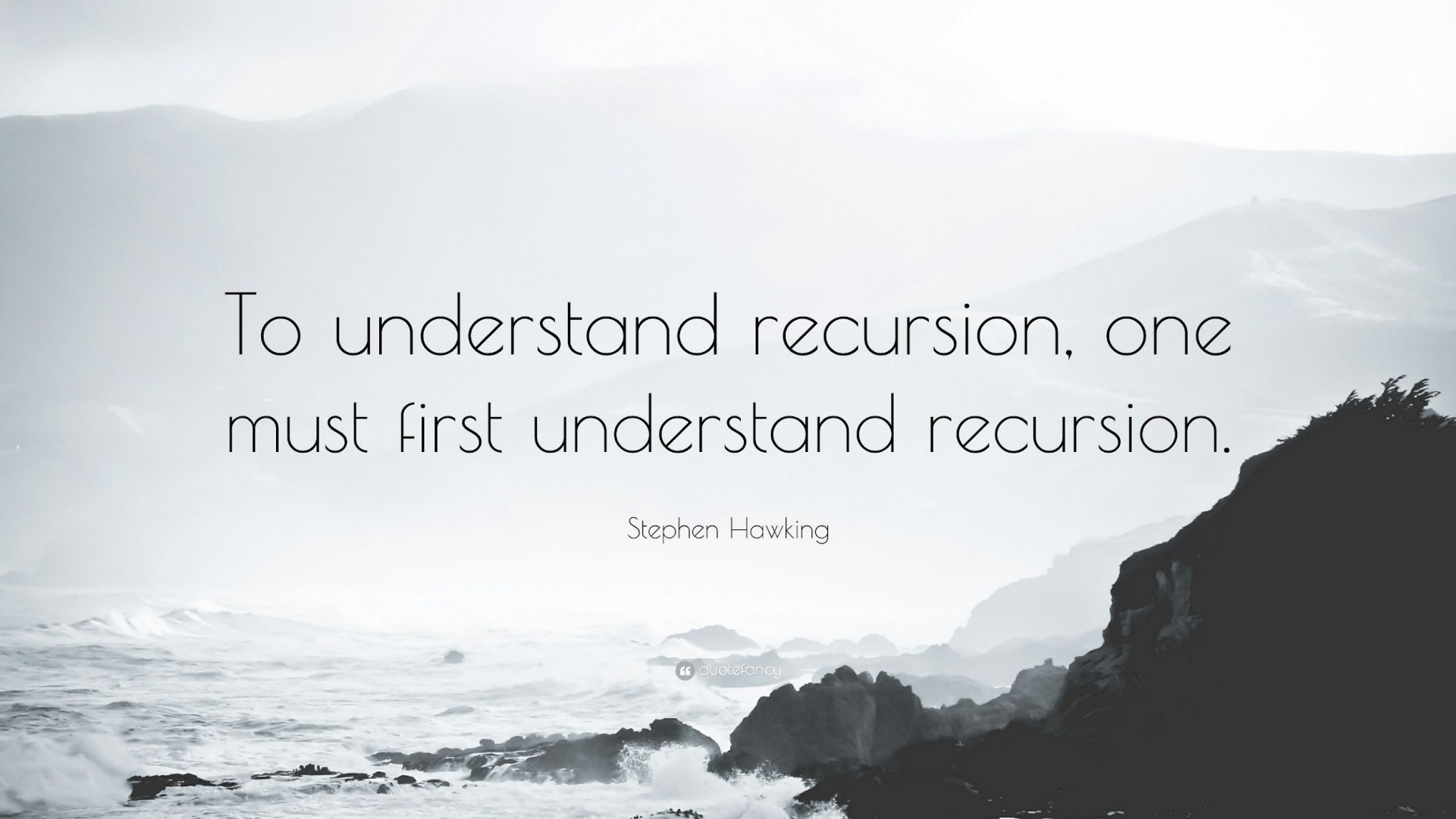

To understand recursion, one must first understand recursion.
Stephen Hawking
quotefancy

stack

- just that: function is called, it goes on top of the stack

```
def somefunction(x):
    i = 0
    print(i)
    i = someotherfunction(x)
    print(i)

def someotherfunction(x)
    return x
```

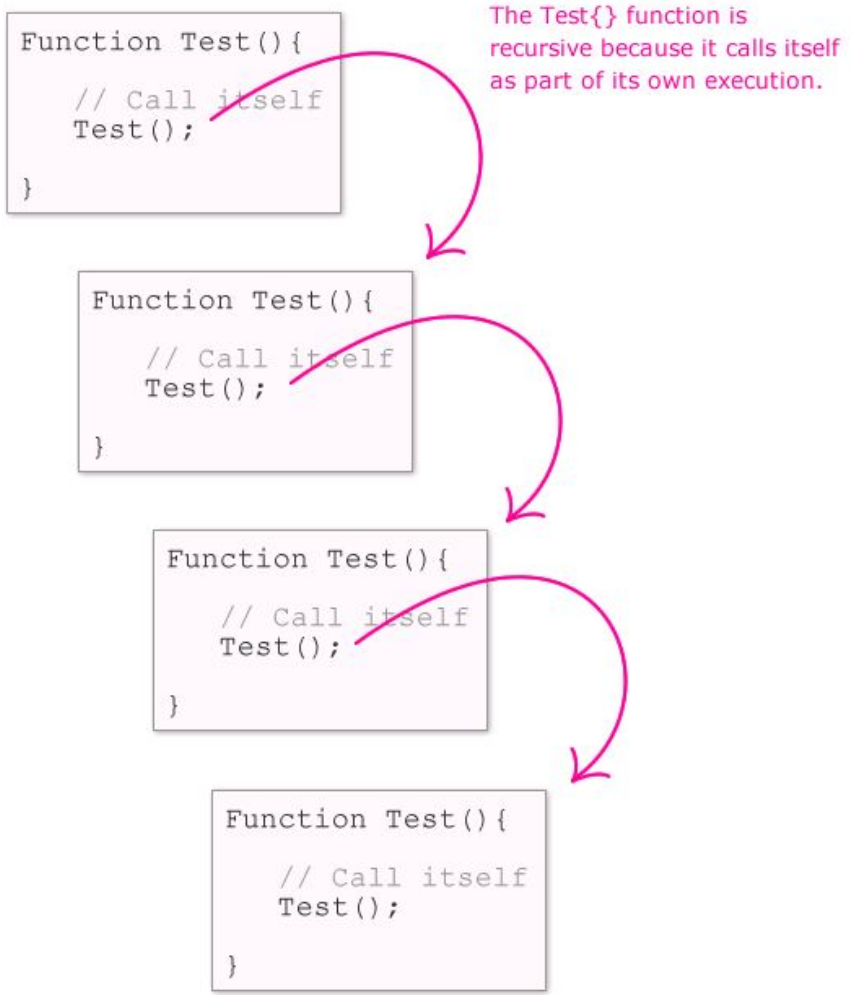
- stack is executed top to bottom, or last in first out

recursion

1. Base Case (i.e., when to stop)
2. Work toward Base Case
3. Recursive Call (i.e., call ourselves)

recursion

- stack overflow



example

```
// Prints the given number of stars on the console.
// Assumes n >= 1.
void printStars(int n) {
    if (n == 1) {
        // n == 1, base case
        cout << "*";
    } else {
        // n > 1,  recursive case
        cout << "*";          // print one star myself
        printStars(n - 1);   // recursion to do the rest
    }
}
```

example

```
def navigate(inElement):
    if inElement.nodeType == xml.dom.Node.ELEMENT_NODE:
        print "ELEMENT: ",inElement.localName
        for (name, value) in inElement.attributes.items():
            print '    ATTR:  Name: %s  Value: %s' % (name, value)
        for e in inElement.childNodes:
            navigate(e)
    elif inElement.nodeType == xml.dom.Node.TEXT_NODE:
        print "CONTENT: ",inElement.data
```

exercise

- no points given if submitted solution didn’t run
- define a template with parameters in XSLT:

```
<xsl:template name="dumpDebugData">
   <xsl:param name="elementToDump" />
   <xsl:for-each select="$elementToDump/@*">
      <xsl:text>&#10;</xsl:text> <!-- newline char -->
      <xsl:value-of select="name()" /> :
      <xsl:value-of select="." />
   </xsl:for-each>
</xsl:template>
```

exercise

- to call:

```
<xsl:call-template name="dumpDebugData">
   <xsl:with-param name="elementToDump" select="some/xpath" />
</xsl:call-template>
```