DIY Design…

```
remAll(e, L)
```

remove all e's from L

The ♥ of CS (and CSers…)

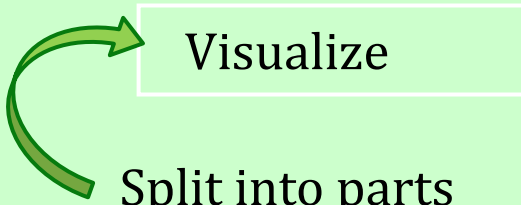
Algorithms!

Design…

`remAll(e, L)`

remove all e's from L

Top-down design



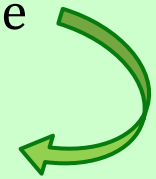
Visualize

Split into parts

Build each part

Combine

Test



```
remAll(42, [5, 7, 42, 8, 42])
```



```
[5, 7, 8]
```

```
remAll('q', 'qaqqlqqiqqiiqeqqnqs')
```



```
'aliiiens'
```

Design...

`remAll(e, L)`

remove all e's from L

Top-down design

Visualize

Split into parts

Build each part

Combine

Test

'it'

L[0] and **L[1:]** — *'the rest'*

Use it!

it

'the rest'

```
remAll(42, [5, 7, 42, 8, 42])
```

```
[5, 7, 8]
```

it

'the rest'

```
remAll('q', 'qaqqlqqiqqiiqeqqnqs')
```

```
'aliiiens'
```

Lose it!

Design...

Top-down design

Visualize

Split into parts

Build each part

Combine

Test

```
remAll(e, L)
```

remove all e's from L

Use it!

it

'the rest'

```
remAll(42, [5, 7, 42, 8, 42])
```

keep L[0]

+ remove e from the rest

```
[5, 7, 8]
```

it

'the rest'

```
remAll('q', 'qaqqlqqiqqiiqeqqnqs')
```

drop L[0]

+ remove e from the rest

```
'aliiiens'
```

Lose it!

Design…

Top-down design

Visualize

Split into parts

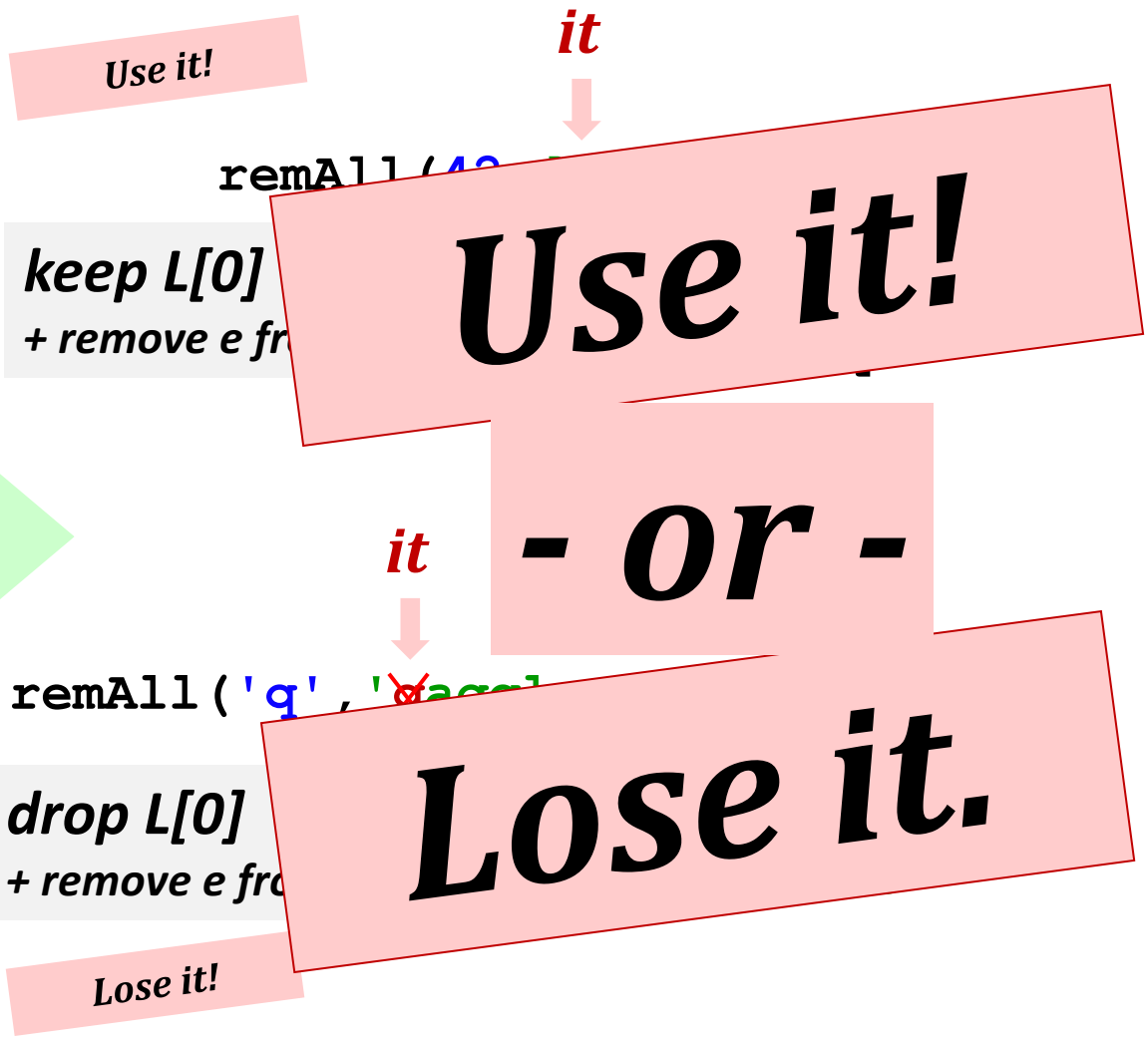
Build each part

Combine

Test

```
remAll(e, L)
```

remove all e's from L

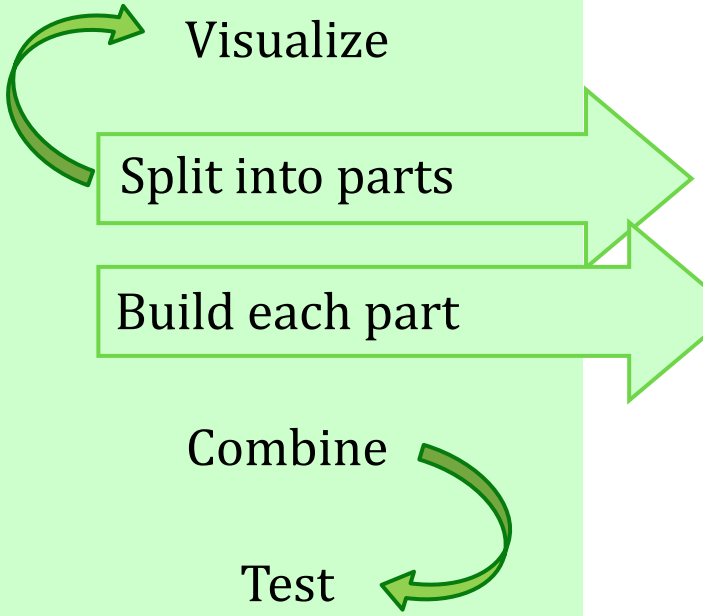


Design ~ *code*

`remAll(e, L)`

remove all e's from L

Top-down design




```
def remAll(e, L):
    if len(L) == 0:
        it
    elif L[0] != e:

    else:
```

L empty

it wasn't **e**

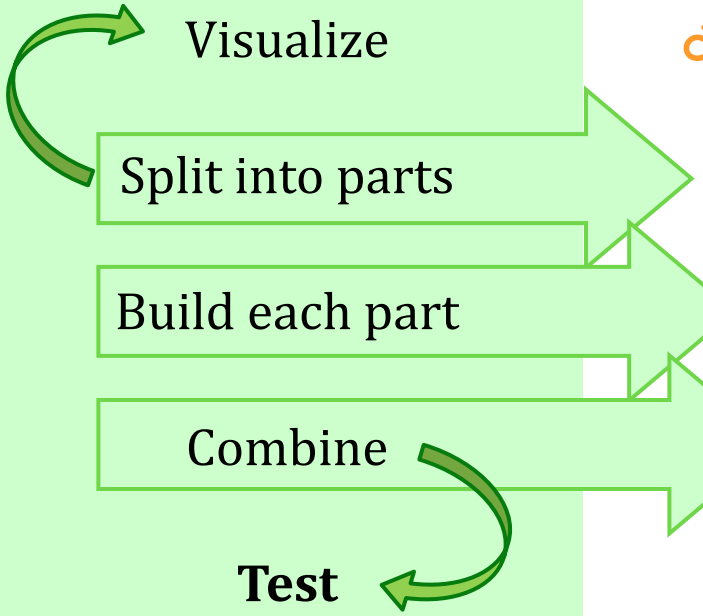
it *was* e

Design ~ *code*

`remAll(e, L)`

remove all e's from L

Top-down design



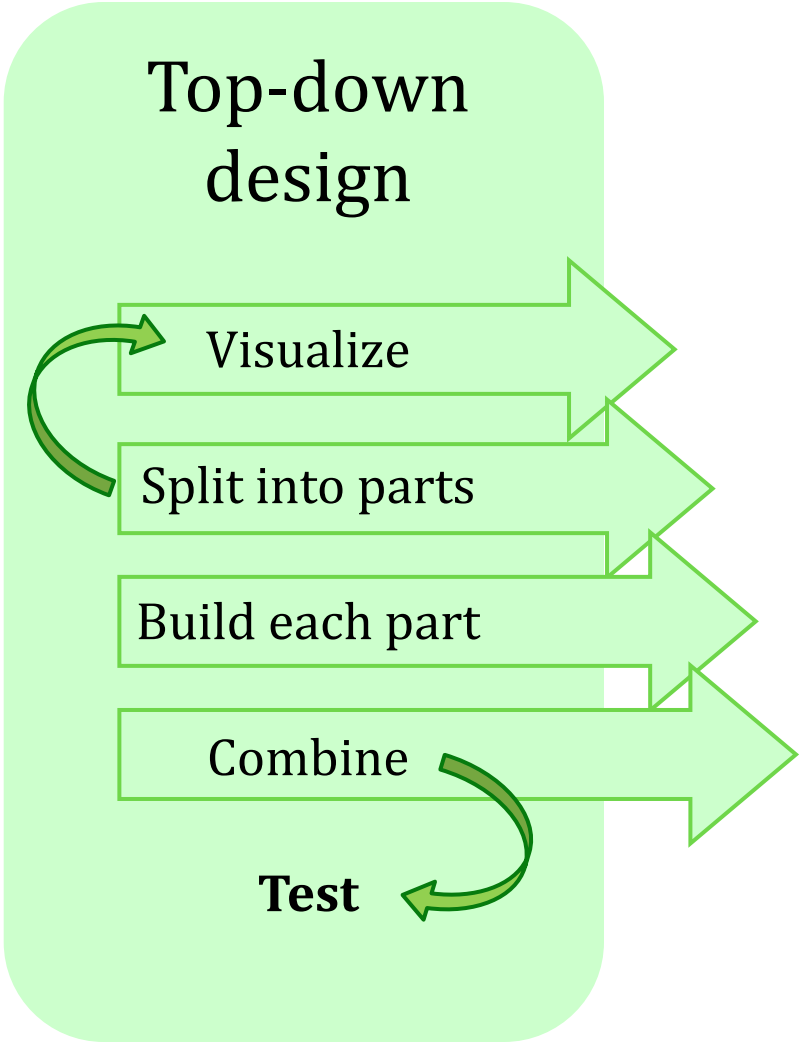

```
def remAll(e, L):
    if len(L) == 0:
        return L

    elif L[0] != e:
        return L[0:1] + remAll(e, L[1:])
    else:
        return      remAll(e, L[1:])
```

Use it!

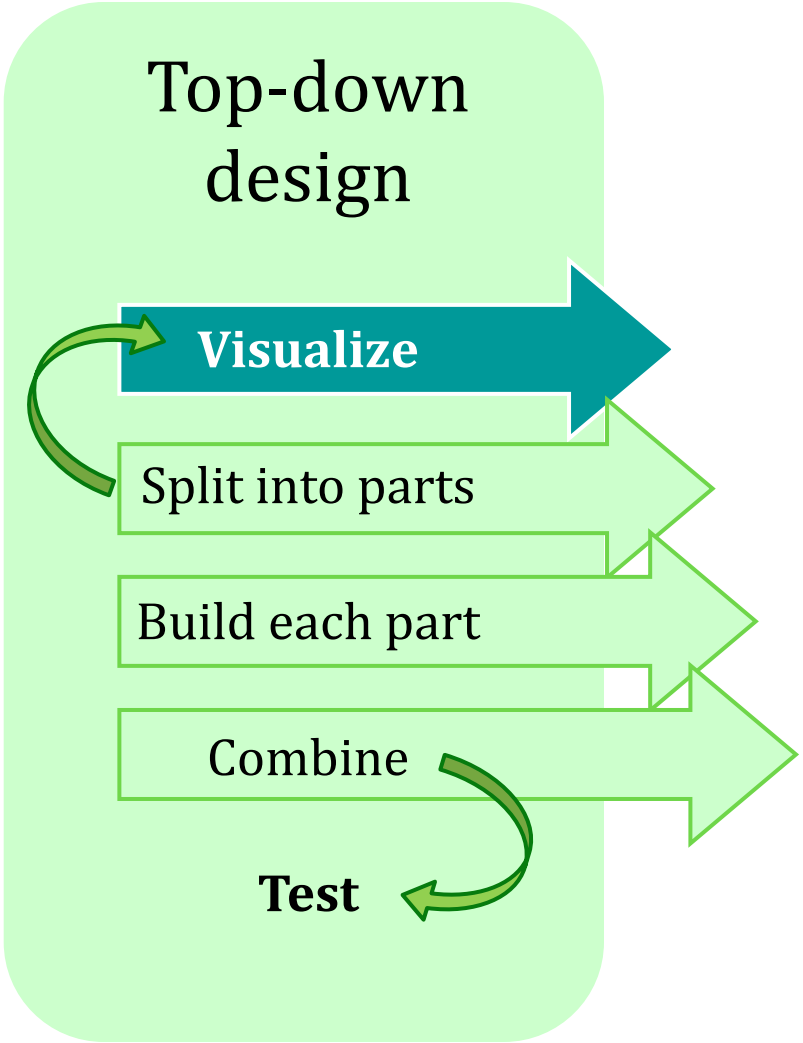
Lose it!

What's the problem?!



*Which **one** of these steps is the most important?*

What's the problem?!



understanding *what the problem demands!!*

*I want some **examples**!*



other **rem** examples...

```
remAll(8, [7,8,9,8]) ➡ [7,9]
remOne(8, [7,8,9,8]) ➡ [7,9,8]
remUpto(8,[7,8,9,8]) ➡ [9,8]
```

remAll **remOne** **remUpto**

```
remAll('d', 'coded') ➡ 'coe'
remOne('d', 'coded') ➡ 'coed'
remUpto('d','coded') ➡ 'ed'
```

remAll **remOne** **remUpto**

from **remAll** to **remOne**

Hint: remove one thing for **remOne**!

```
def remAll(e, L):
    """Removes all e's from L."""
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return L[0:1] + remAll(e, L[1:])

    else:
        return remAll(e, L[1:])
```

`remOne(8, [7,8,9,8])` ➡ `[7,9,8]`

`remOne('d', 'coded')` ➡ `'coed'`

from **remAll** to **remOne**

Hint: remove one thing for **remOne**!

```
def remOne(e, L):
    """Removes the first e from L."""
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return L[0:1] + remOne(e, L[1:])

    else:
        return L[1:]
```

`remOne(8, [7,8,9,8])` ➡ `[7,9,8]`

`remOne('d', 'coded')` ➡ `'coed'`

from **remOne** to **remUpto**

Hint: remove one ***more*** thing for **remUpto**!

```
def remOne(e, L):
    """Removes the first e from L."""
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return L[0:1] + remOne(e, L[1:])

    else:
        return L[1:]
```

`remUpto(8, [7,8,9,8])` ➡ `[9,8]`

`remUpto('d', 'coded')` ➡ `'ed'`

from **remOne** to **remUpto**

Hint: remove one ***more*** thing for **remUpto**!

```
def remOne(e, L):
    """Removes the first e from L."""
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return L[0:1] + remOne(e, L[1:])

    else:
        return L[1:]
```

`remUpto(8, [7,8,9,8])` ➡ `[9,8]`

`remUpto('d', 'coded')` ➡ `'ed'`

from **remOne** to **remUpto**

Hint: remove one ***more*** thing for **remUpto**!

```
def remUpto(e, L):
    """Removes the first e from L."""
    if  len(L) == 0:
        return L

    elif  L[0] != e:
        return remUpto(e, L[1:])

    else:
        return L[1:]
```

`remUpto(8, [7,8,9,8])` ➡ `[9,8]`

`remUpto('d', 'coded')` ➡ `'ed'`

`reduce`-ing with Python...

```
from functools import reduce

def add(x, y):
   """ returns x + y """
   return x + y

>>> reduce(add, [1, 2, 3, 4])
10
```

