More Recursion!

Computer Science 111
Boston University

Vahid Azadeh Ranjbar, Ph.D.

Designing a Recursive Function

1. Start by programming the base case(s).
   - *What instance(s) of this problem can I solve directly (without looking at anything smaller)?*

2. Find the recursive substructure.
   - *How could I use the solution to **any smaller version** of the problem to solve the overall problem?*

   I. Make a recursive call!

   II. Trace your function before designing the rest of the function after recursive call.

   III. Do one step and build your solution on the result of the recursive call
   - use ***concrete cases*** to figure out what you need to do

A Recursive Function for Counting Vowels

```
def num_vowels(s):
    """ returns the number of vowels in s
        input s: a string of lowercase letters
    """
    # we'll design this together!
```

- Examples of how it should work:

```
>>> num_vowels('compute')
3
>>> num_vowels('now')
1
```

- The `in` operator will be helpful:

```
>>> 'fun' in 'function'
True
>>> 'i' in 'team'
False
```

Design Questions for `num_vowels()`

(base case) When can I determine the # of vowels in `s` *without* looking at a smaller string? an empty string has 0 vowels!

Design Questions for `num_vowels()`

```
def num_vowels(s):
    if s == '':
        return 0
```

Base case

Design Questions for `num_vowels()`

(base case)	When can I determine the # of vowels in `s` *without* looking at a smaller string? an empty string has 0 vowels!
(recursive substructure)	How could I use the solution to ***anything smaller*** than `s` to determine the solution to `s`?

I. Make a recursive call!
II. Trace your function before designing the rest of the function after recursive call.
III. Do one step and build your solution on the result of the recursive call
   - use ***concrete cases*** to figure out what you need to do

Design Questions for `num_vowels()`

(base case) When can I determine the # of vowels in s *without* looking at a smaller string? an empty string has 0 vowels!

(recursive substructure) How could I use the solution to ***anything smaller*** than s to determine the solution to s?

I. Make a recursive call!

It is highly recommended to follow these steps:

1. Call the function
2. Play around the parameter of the function to make the problem smaller (**converging to the base case**)
3. Assign the function to a variable called "**storage variable**". We will use this **storage variable** in the next step "Do one step"

Design Questions for `num_vowels()`

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
```

3- Store in a variable

2- Play around the parameter

1- Call the function

Design Questions for `num_vowels()`

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
```

II. Trace your function before designing the rest of the function after recursive call.

- Make sure that it converges to the base case.
- Do not go for "**do one step!**" before tracing your function.

Design Questions for `num_vowels()`

(base case) When can I determine the # of vowels in `s` *without* looking at a smaller string? an empty string has 0 vowels!

(recursive substructure) How could I use the solution to ***anything smaller*** than `s` to determine the solution to `s`?

III. Do one step and build your solution on the result of the recursive call

- use **concrete cases** to figure out what you need to do

Design Questions for `num_vowels()`

(base case) When can I determine the # of vowels in s *without* looking at a smaller string? an empty string has 0 vowels!

(recursive substructure) How could I use the solution to ***anything smaller*** than s to determine the solution to s?

```
total # of vowels
= 1 + (# in covered)
```

```
total # of vowels
= 0 + (# in covered)
```

The recursive call gives us `(# in covered)`!!!

Think about one character at a time!

How Many Lines of This Function Have a Bug?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[0:])
        if s[0] in 'aeiou':
            return 1
        else:
            return 0
```

After you make your group vote, fix the function!

A. 0
B. 1
C. 2
D. 3
E. more than 3

How Many Lines of This Function Have a Bug?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[0:])
        if s[0] in 'aeiou':
            return 1
        else:
            return 0
```

A. 0
B. 1
C. 2
D. **3**
E. more than 3

How Many Lines of This Function Have a Bug?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1
        else:
            return 0
```

To converge to the base case

A. 0
B. 1
C. 2
D. **3**
E. more than 3

How Many Lines of This Function Have a Bug?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1 + num_rest
        else:
            return 0 + num_rest
```

```
total # of vowels
= 1 + (# in covered)
```

```
total # of vowels
= 0 + (# in covered)
```

A. 0
B. 1
C. 2
D. **3**
E. more than 3

How Many Lines of This Function Have a Bug?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1 + num_rest
        else:
            return 0 + num_rest
```

Base case

Recursive Substructure

A. 0
B. 1
C. 2
D. **3**
E. more than 3

How Many Lines of This Function Have a Bug?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])   ← Call the function!
        if s[0] in 'aeiou':
            return 1 + num_rest         ← Do one step!
        else:
            return 0 + num_rest
```

A. 0

B. 1

C. 2

D. **3**

E. more than 3

How about looking for s[-1] in this problem?

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[-1] in 'aeiou':
            return 1 + num_rest
        else:
            return 0 + num_rest
```

- It does not work because we removed index s[0] each time in the step "call the function".

Consider this initial call…

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1 + num_rest
        else:
            return 0 + num_rest
```

```
                                num_vowels('ate')
                                  s = 'ate'
num_vowels('ate')
```

Consider this initial call…

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1 + num_rest
        else:
            return 0 + num_rest
```

```
                                num_vowels('ate')
                                  s = 'ate'
num_vowels('ate')
```

What value is eventually assigned to num_rest? (i.e., what does the recursive call return?)

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            return 1 + num_rest
        else:
            return 0 + num_rest
```

```
num_vowels('ate')
```

```
num_vowels('ate')
  s = 'ate'
  num_rest = ??
```

What value is eventually assigned to num_rest? (i.e., what does the recursive call return?)

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        ...
```

```
num_vowels('ate')
```

A. 0

B. 1

C. 2

D. 3

```
num_vowels('ate')
  s = 'ate'
  num_rest = ??
```

What value is eventually assigned to num_rest? (i.e., what does the recursive call return?)

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        ...


num_vowels('ate')
```

A. 0

B. 1

C. 2

D. 3

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

What value is eventually assigned to num_rest? (i.e., what does the recursive call return?)

```
def num_vowels(s):
    if s == '':
        return 0
    else:
        num_rest = num_vowels(s[1:])
        ...


num_vowels('ate')
```

A. 0

B. 1

C. 2

D. 3

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
           = 1
```

How recursion works...

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

How recursion works...

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

How recursion works...

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

How recursion works...

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

```
num_vowels('e')
  s = 'e'
```

How recursion works...

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

num_vowels('ate')
```
s = 'ate'
num_rest = num_vowels('te')
```

num_vowels('te')
```
s = 'te'
num_rest = num_vowels('e')
```

num_vowels('e')
```
s = 'e'
num_rest = num_vowels('')
```

How recursion works...

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

num_vowels('ate')
```
s = 'ate'
num_rest = num_vowels('te')
```

num_vowels('te')
```
s = 'te'
num_rest = num_vowels('e')
```

num_vowels('e')
```
s = 'e'
num_rest = num_vowels('')
```

num_vowels('')
```
s = ''
```

4 different stack frames, each with its own set of variables!

How recursion works...

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

```
num_vowels('e')
  s = 'e'
  num_rest = num_vowels('')
```

```
num_vowels('')
  s = ''
  base case!
  return 0
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

4 different stack frames, each with its own set of variables!

How recursion works...

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

```
num_vowels('e')
  s = 'e'
  num_rest = 0
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

How recursion works...

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

```
num_vowels('e')
  s = 'e'  s[0] -> 'e'
  num_rest = 0
```

How recursion works...

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

```
num_vowels('e')
  s = 'e'  s[0] -> 'e'
  num_rest = 0
  return 1 + 0 = 1
```

num_rest

The final result
gets built up
on the way back
from the base case!

How recursion works...

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = num_vowels('e')
```

```
num_vowels('e')
  s = 'e'   s[0] -> 'e'
  num_rest = 0
  return 1 + 0 = 1
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

The final result gets built up ***on the way back*** from the base case!

How recursion works...

```
num_vowels('ate')
  s = 'ate'
  num_rest = num_vowels('te')
```

```
num_vowels('te')
  s = 'te'
  num_rest = 1
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

The final result gets built up ***on the way back*** from the base case!

How recursion works...

```
num_vowels('ate')
   s = 'ate'
   num_rest = num_vowels('te')
```

```
num_vowels('te')
   s = 'te'   s[0] -> 't'
   num_rest = 1
   return 0 + 1 = 1
```

num_rest

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

The final result gets built up ***on the way back*** from the base case!

How recursion works...

```
num_vowels('ate')
   s = 'ate'
   num_rest = num_vowels('te')
```

```
num_vowels('te')
   s = 'te'   s[0] -> 't'
   num_rest = 1
   return 0 + 1 = 1
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

The final result gets built up ***on the way back*** from the base case!

How recursion works...

```
num_vowels('ate')
   s = 'ate'
   num_rest = 1
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

The final result
gets built up
on the way back
from the base case!

How recursion works...

```
num_vowels('ate')
   s = 'ate'  s[0] -> 'a'
   num_rest = 1
   return 1 + 1 = 2
                ↑
                 num_rest
```

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

The final result
gets built up
on the way back
from the base case!

How recursion works...

```
num_vowels('ate')
  s = 'ate'  s[0] -> 'a'
  num_rest = 1
  return 1 + 1 = 2
```

final result: 2

```
def num_vowels(s):
  if s == '':
     return 0
  else:
     num_rest = num_vowels(s[1:])
     if s[0] in 'aeiou':
         return 1 + num_rest
     else:
         return 0 + num_rest
```

Debugging Technique: Adding Temporary prints

```
def num_vowels(s):
    print('beginning call for', s)
    if s == '':
        print('base case returns 0')
        return 0
    else:
        num_rest = num_vowels(s[1:])
        if s[0] in 'aeiou':
            print('call for', s, 'returns', 1 + num_rest)
            return 1 + num_rest
        else:
            print('call for', s, 'returns', 0 + num_rest)
            return 0 + num_rest
```

From the pre-lecture quiz: What is the output of this program?

```python
def myst(s):
    if len(s) <= 1:
        return s
    else:
        return s[-1] + myst(s[:-1]) + s[-1]

print(myst('bar'))
```

A. `rabar`

B. `rabbar`

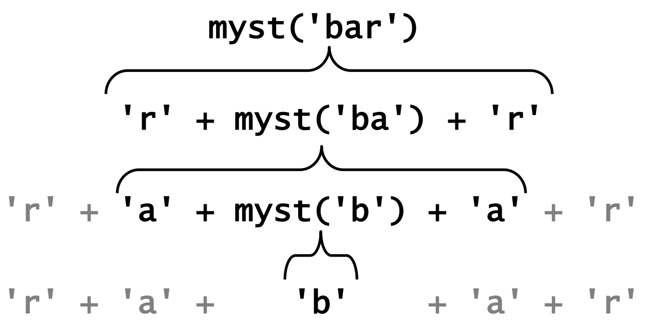
C. `barab`

D. `barrab`

E. none of these

How recursion works...

```python
def myst(s):
    if len(s) <= 1:
        return s
    else:
        return s[-1] + myst(s[:-1]) + s[-1]
```



```
          myst('bar')
'r' + myst('ba') + 'r'
'r' + 'a' + myst('b') + 'a' + 'r'
'r' + 'a' +    'b'    + 'a' + 'r'
```

How recursion works...

```
def myst(s):
    if len(s) <= 1:
        return s
    else:
        return s[-1] + myst(s[:-1]) + s[-1]
```

$$\overbrace{\texttt{'r' + myst('ba') + 'r'}}^{\texttt{myst('bar')}}$$

$$\texttt{'r'} + \overbrace{\texttt{'a' + 'b' + 'a'}} + \texttt{'r'}$$

How recursion works...

```
def myst(s):
    if len(s) <= 1:
        return s
    else:
        return s[-1] + myst(s[:-1]) + s[-1]
```

$$\overbrace{\texttt{'r' + 'aba' + 'r'}}^{\texttt{myst('bar')}}$$

How recursion works...

```
def myst(s):
    if len(s) <= 1:
        return s
    else:
        return s[-1] + myst(s[:-1]) + s[-1]
```

myst('bar')

result: **'rabar'**

What is the output of this program?

```
def myst(s):
    if len(s) <= 1:
        return s
    else:
        return s[-1] + myst(s[:-1]) + s[-1]

print(myst('bar'))
```

A. **rabar**

B. rabbar

C. barab

D. barrab

E. none of these