CS61A

section 3

---

attendance (no password today)

http://links.cs61a.org/jasonxu

upcoming

hw 3
hog contest ~ optional

CS61A

my thoughts

definitely difficult 🤬

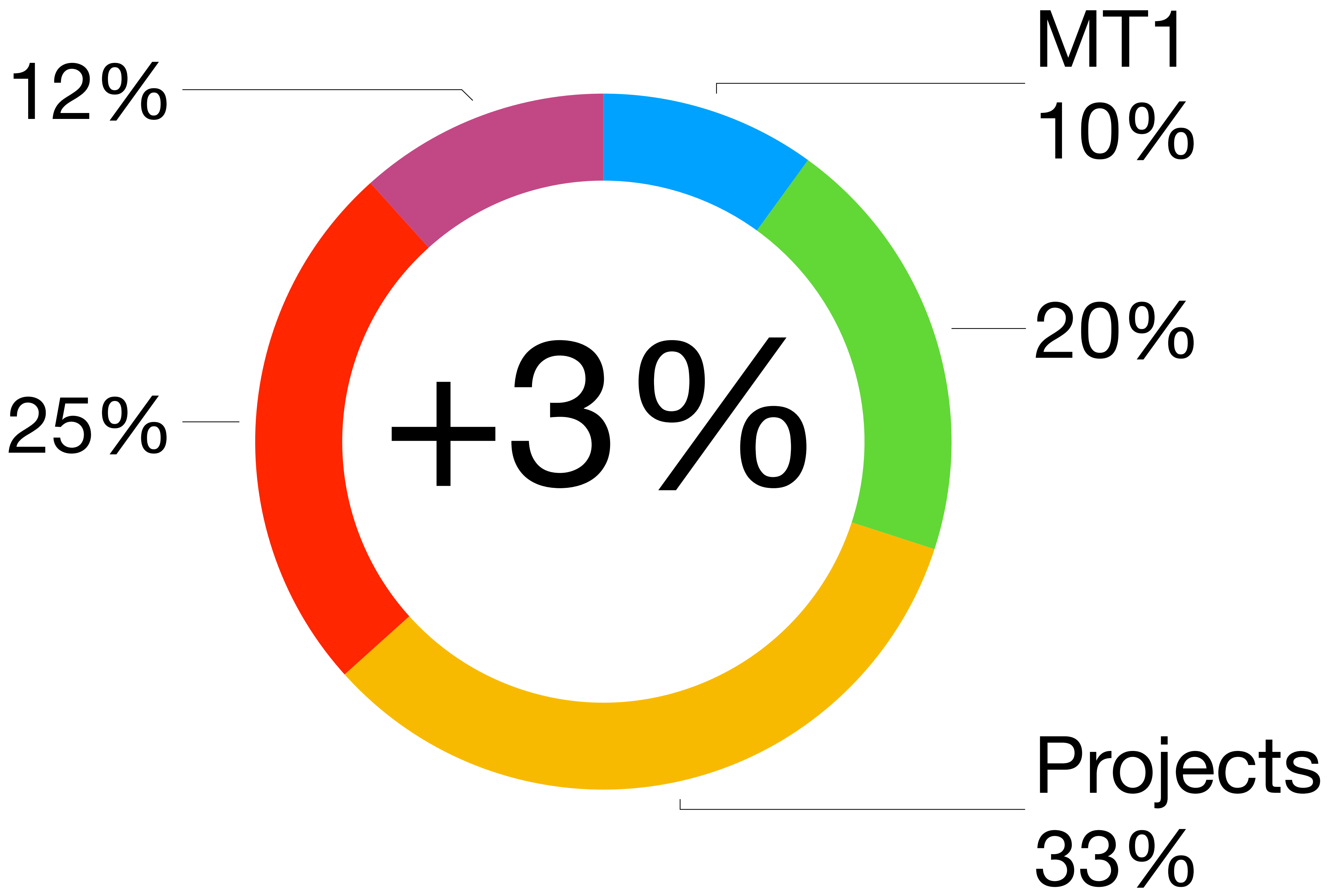
midterm recovery points

CS61A

my thoughts

definitely difficult 🤬

61A has *a lot* of resources



CS61A

recursion

things *defined* by themselves

CS61A

recursion

factorial!

```
def factorial(n):
    if n == 0:
       return 1
    else:
       return n * factorial(n - 1)
```

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

CS61A

recursion

factorial!

```
def factorial(n):
    if n == 0:
       return 1
    else:
       return n * factorial(n - 1)
```

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

uhhhhhhhh

when do i stop?

0! = 0 * -1!

-1! = -1 * -2! ...

CS61A

recursion

factorial!

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

5! = 5 * 4!

4! = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

uhhhhhhhh

base case!

0! = 0 * -1!

-1! = -1 * -2! ...

CS61A

recursion

factorial!

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

$5! = 5 * 4!$

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1!$

$1! = 1 * 0!$

yay! 😍

base case!

$0! = 1$

CS61A

recursion

factorial!

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

how do we come up with this

by definition, 😱

by assuming it works, 🤪

pattern: how can we solve sub-problems to solve the current problem?

CS61A

recursion

factorial!

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

how do we calculate 5!

$5! = 5 * 4!$

for this to be true, don’t we have to assume that ‘!’ really does what it says

well in code we can’t name a function ‘!’

we assume that (n-1)! works

recursive leap of faith

well… i have to test it by tracing it

well… big headache

CS61A

recursion

strategy

if you capture all the base cases

you can assume it works

so you can create the recursive call

CS61A

recursion

analogy

black friday shopping… long line

you want to know how many people in front

accurately, you only know if you're the first person

otherwise, you have to ask the person in front of you for their position

is this a good recursive procedure…?

CS61A

recursion

analogy

black friday shopping… long line

you want to know how many people in front

accurately, you only know if you’re the first person

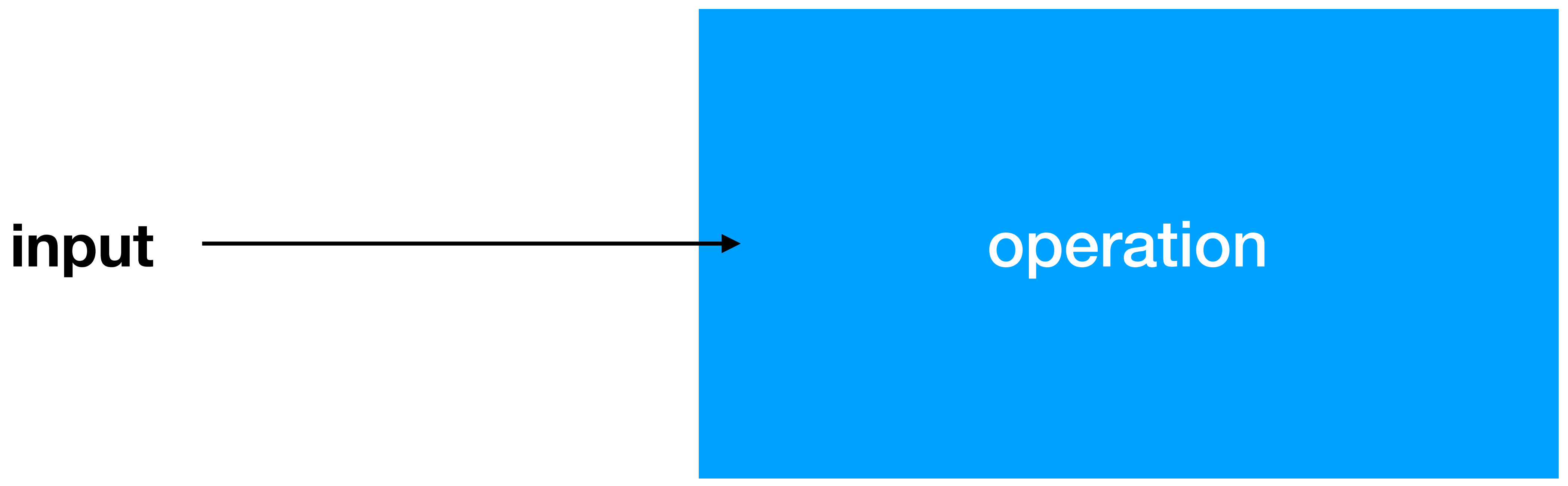
otherwise, you have to ask the person in front of you for their position and add 1

is this a recursive procedure…?

CS61A

recursion

motivation for it



input

operation

CS61A

recursion

things *defined* by themselves

CS61A

recursion

things *defined* by themselves

input

operation

output :)

CS61A

recursion

things *defined* by themselves

input

operation

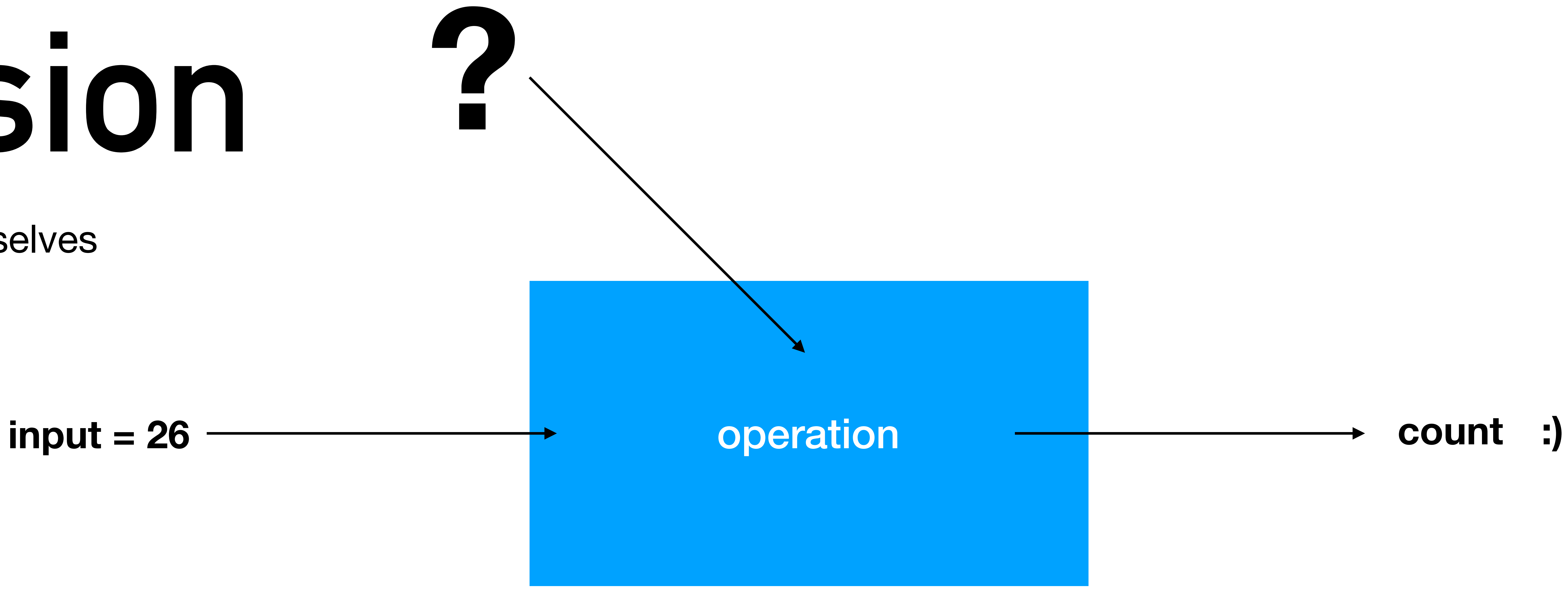
output :)

tell me the number of ways to line $26

CS61A

recursion

things *defined* by themselves



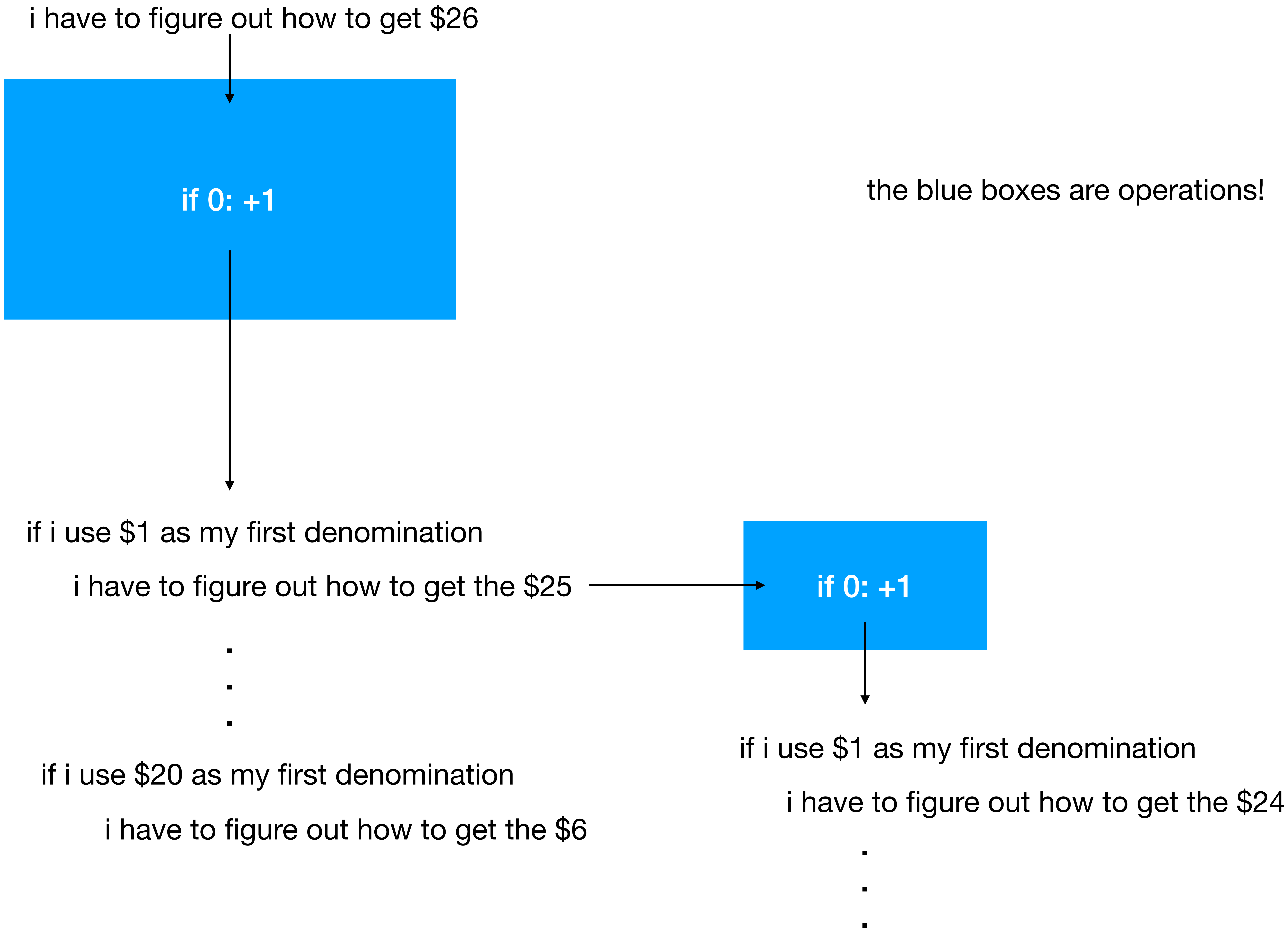
?

input = 26

operation

count :)

tell me the number of ways to line $26



i have to figure out how to get $26

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $25

.
.
.

if i use $20 as my first denomination

i have to figure out how to get the $6

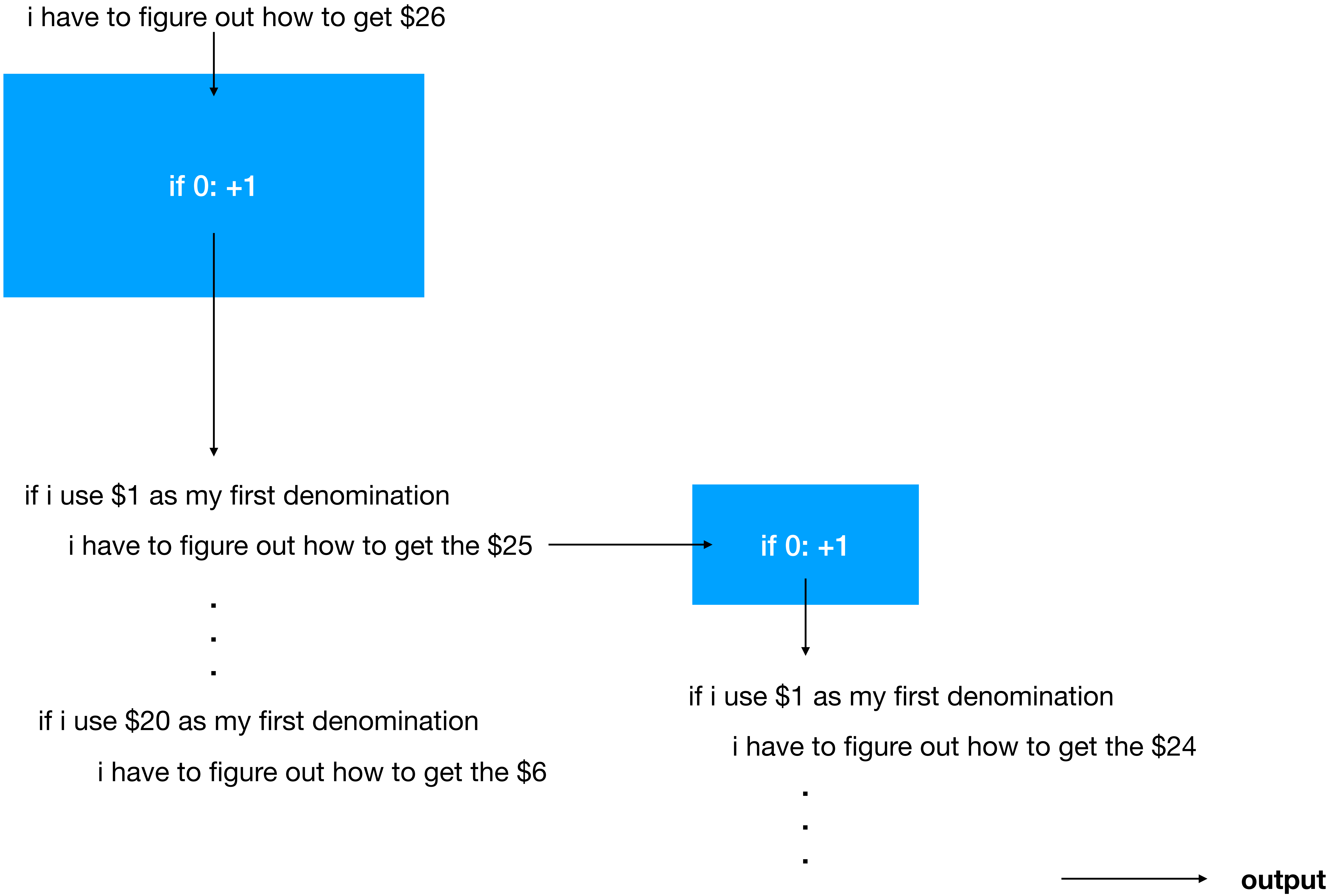
the blue boxes are operations!

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $24

.
.
.



i have to figure out how to get $26

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $25

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $24

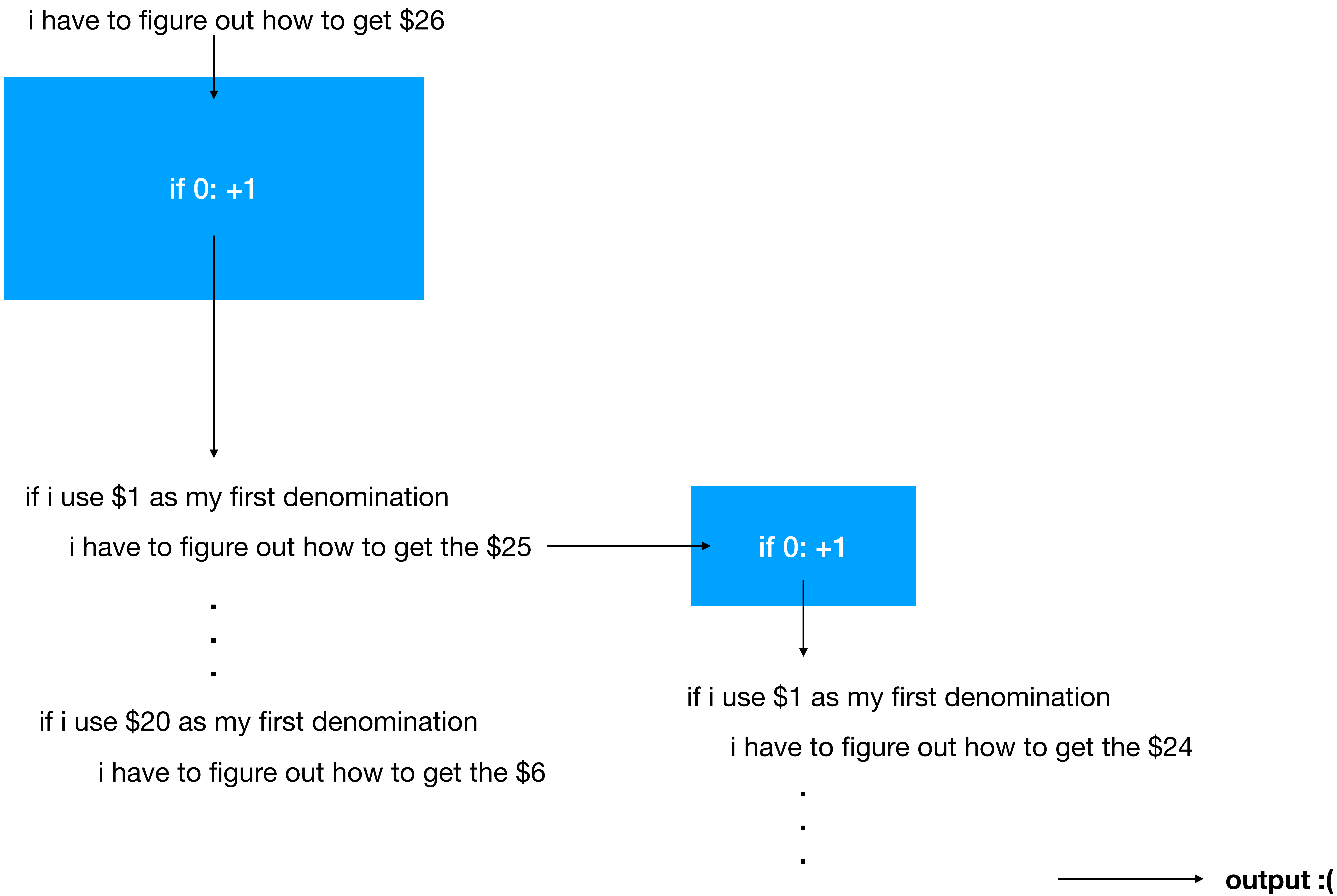
⋮

⋮

if i use $20 as my first denomination

i have to figure out how to get the $6

output



i have to figure out how to get $26

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $25

.
.
.

if i use $20 as my first denomination

i have to figure out how to get the $6

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $24

.
.
.

output :(

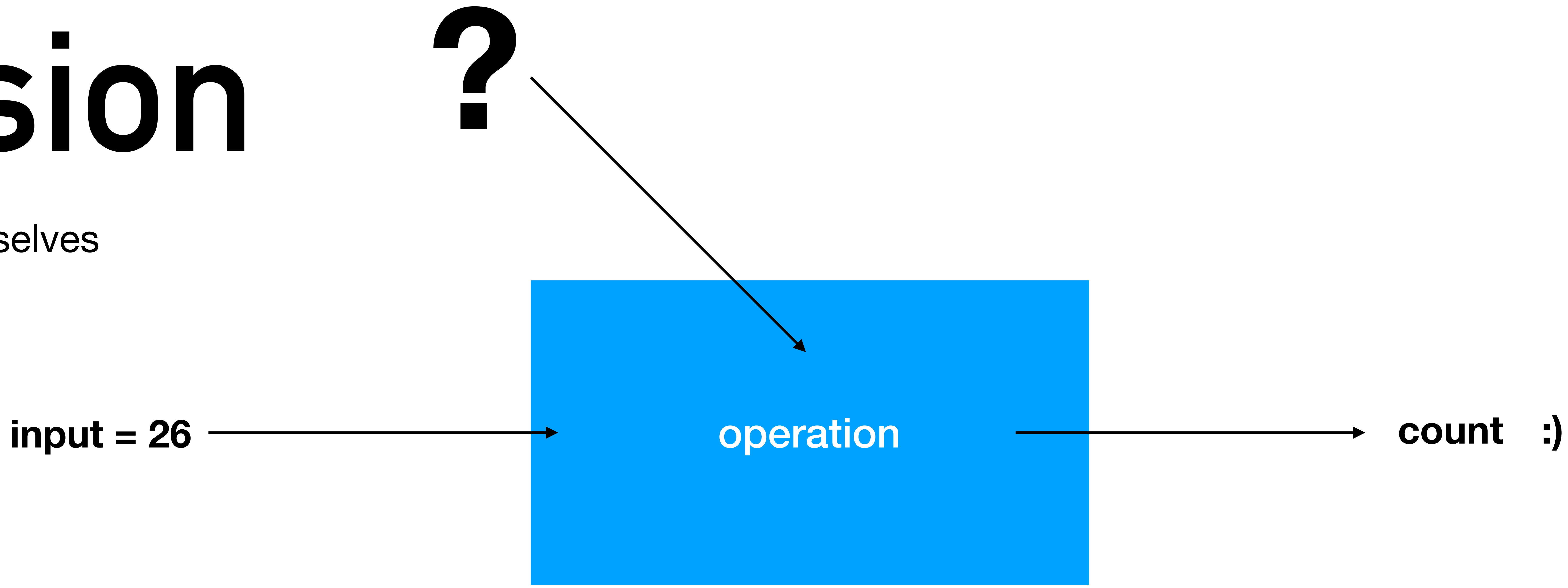
```
def count(n):
    total = 0
    options = [n]
    while len(options) > 0:
        curr = options.pop(0)
        for change in [1, 5, 10, 20]:
            val = curr - change
            if val == 0:
                total += 1
            elif val > 0:
                options.append(val)
    return total
```

```
def count(n):
     total = 0
     options = [n]
     while len(options) > 0:
         curr = options.pop(0)
         for change in [1, 5, 10, 20]:
             val = curr - change
             if val == 0:
                 total += 1
             elif val > 0:
                 options.append(val)
     return total
```

CS61A

recursion

things *defined* by themselves



?

input = 26

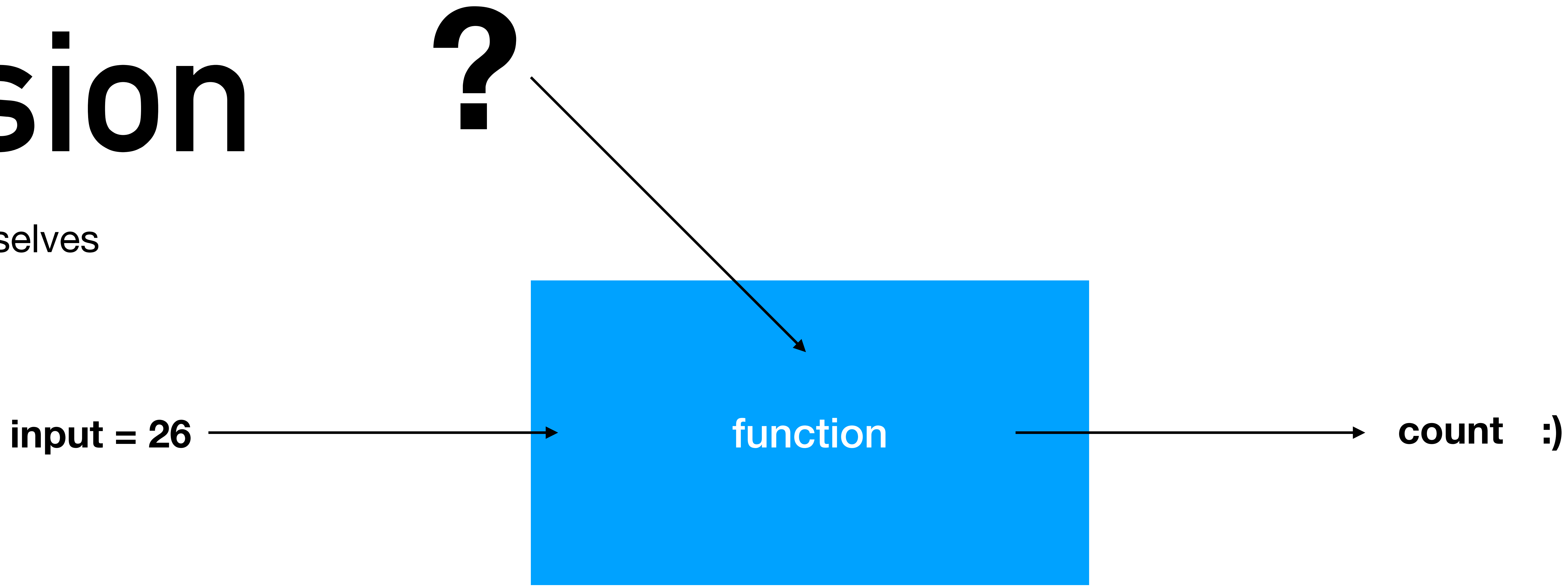
operation

count :)

CS61A

recursion

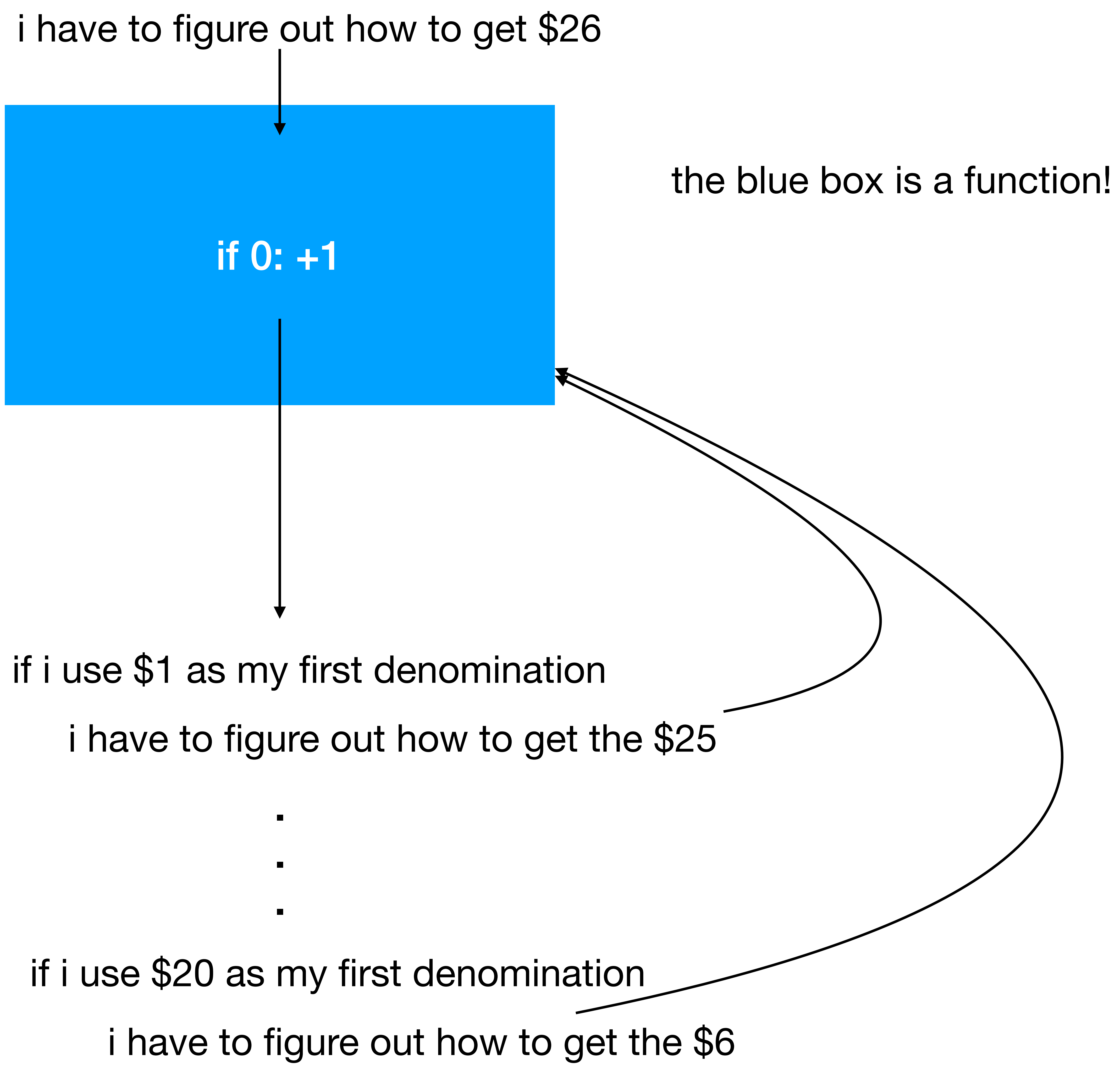
things *defined* by themselves



CS61A

recursion

things *defined* by themselves



i have to figure out how to get $26

if 0: +1

the blue box is a function!

if i use $1 as my first denomination

i have to figure out how to get the $25

.
.
.

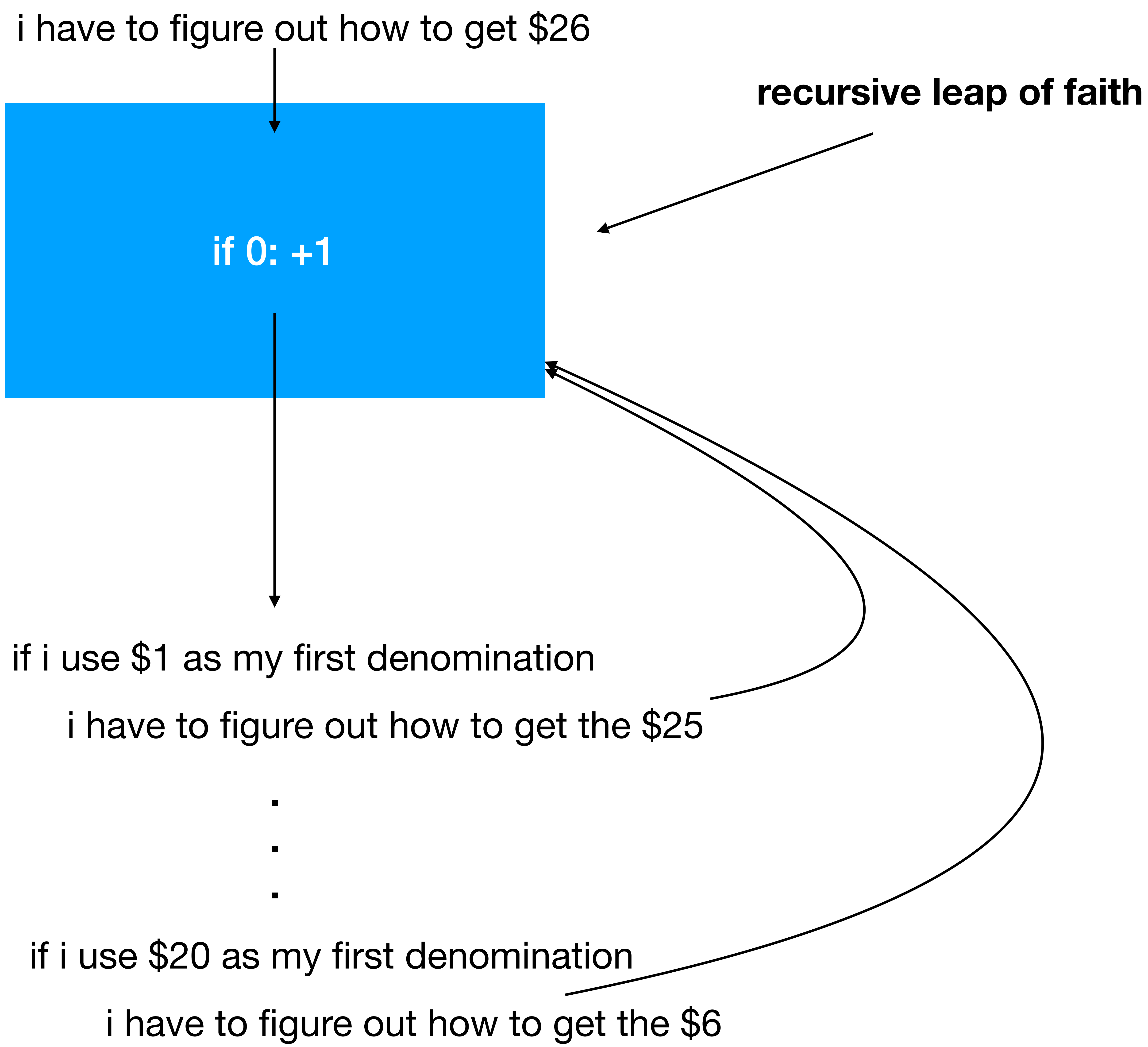
if i use $20 as my first denomination

i have to figure out how to get the $6

CS61A

recursion

things *defined* by themselves



i have to figure out how to get $26

recursive leap of faith

if 0: +1

if i use $1 as my first denomination

i have to figure out how to get the $25

.
.
.

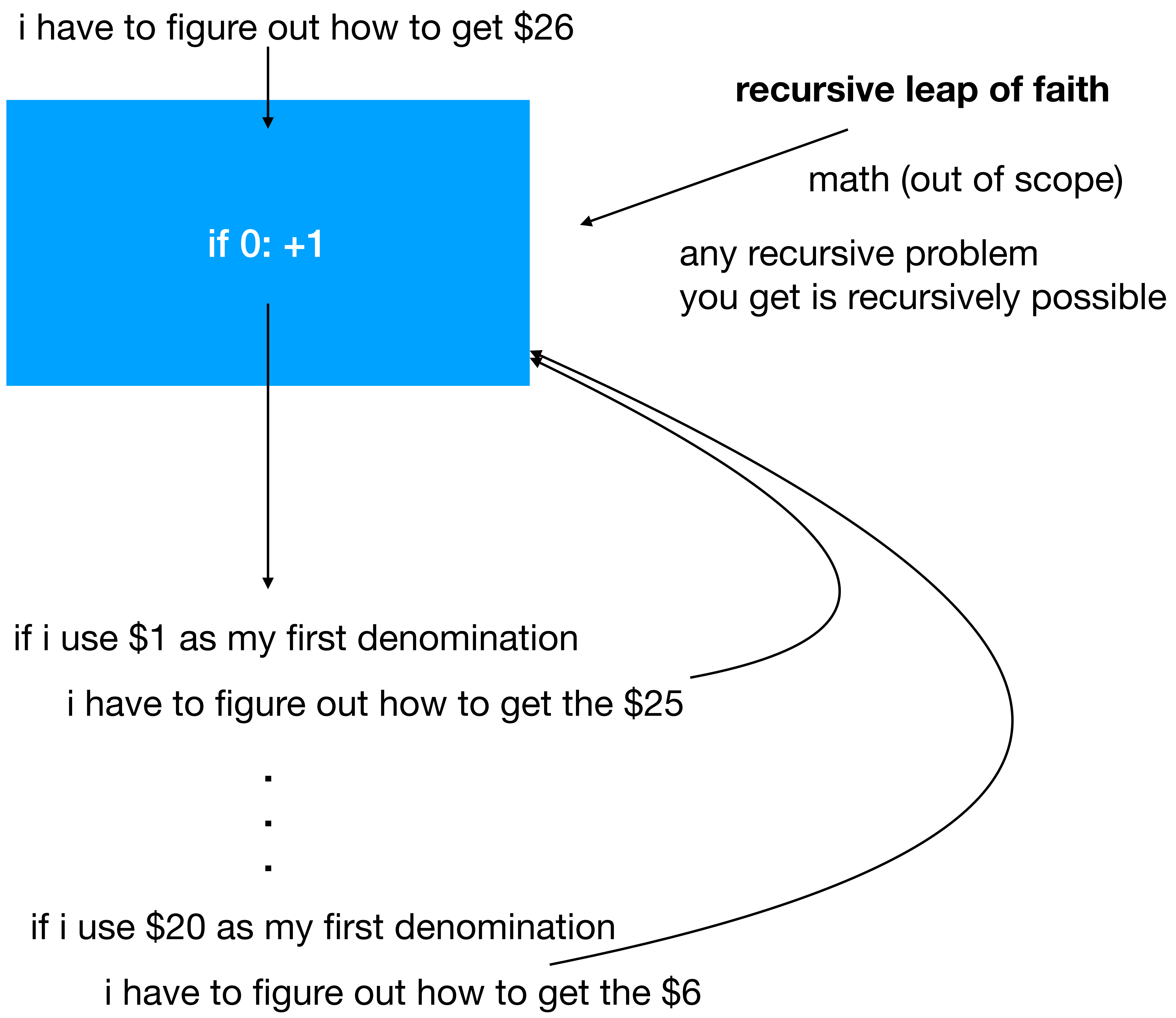
if i use $20 as my first denomination

i have to figure out how to get the $6

CS61A

recursion

things *defined* by themselves



i have to figure out how to get $26

if 0: +1

recursive leap of faith

math (out of scope)

any recursive problem
you get is recursively possible

if i use $1 as my first denomination

i have to figure out how to get the $25

.
.
.

if i use $20 as my first denomination

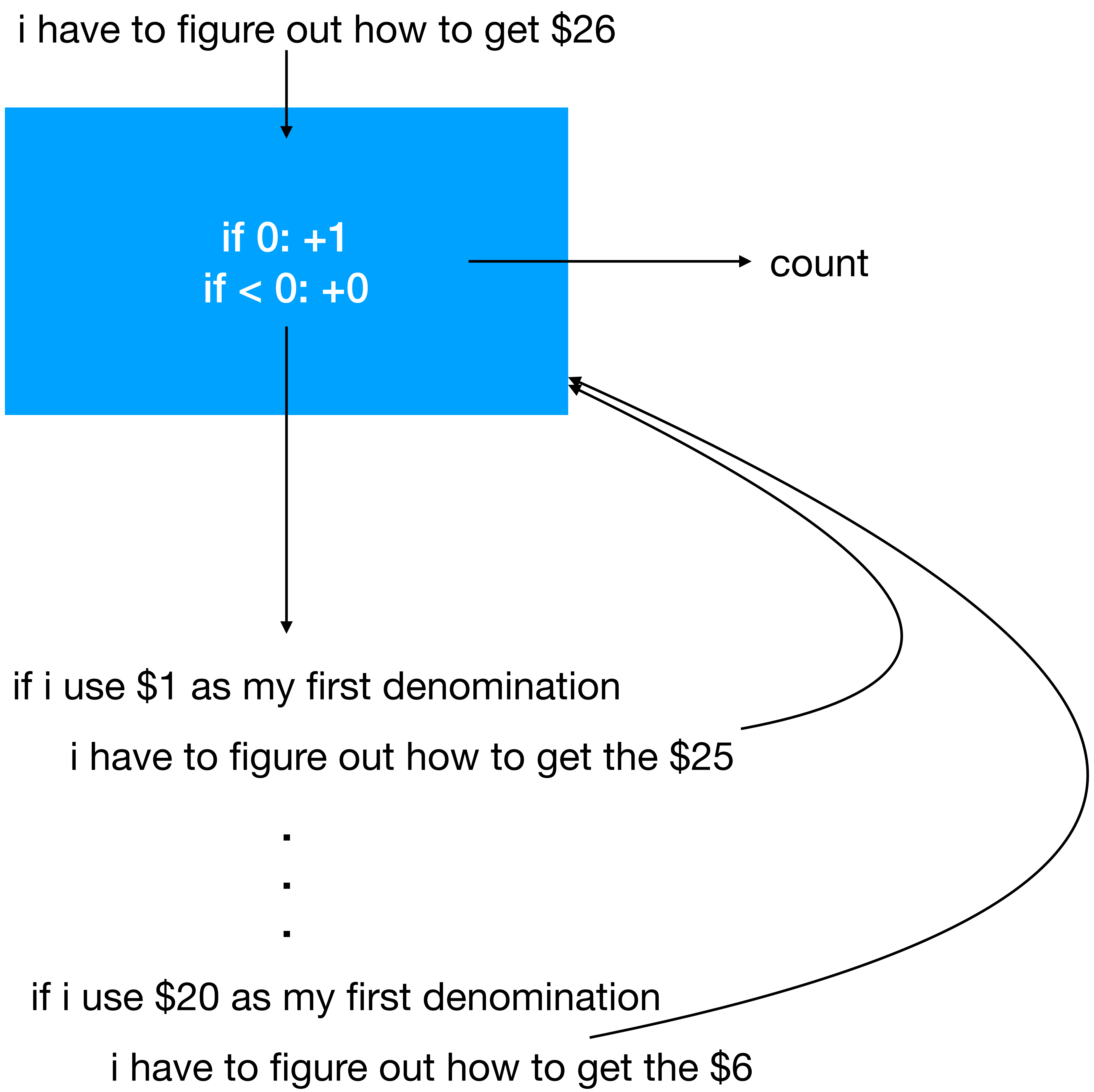
i have to figure out how to get the $6

CS61A

recursion

things *defined* by themselves

```
def count_recurse(n):
     if n < 0:
         return 0
     elif n == 0:
         return 1
     else:
         return count_recurse(n - 1)
            + count_recurse(n - 5)
            + count_recurse(n - 10)
            + count_recurse(n - 20)
```



i have to figure out how to get $26

if 0: +1
if < 0: +0

count

if i use $1 as my first denomination

i have to figure out how to get the $25

.
.
.

if i use $20 as my first denomination

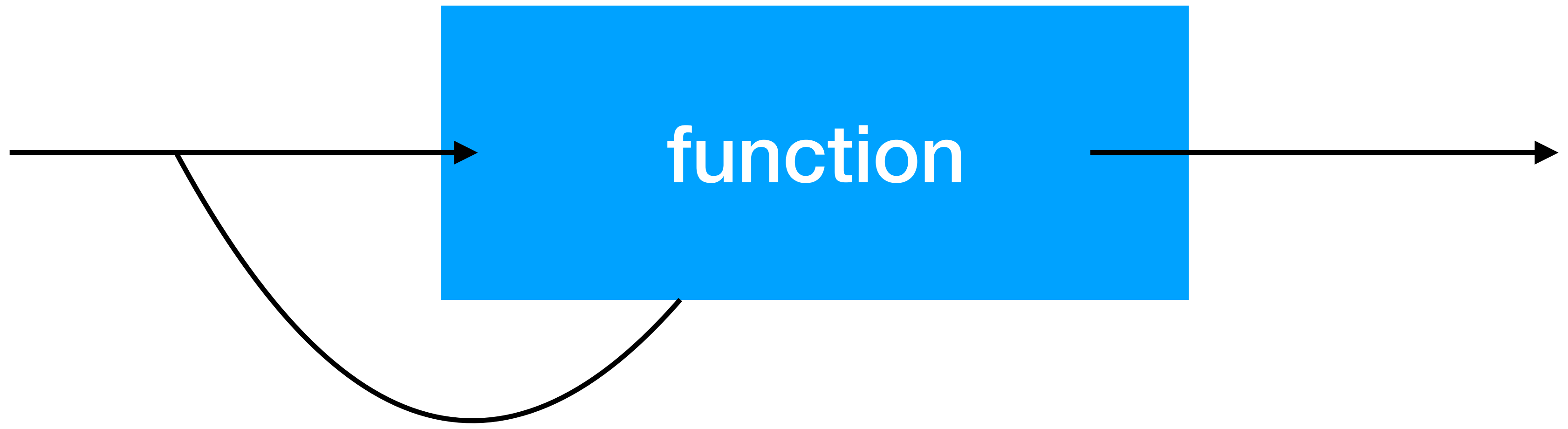
i have to figure out how to get the $6

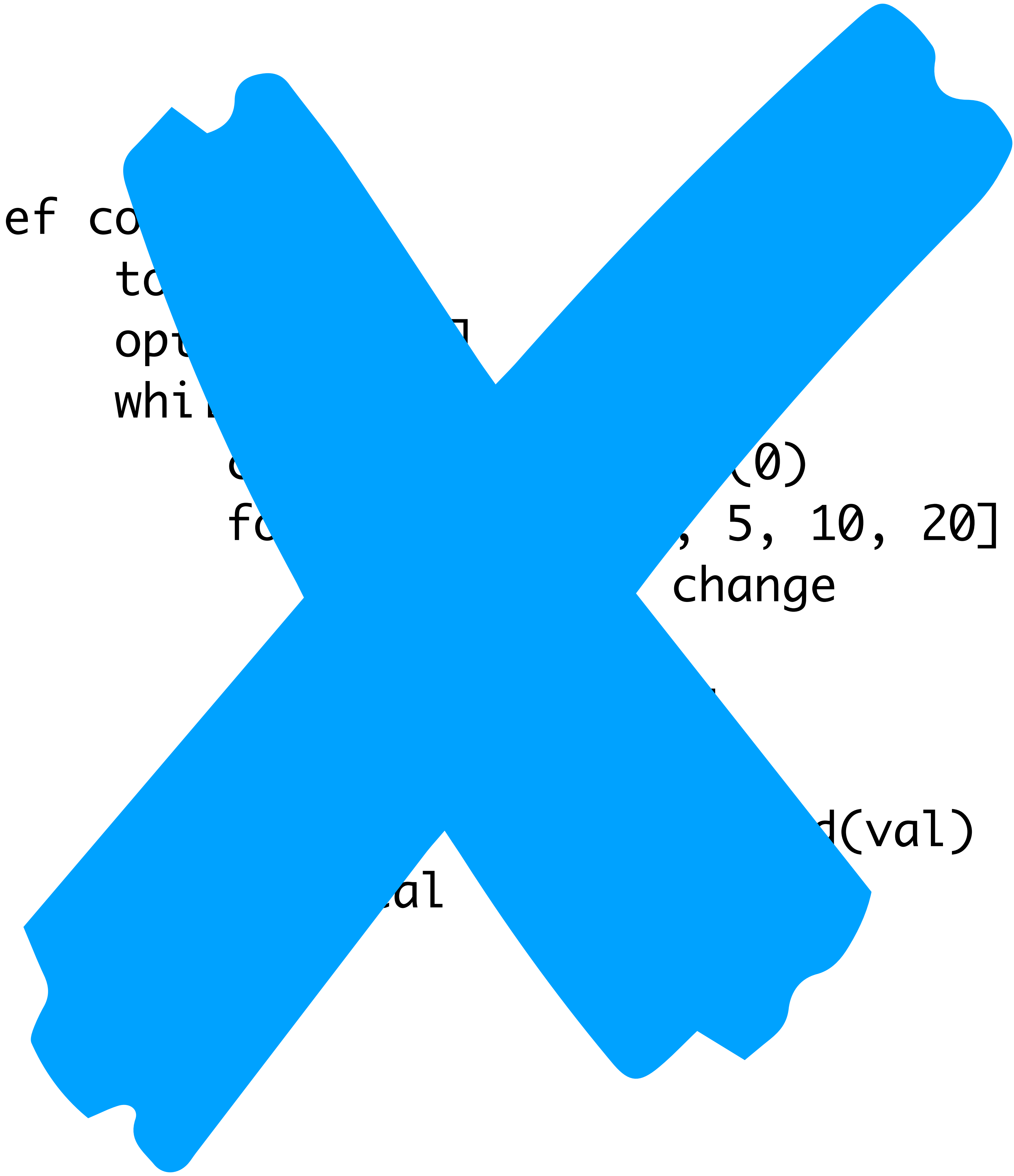
```
def count(n):
     total = 0
     options = [n]
     while len(options) > 0:
         curr = options.pop(0)
         for change in [1, 5, 10, 20]:
            val = curr - change
             if val == 0:
                 total += 1
             elif val > 0:
                 options.append(val)
     return total
```

```
def count_recurse(n):
     if n < 0:
         return 0
     elif n == 0:
         return 1
     else:
         return count_recurse(n - 1)
            + count_recurse(n - 5)
            + count_recurse(n - 10)
            + count_recurse(n - 20)
```

```
def count(n):
     total = 0
     options = [n]
     while len(options) > 0:
         curr = options.pop(0)
         for change in [1, 5, 10, 20]:
            val = curr - change
             if val == 0:
                 total += 1
             elif val > 0:
                 options.append(val)
     return total
```

```
def count_recurse(n):
     if n < 0:
         return 0
     elif n == 0:
         return 1
     else:
         return count_recurse(n - 1)
            + count_recurse(n - 5)
            + count_recurse(n - 10)
            + count_recurse(n - 20)
```





```
def co
    to
    opt              ]
    whi
        c                 (0)
        fo               , 5, 10, 20]:
                         change



                              d(val)
               al
```

```
def count_recurse(n):
    if n < 0:
        return 0
    elif n == 0:
        return 1
    else:
        return count_recurse(n - 1)
           + count_recurse(n - 5)
           + count_recurse(n - 10)
           + count_recurse(n - 20)
```

so what does this mean

we have a strategy on how to create recursive functions

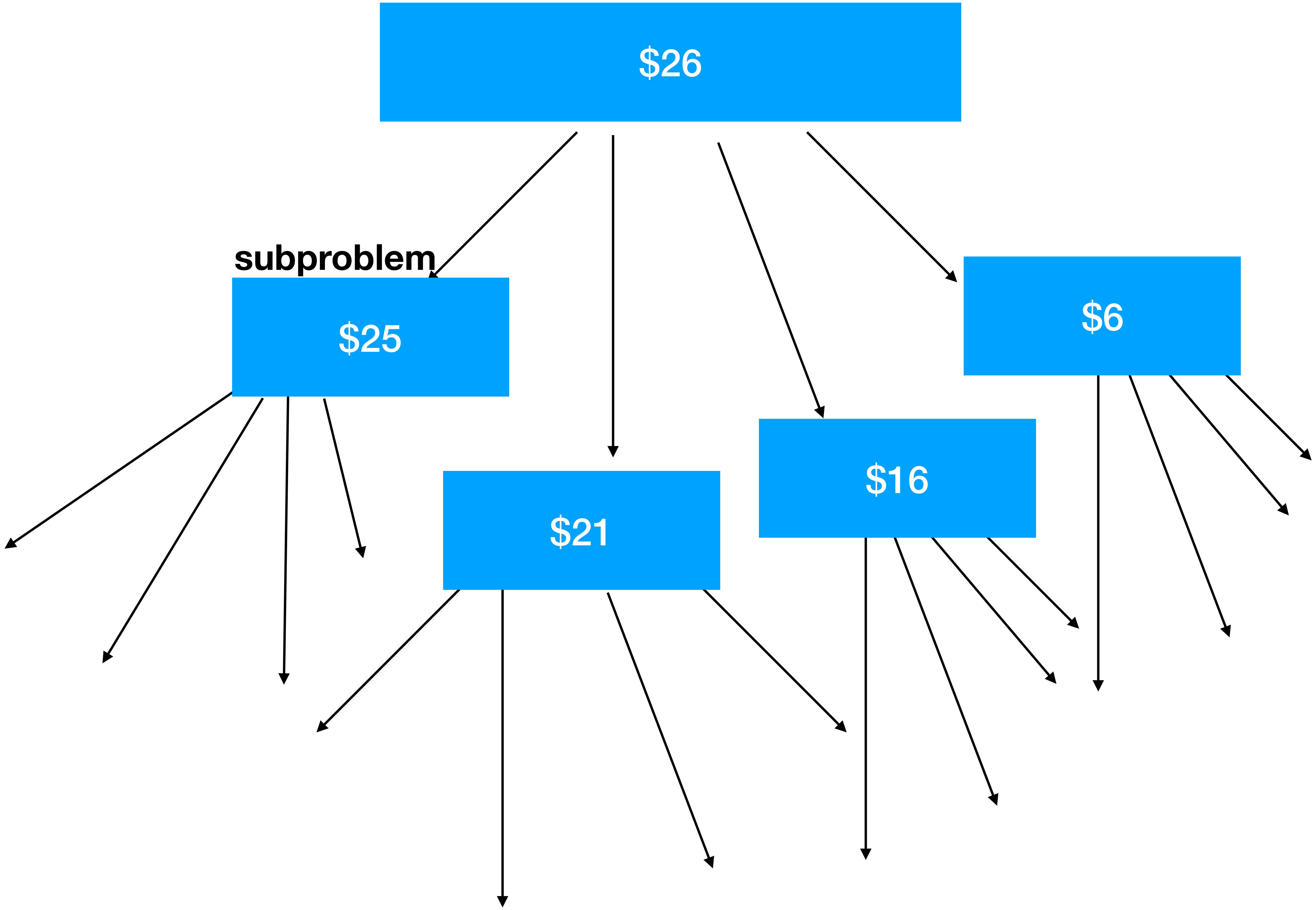
we can see that recursion isn't pointless…
at least for more complex problems

CS61A

recursion

things *defined* by themselves

```
def count_recurse(n):
     if n < 0:
         return 0
     elif n == 0:
         return 1
     else:
         return count_recurse(n - 1)
            + count_recurse(n - 5)
            + count_recurse(n - 10)
            + count_recurse(n - 20)
```



CS61A

recursion

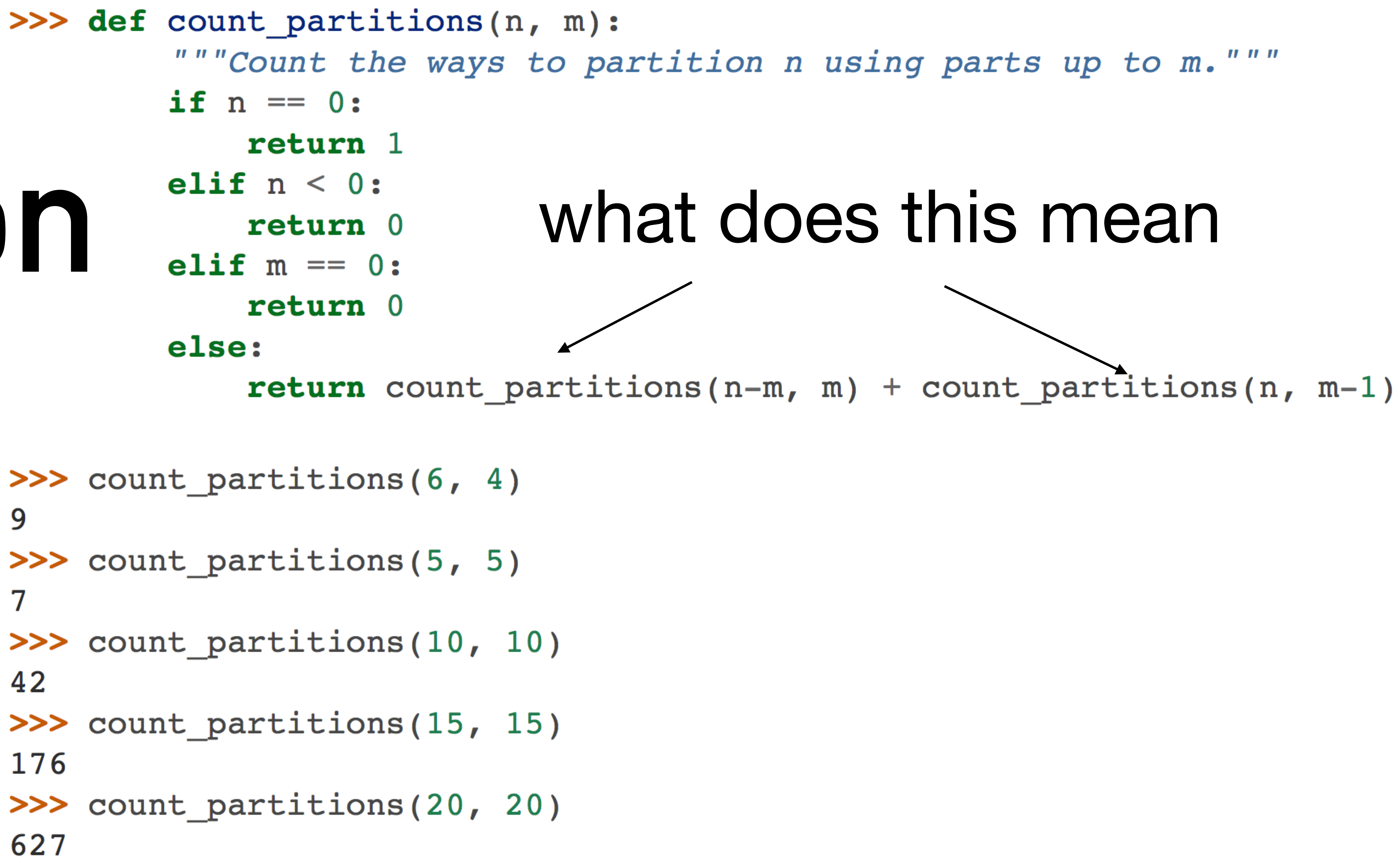
things *defined* by themselves

1. set up rules (base cases)
2. assume it works

CS61A

recursion

🤯



```
>>> def count_partitions(n, m):
        """Count the ways to partition n using parts up to m."""
        if n == 0:
            return 1
        elif n < 0:
            return 0
        elif m == 0:
            return 0
        else:
            return count_partitions(n-m, m) + count_partitions(n, m-1)

>>> count_partitions(6, 4)
9
>>> count_partitions(5, 5)
7
>>> count_partitions(10, 10)
42
>>> count_partitions(15, 15)
176
>>> count_partitions(20, 20)
627
```

what does this mean