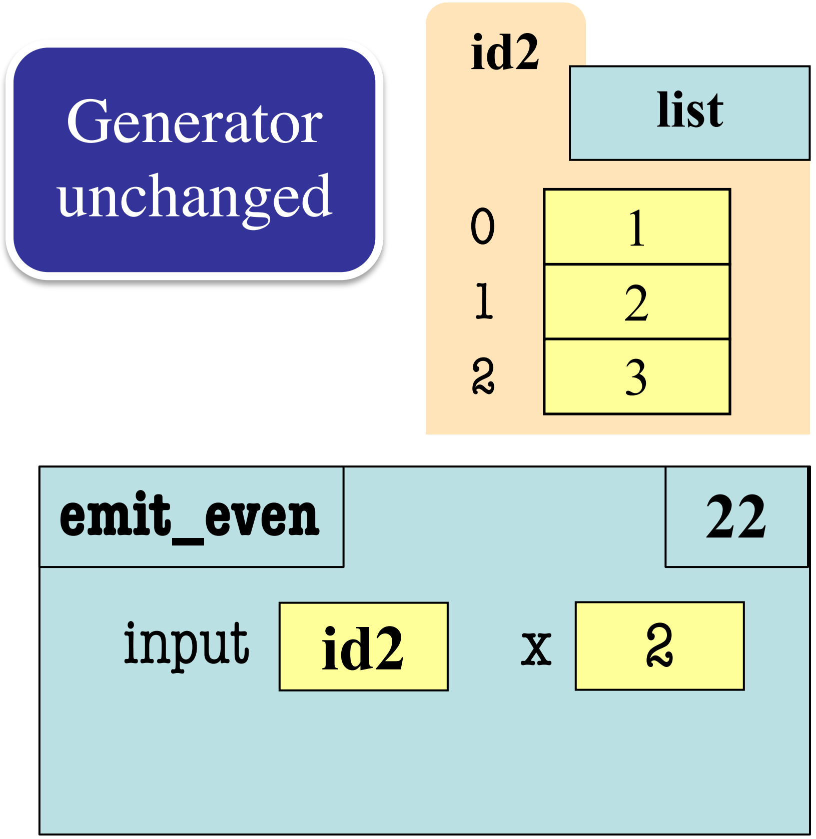


# Generator Call Frames: At the Start

## Generator and Code

```
   def emit_even(input):
       """Gens all even #s in input
          Prec: input list of ints"""
21     for x in input:
22         if x % 2 == 0:
23             yield x
24
25 # Code to execute
26 a = emit_evens([1,2,3])
27 b = next(a)
```

## Diagram Step 4

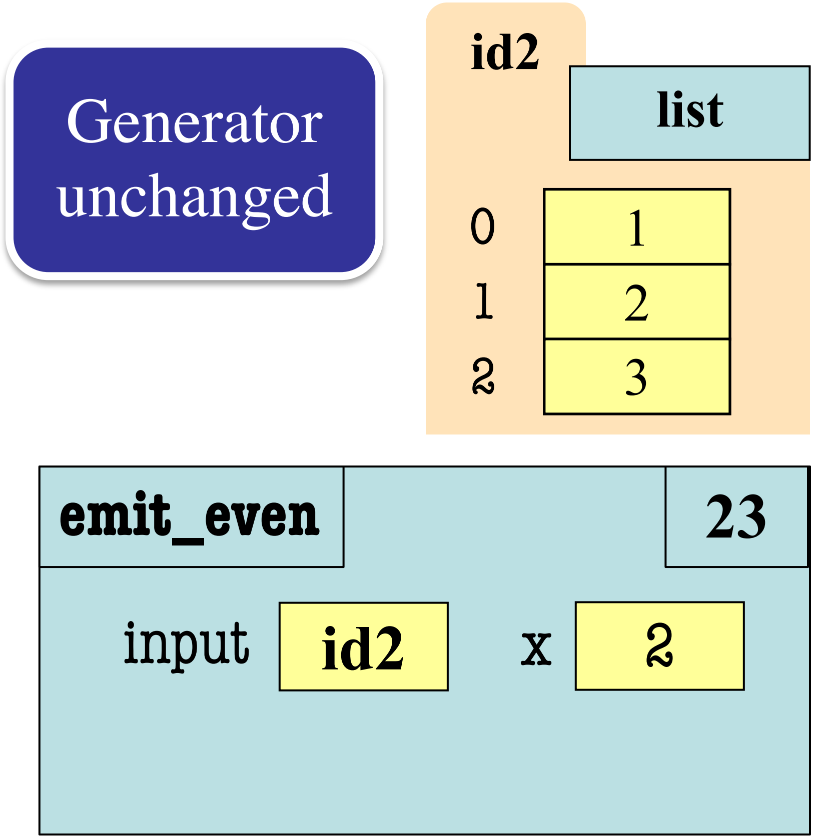


# Generator Call Frames: At the Start

## Generator and Code

```
    def emit_even(input):
        """Gens all even #s in input
           Prec: input list of ints"""
21      for x in input:
22          if x % 2 == 0:
23              yield x
24
25  # Code to execute
26  a = emit_evens([1,2,3])
27  b = next(a)
```

## Diagram Step 5



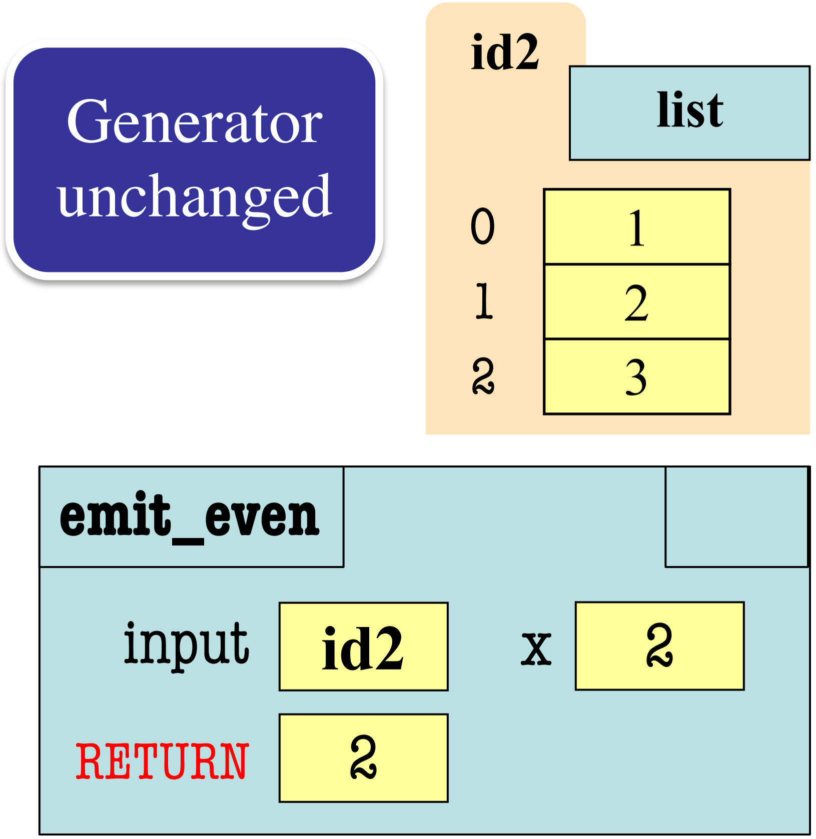
# Generator Call Frames: At the Start

## 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 6



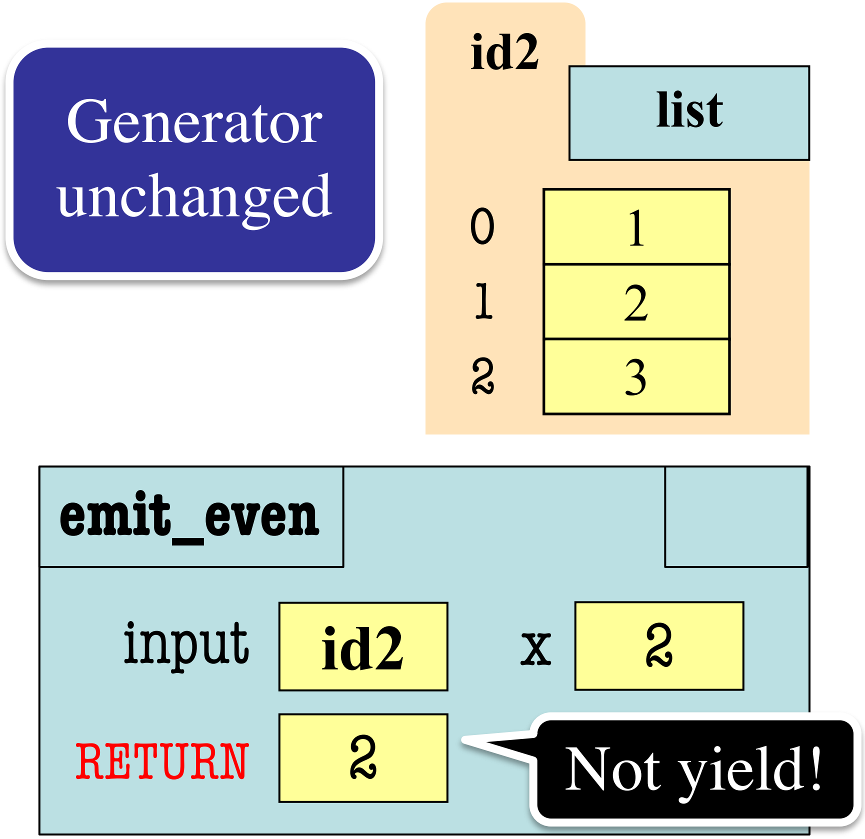
# Generator Call Frames: At the Start

## 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 6



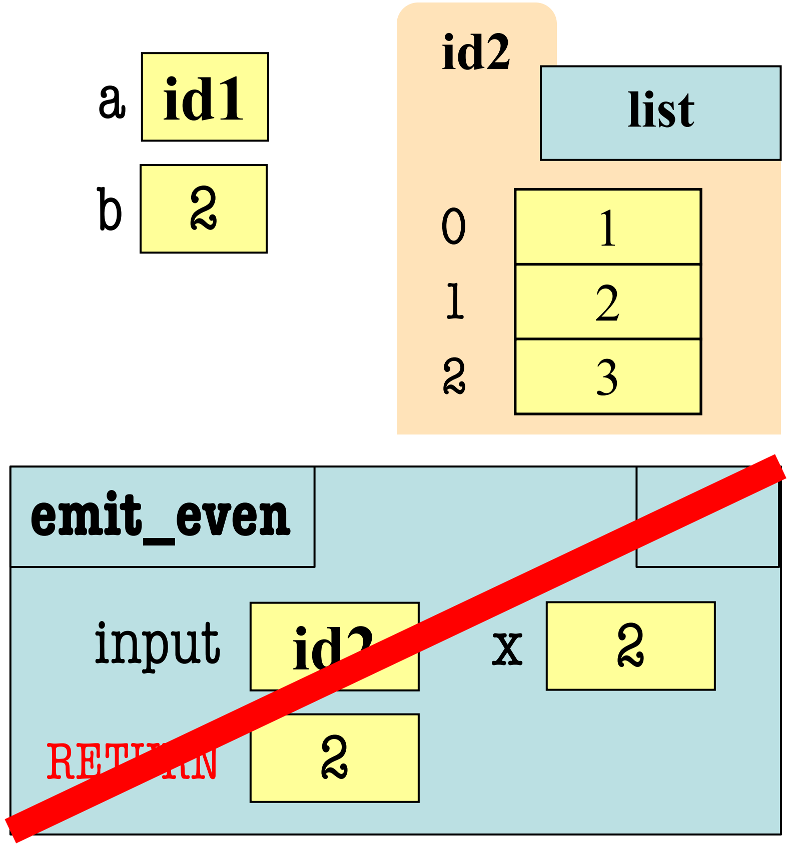
# Generator Call Frames: At the Start

## 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)
```

## Erase the Frame

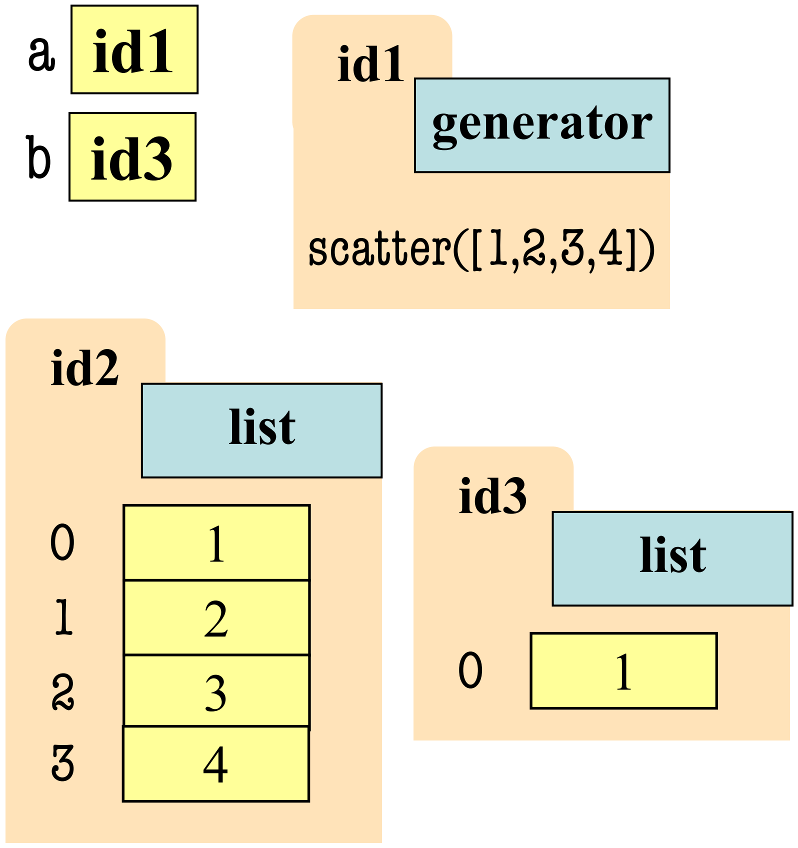


# Generator Call Frames: In Progress

## Generator and Code

```
   def scatter(input):
       """Gens input as 1-elt lists"""
20     for x in input:
21         item = [x]
22         yield item
23
24 # Code to execute
25 a = scatter([1,2,3,4])
26 b = next(a)
27 c = next(a)
```

## Given After Line 26

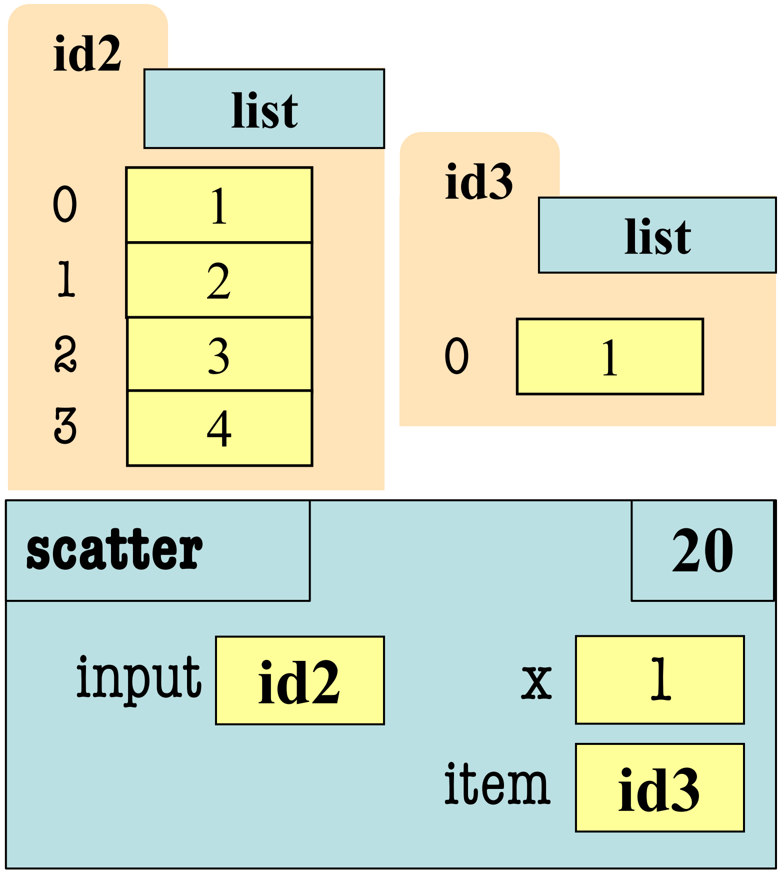


# Generator Call Frames: In Progress

## Generator and Code

```
   def scatter(input):
       """Gens input as 1-elt lists"""
20     for x in input:
21         item = [x]
22         yield item
23
24 # Code to execute
25 a = scatter([1,2,3,4])
26 b = next(a)
27 c = next(a)
```

## Initial Step

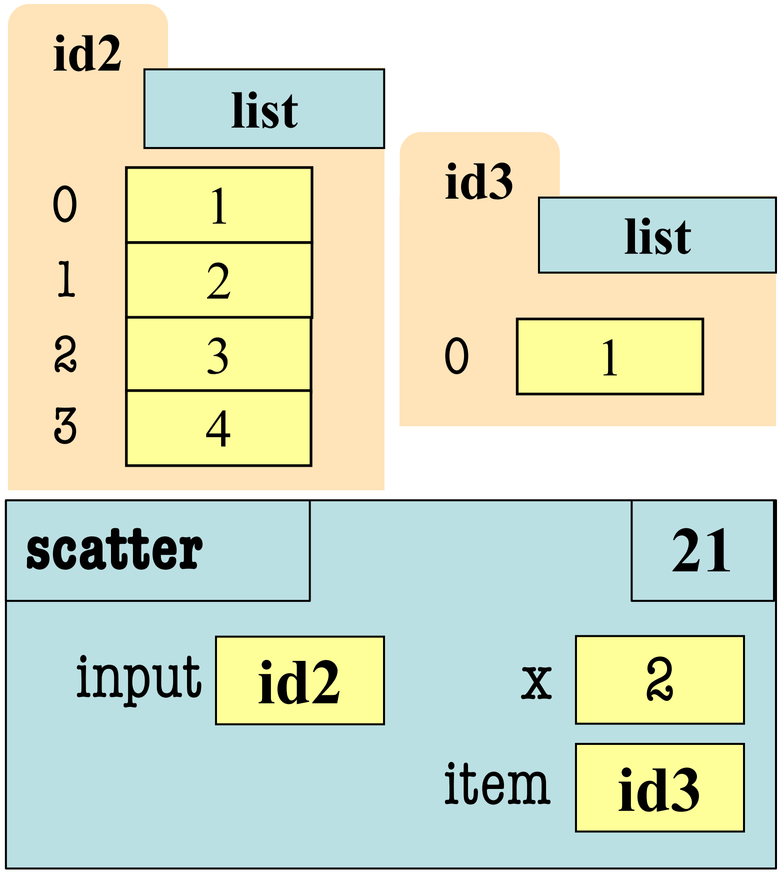


# Generator Call Frames: In Progress

## Generator and Code

```
   def scatter(input):
       """Gens input as 1-elt lists"""
20     for x in input:
21         item = [x]
22         yield item
23
24 # Code to execute
25 a = scatter([1,2,3,4])
26 b = next(a)
27 c = next(a)
```

## Diagram Step 2

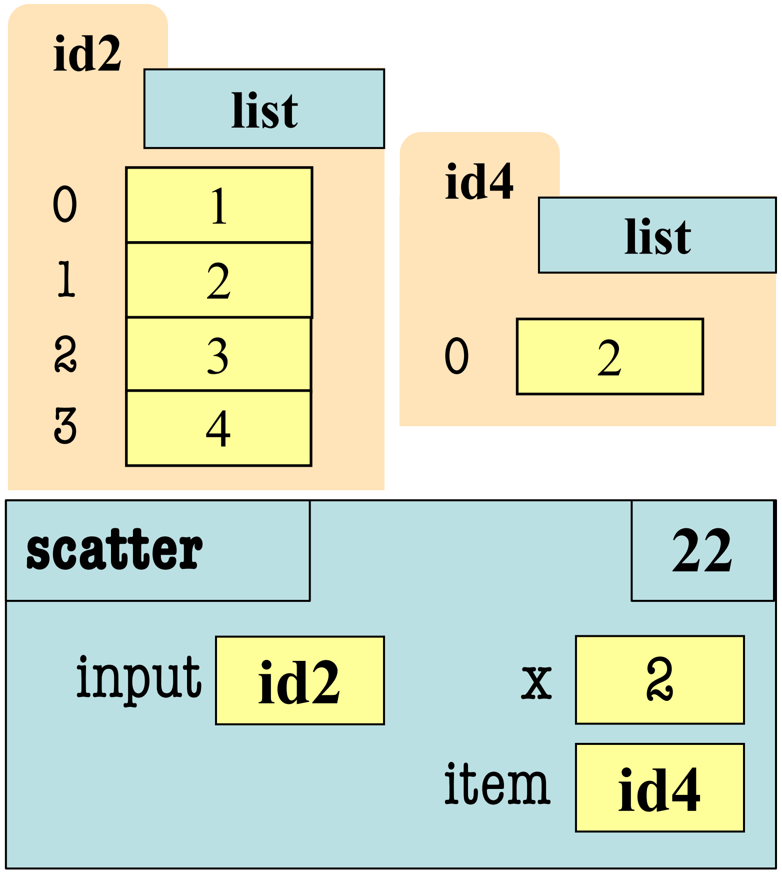


# Generator Call Frames: In Progress

## Generator and Code

```
   def scatter(input):
       """Gens input as 1-elt lists"""
20     for x in input:
21         item = [x]
22         yield item
23
24 # Code to execute
25 a = scatter([1,2,3,4])
26 b = next(a)
27 c = next(a)
```

## Diagram Step 3



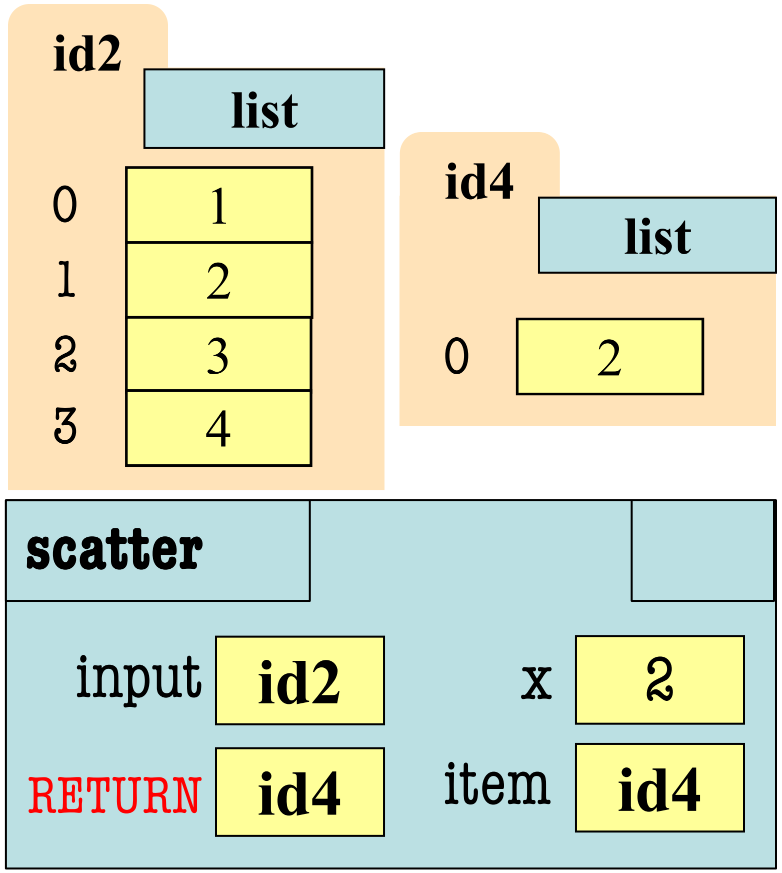
# Generator Call Frames: In Progress

## Generator and Code

```
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



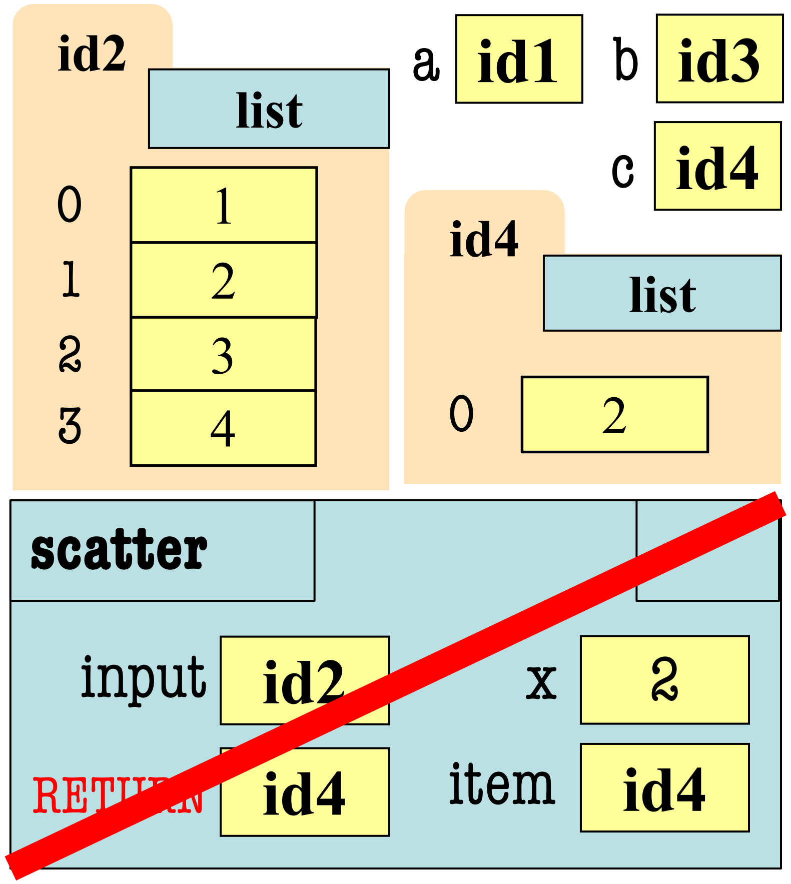
# Generator Call Frames: In Progress

## Generator and Code

```
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)
```

## Erase the Frame



# Generators and Functions

## Function Definitions

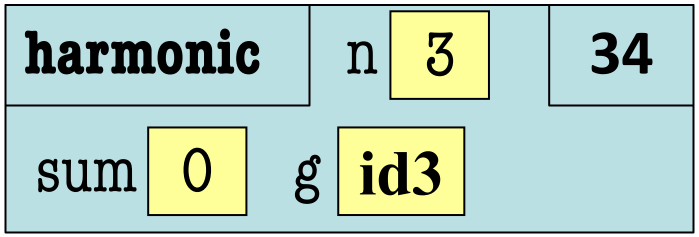
```
   def rnginv(n):        #Inverse range
19    for x in range(1,n):
20        yield 1/x

   def harmonic(n):     #Harmonic sum
32    sum = 0
33    g = rnginv(n)
34    for x in g:
35        sum = sum+x
36    return x
```

## Function Call

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

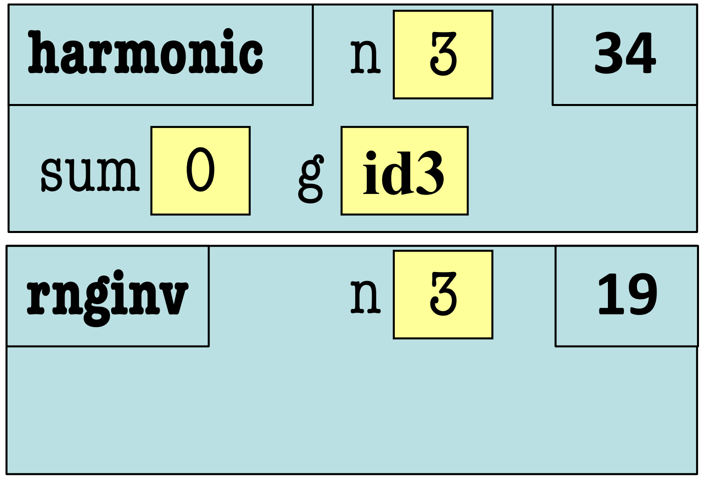
Assume we are here:



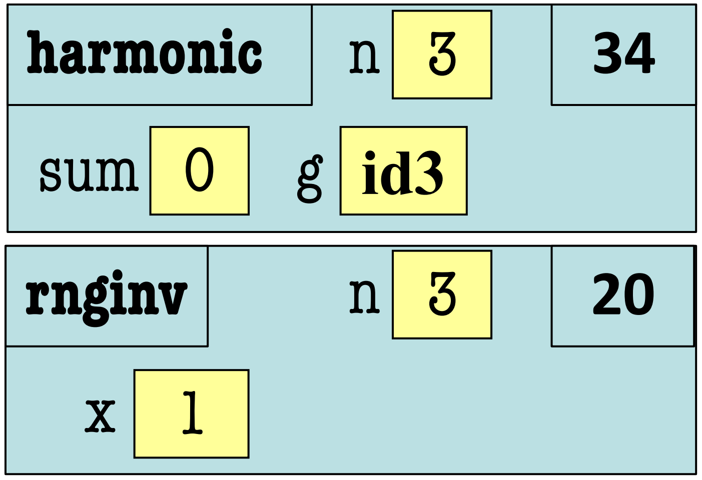
**Ignoring the heap**, what is the **next step**?

# Which One is Closest to Your Answer?

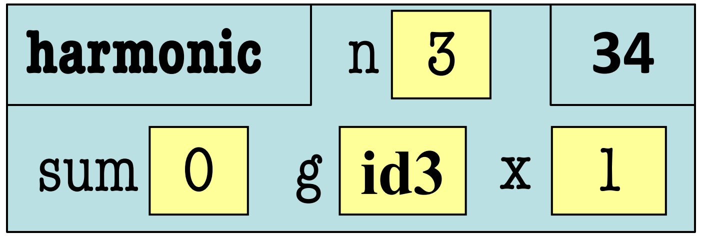
A:



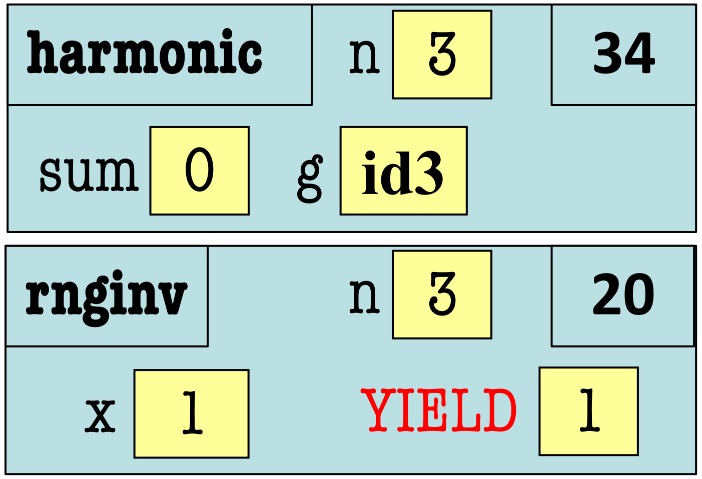
B:



C:



D:



# Generators and Functions

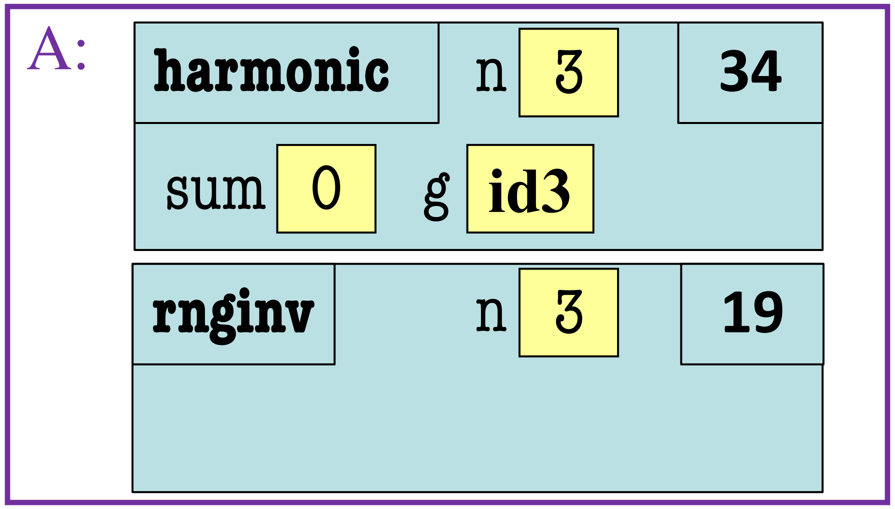
## Function Defintions

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

    def harmonic(n):    #Harmonic sum
32      sum = 0
33      g = rnginv(n)
34      for x in g:
35          sum = sum+x
36      return x
```

## Function Call

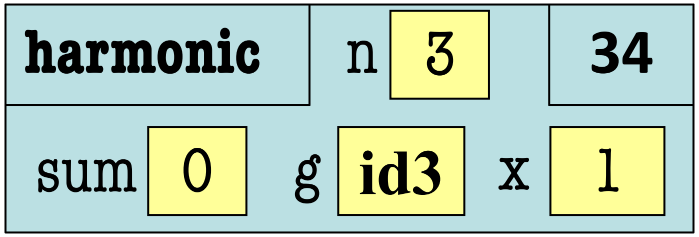
```
>>> x = harmonic(3)
```



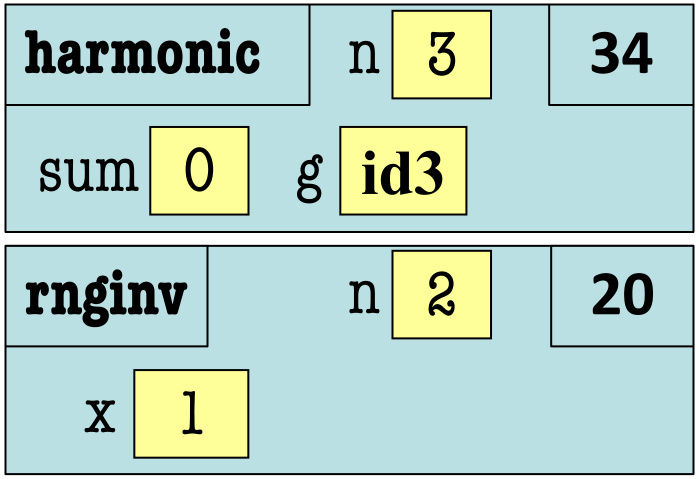
What is the **next step**?

# Which One is Closest to Your Answer?

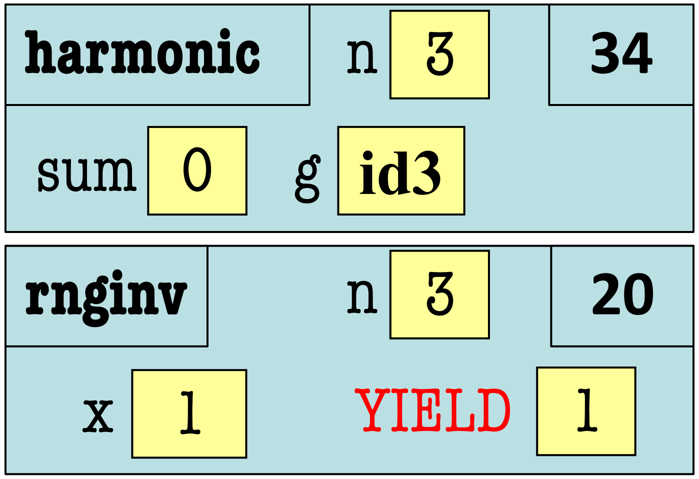
A:



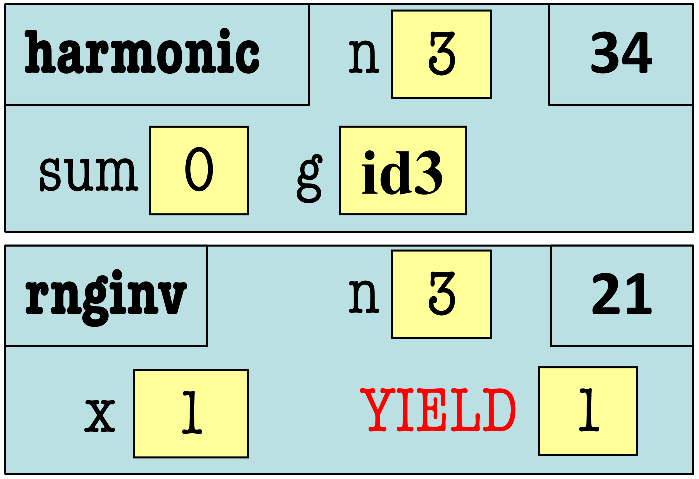
B:



C:



D:



# Generators and Functions

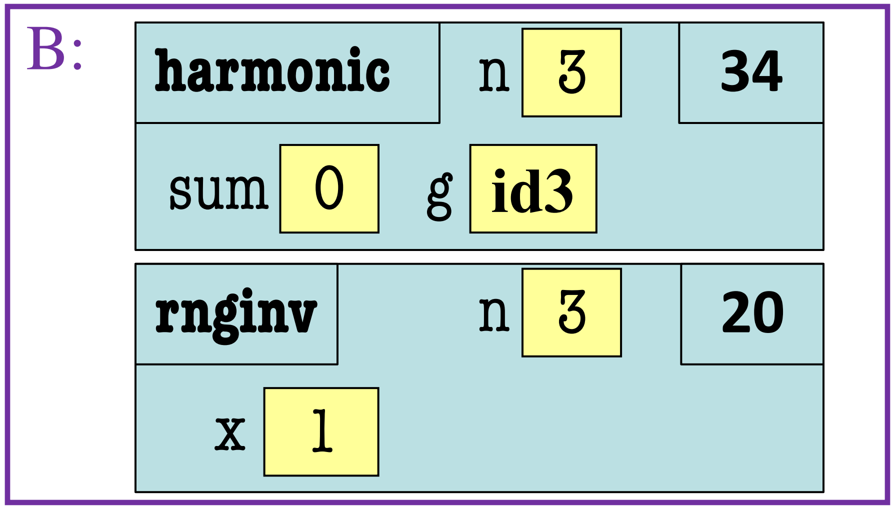
## Function Defintions

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

    def harmonic(n):     #Harmonic sum
32      sum = 0
33      g = rnginv(n)
34      for x in g:
35          sum = sum+x
36      return x
```

## Function Call

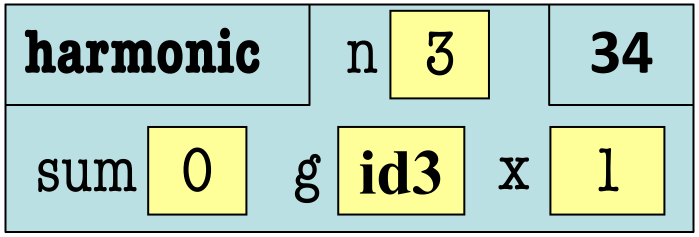
```
>>> x = harmonic(3)
```



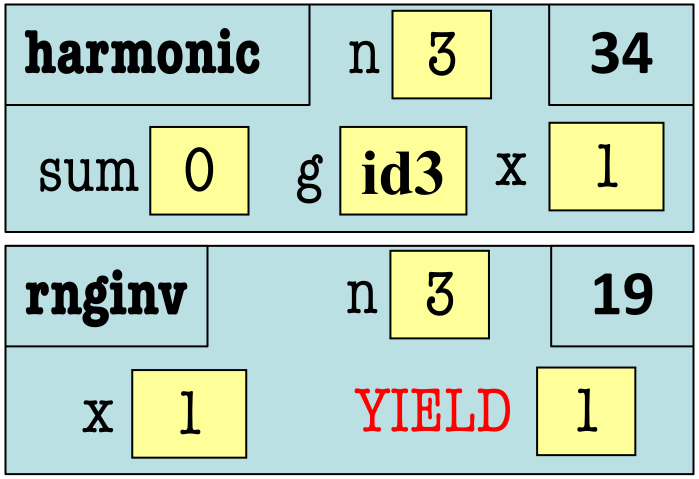
What is the **next step**?

# Which One is Closest to Your Answer?

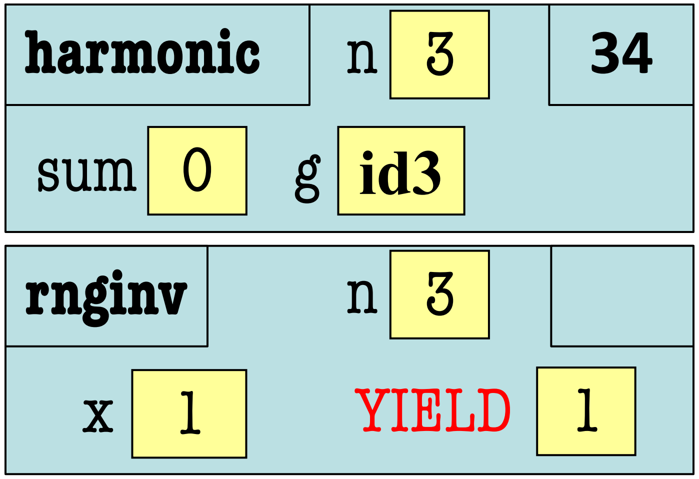
A:



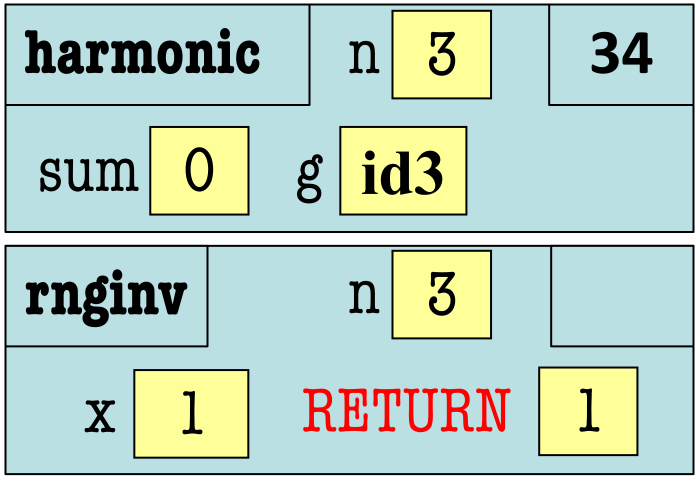
B:



C:



D:



# Generators and Functions

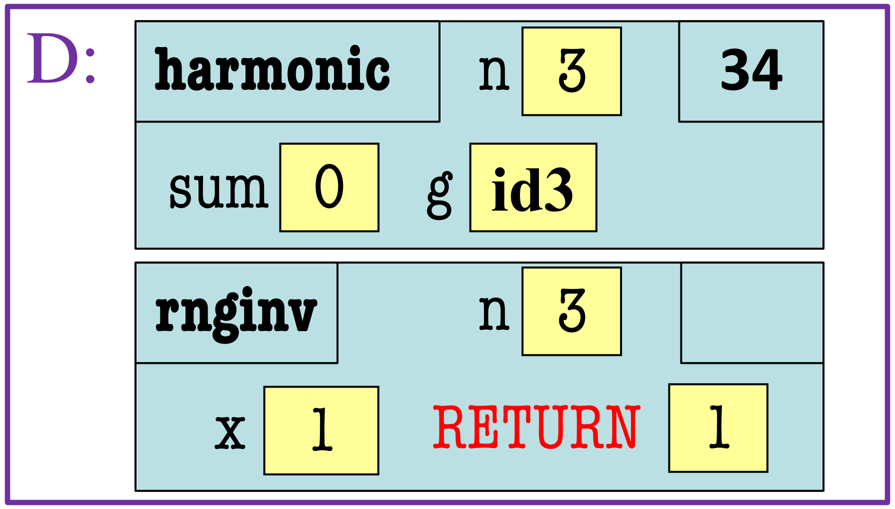
## Function Defintions

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

    def harmonic(n):    #Harmonic sum
32      sum = 0
33      g = rnginv(n)
34      for x in g:
35          sum = sum+x
36      return x
```

## Function Call

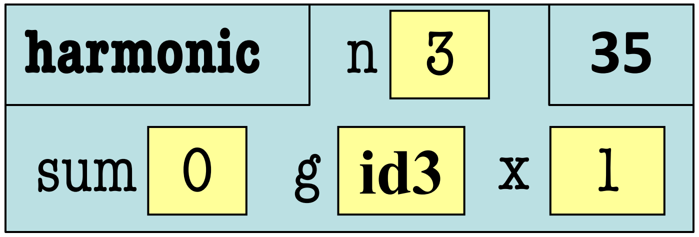
```
>>> x = harmonic(3)
```



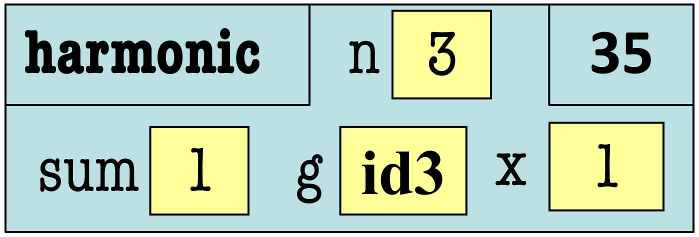
What is the **next step**?

# Which One is Closest to Your Answer?

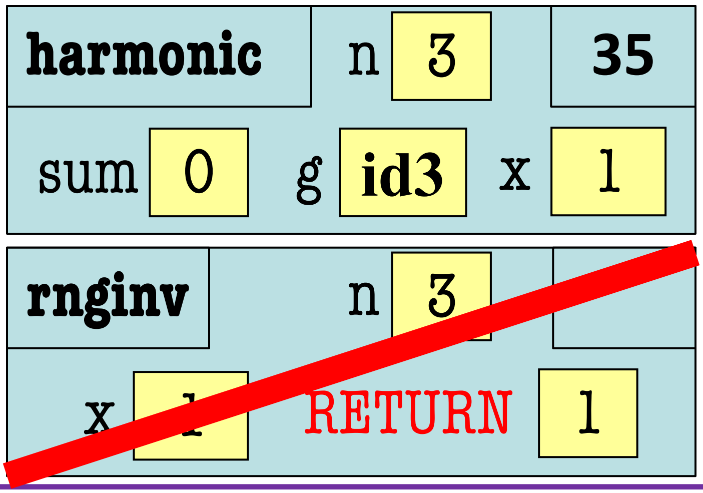
A:



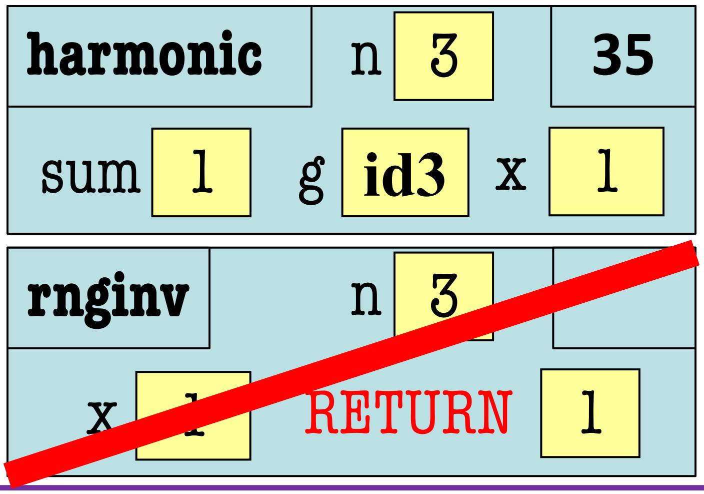
B:



C:



D:



# Generators and Functions

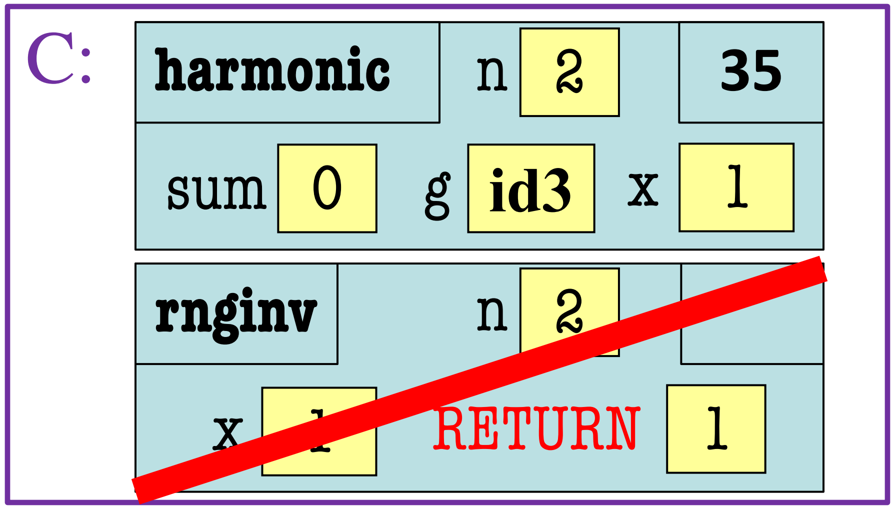
## Function Defintions

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

   def harmonic(n):    #Harmonic sum
32   sum = 0
33   g = rnginv(n)
34   for x in g:
35     sum = sum+x
36   return x
```

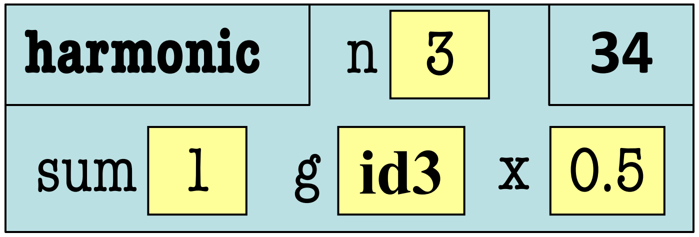
## Function Call

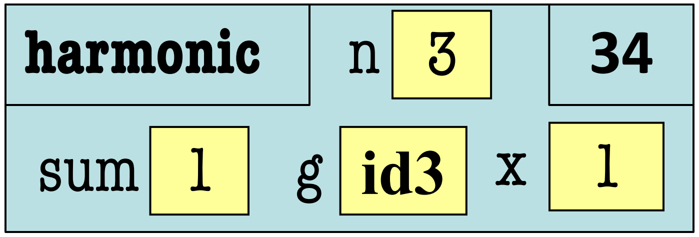
```
>>> x = harmonic(2)
```

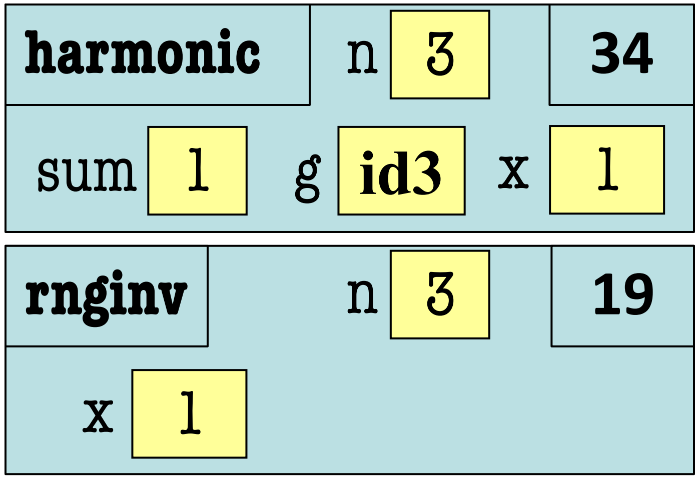


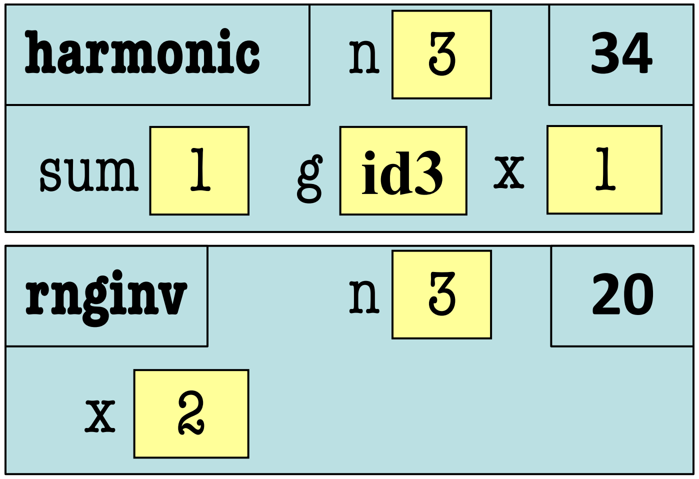
What is the **next step**?

# Which One is Closest to Your Answer?

A:


B:


C:


D:


# Generators and Functions

## Function Defintions

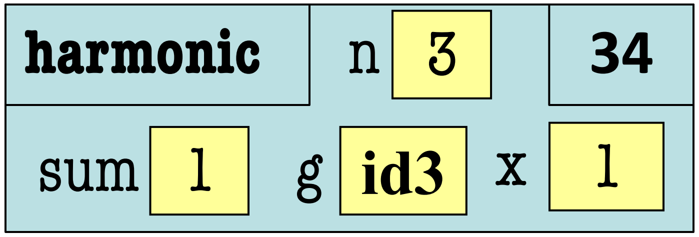
```
   def rnginv(n):       #Inverse range
19   for x in range(1,n):
20     yield 1/x

   def harmonic(n):   #Harmonic sum
32   sum = 0
33   g = rnginv(n)
34   for x in g:
35     sum = sum+x
36   return x
```

## Function Call

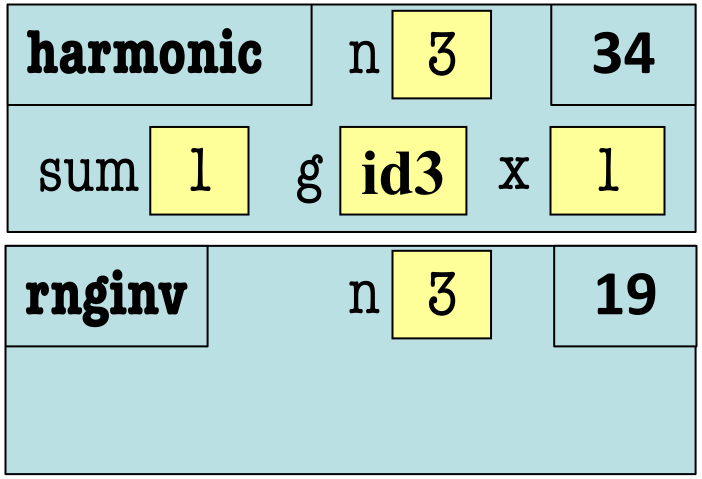
```
>>> x = harmonic(2)
```

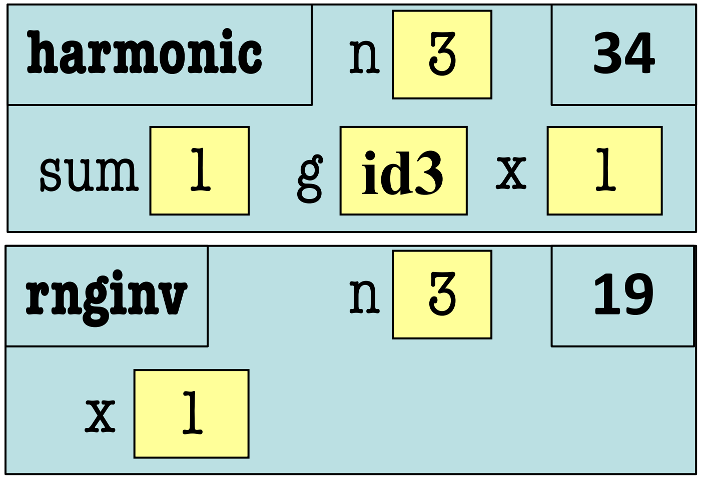
B:

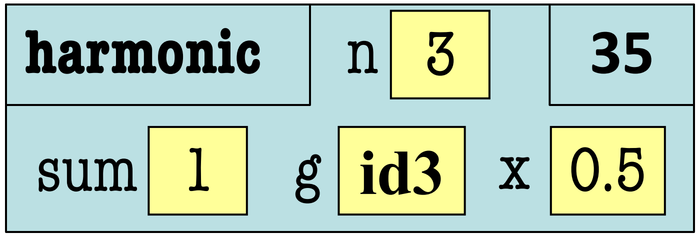


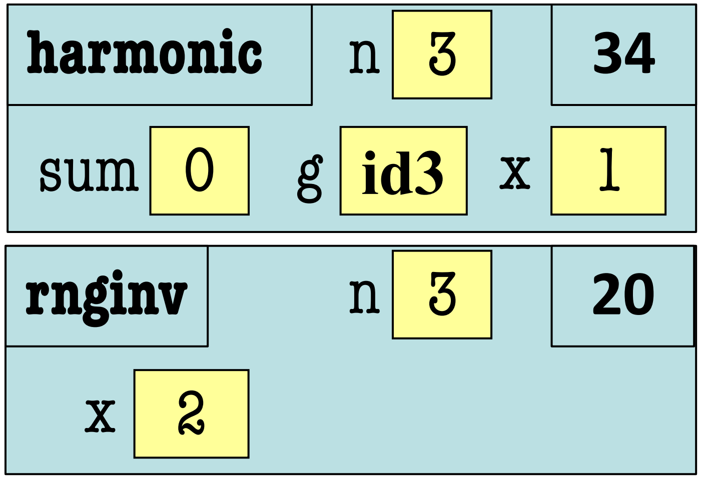
What is the **next step**?

# Which One is Closest to Your Answer?

A:


B:


C:


D:


## Generators and Functions

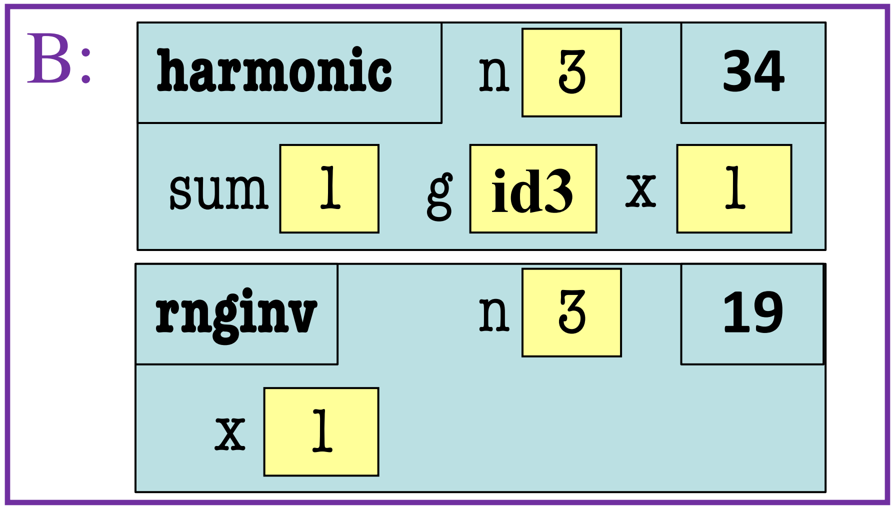
### Function Defintions

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

   def harmonic(n):    #Harmonic sum
32   sum = 0
33   g = rnginv(n)
34   for x in g:
35     sum = sum+x
36   return x
```

### Function Call

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



Try the rest on your own

## Questions?