Recursion
How do you reverse a string?

(instructions in English, not Python)
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

string
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

s t r i n g

S
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

string

s t r i n g

t s
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

strotn
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

string
irts
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

string

nirts
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

string

gnirts
How do you reverse a string?
(instructions in English, not Python)

"Scan the string from left to right, copying characters as you go. Add each copied character to the left of the string you're building."

**Easy to code this in Python:**

```python
def revstring(t):
    s = ''
    for c in t:
        s = c + s
    return s
```
A different approach

"Reverse the substring consisting of everything but the last character, then attach the last character at the left."
A different approach

"Reverse the substring consisting of everything but the last character, then attach the last character at the left."

s t r i n g
A different approach

"Reverse the substring consisting of everything but the last character, then attach the last character at the left."

string

nirts
A different approach

"Reverse the substring consisting of everything but the last character, then attach the last character at the left."

string

gnirts
A different approach

"Reverse the substring consisting of everything but the last character, then attach the last character at the left."

It looks kind of fishy, because it appears to define string reversal in terms of... string reversal.
A different approach

"Reverse the substring consisting of everything but the last character, then attach the last character at the left."

It looks kind of fishy, because it appears to define string reversal in terms of... string reversal.

Can you do that? What if we tried it in Python?
It doesn't work, but the problem is not so much the self-reference, as the fact that the definition is incorrect if the string you're reversing is empty, and there is no 'last character'.
It doesn't work, but the problem is not so much the self-reference, as the fact that the definition is incorrect if the string you're reversing is empty, and there is no 'last character'.

"If the string is not empty, reverse the substring consisting of everything but the last character, then attach the last character at the left.

If the string is empty, it is its own reversal."
It doesn't work, but the problem is not so much the self-reference, as the fact that the definition is incorrect if the string you're reversing is empty, and there is no 'last character'.

"If the string is not empty, reverse the substring consisting of everything but the last character, then attach the last character at the left.

If the string is empty, it is its own reversal."

This is what it looks like in Python:

```python
def revstring3(t):
    if len(t) == 0:
        return t
    else:
        return t[-1] + revstring3(t[:-1])
```
It doesn't work, but the problem is not so much the self-reference, as the fact that the definition is incorrect if the string you're reversing is empty, and there is no 'last character'.

"If the string is not empty, reverse the substring consisting of everything but the last character, then attach the last character at the left.

If the string is empty, it is its own reversal."

This is what it looks like in Python:

```python
def revstring3(t):
    if len(t)==0:
        return t
    else:
        return t[-1]+revstring3(t[:-1])
```

And it works!
The definition is not circular---it does not define the process of string reversal in terms of itself...
The definition is not *circular*---it does not define the process of string reversal in terms of itself...

It defines a process in terms of *smaller instances* of the process.
The definition is not *circular*---it does not define the process of string reversal in terms of itself...

It defines a process in terms of *smaller instances* of the process.

Usually this requires a two-part definition, with a special clause for the smallest instances.

The definition is *recursive*. 
Another example: factorial
Another example: factorial

\( n! \) ("n factorial") is the product of the first \( n \) positive integers.
Another example: factorial

n! ("n factorial") is the product of the first n positive integers.

1! = 1  
2! = 1 \times 2 = 2  
3! = 1 \times 2 \times 3 = 6  
4! = 1 \times 2 \times 3 \times 4 = 24
Another example: factorial

\[ n! \text{ ("n factorial") is the product of the first } n \text{ positive integers.} \]

1! = 1
2! = 1 \times 2 = 2
3! = 1 \times 2 \times 3 = 6
4! = 1 \times 2 \times 3 \times 4 = 24
0! = 1
Recursive definition of $n!$

If $n = 0$, then $n! = 1$

otherwise, $n! = n \times (n-1)!$
Recursive definition of n!

If \( n = 0 \), then \( n! = 1 \)

otherwise, \( n! = n \times (n-1)! \)

You can translate it directly into Python:

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n*factorial(n-1)
```
1. What if we used this as our recursive definition of reversing a string: If the string is empty, it is its own reversal. Otherwise, reverse the substring consisting of everything but the first character, and then adjoin the first character to the right. Write the Python code for this and test it.

2. Strings of length 1 are also their own reversal. What if we made the smallest case for string reversal strings of length 1 instead of empty strings. Does it still work?

3. What happens if you call revstring3 with a list instead of a string as an argument?

4. What happens if you call factorial with -1 as an argument? With 1.5 as an argument?