Example: badpassword II: the recursening

Goal: Replace all e’s with 3’s

What is the base case?

What is the recursion case?

Trace how this program works with the input “feet”

```
"""
Replace all "e"'s with "3"'s
"""

def badpassword(text):
    passwd = ""
    for i in range(len(text)):
       if text[i] == "e":
          passwd += "3"
       else:
          passwd += text[i]
    return passwd

def rbadpassword(text):
    if len(text) == 0:
       return ""

    if text[0] == "e":
       return "3" + rbadpassword(text[1:])
    else:
       return text[0] + rbadpassword(text[1:])

def main():
    print(badpassword("fee"))
    print(rbadpassword("fee"))
    print(badpassword("elephant"))
    print(rbadpassword("elephant"))


main()
```

Example: badpassword II: the recursening

rbadpassword(“feet”)

= “f” + **rbadpassword**(“eet”)
= “f” + [“3” + **rbadpassword**(“et”)]
= “f” + [“3” + [“3” + **rbadpassword**(“t”)]]
= “f” + [“3” + [“3” + [“t” + **rbadpassword**(“”)]]]
= “f” + [“3” + [“3” + [“t” + “”]]]
= “f” + [“3” + [“3” + “t”]]
= “f” + [“3” + “3t”]
= “f” + “33t”
= “f33t”

Example: replace all letters with “X”

Trace how this program works with the input “cat”

```
"""
Replace all letters with X
"""

def replaceX(text):
    passwd = ""
    for i in range(len(text)):
       passwd += "X"
    return passwd

def rreplaceX(text):
    if len(text) == 0:
       return ""

    return "X" + rreplaceX(text[1:])

def main():
    print(replaceX("fee"))
    print(rreplaceX("fee"))
    print(replaceX("elephant"))
    print(rreplaceX("elephant"))


main()

```

Example: replace all letters with “X”

rreplaceX(“cat”)

= “X” + **rreplaceX**(“at”)
= “X” + [“X” + **rreplaceX**(“t”)]
= “X” + [“X” + [“X” + **rreplaceX**(“”)]]
= “X” + [“X” + [“X” + “”]]
= “X” + [“X” + “X”]
= “X” + [“XX”]
= “XXX”

Example: Check if all elements are even

Trace the input when L = [2,4,3] and when L = [2,4,0]

```
"""
Write iterative and recursive functions to check if a list has only even numbers
"""

import random

def allEven(L):
    for i in range(len(L)):
       if L[i] % 2 == 1:
          return False
    return True

def rallEven(L):
    if len(L) == 0:
       return True

    if len(L) == 1:
       return (L[0] % 2) == 0

    return L[0] % 2 == 0 and rallEven(L[1:])

def main():
    L = [3, 5, 1, 2, 4]
    assert(allEven(L) == False)
    assert(rallEven(L) == False)

    L = [-2, 4, 0, 2, 4]
    assert(rallEven(L) == True)
    assert(rallEven(L) == True)

    assert(allEven([]) == True)
    assert(rallEven([]) == True)

main()
```

Example: Check if all elements are even

rallEven([2,4,3])

= (2 % 2 == 0) and **rallEven**([4,3])
= (2 % 2 == 0) and [(4 % 2 == 0) and **rallEven**([3])]
= (2 % 2 == 0) and [(4 % 2 == 0) and [(3 % 2 == 0)]
= True and [True and False]
= True and [False]
= False

Example: Check if all elements are even

rallEven([2,4,0])

= (2 % 2 == 0) and **rallEven**([4,0])
= (2 % 2 == 0) and [(4 % 2 == 0) and **rallEven**([0])]
= (2 % 2 == 0) and [(4 % 2 == 0) and [(0 % 2 == 0)]
= True and [True and True]
= True and [True]
= True

Example: length of a string

Trace how this program works with the input “coffee”

```
"""
Write iterative and recursive functions to the length of a string
"""

def strlen(text):
    total = 0
    for i in range(len(text)):
        total += 1
    return total

def rstrlen(text):
    if text == "":
        return 0

    total = 1 + rstrlen(text[1:])
    return total

def main():
    assert(strlen("abba") == 4)
    assert(rstrlen("cat") == 3)
    assert(rstrlen("hello") == 5)

main()
```

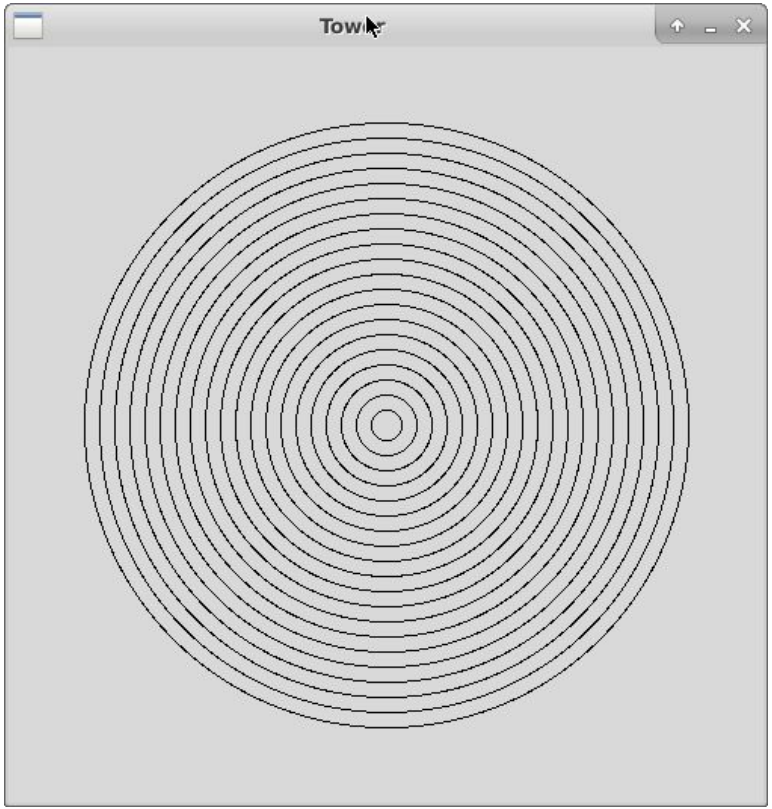
Example: length of a string

strlen(“coffee”)

= 1 + **strlen**(“offee”)
= 1 + 1 + **strlen**(“ffee”)
= 1 + 1 + 1 + **strlen**(“fee”)
= 1 + 1 + 1 + 1 + **strlen**(“ee”)
= 1 + 1 + 1 + 1 + 1 + **strlen**(“e”)
= 1 + 1 + 1 + 1 + 1 + 1 + **strlen**(“”)
= 1 + 1 + 1 + 1 + 1 + 1 + 0
= 1 + 1 + 1 + 1 + 1 + 1
= 1 + 1 + 1 + 1 + 2
= 1 + 1 + 1 + 3
= 1 + 1 + 4
= 1 + 5
= 6

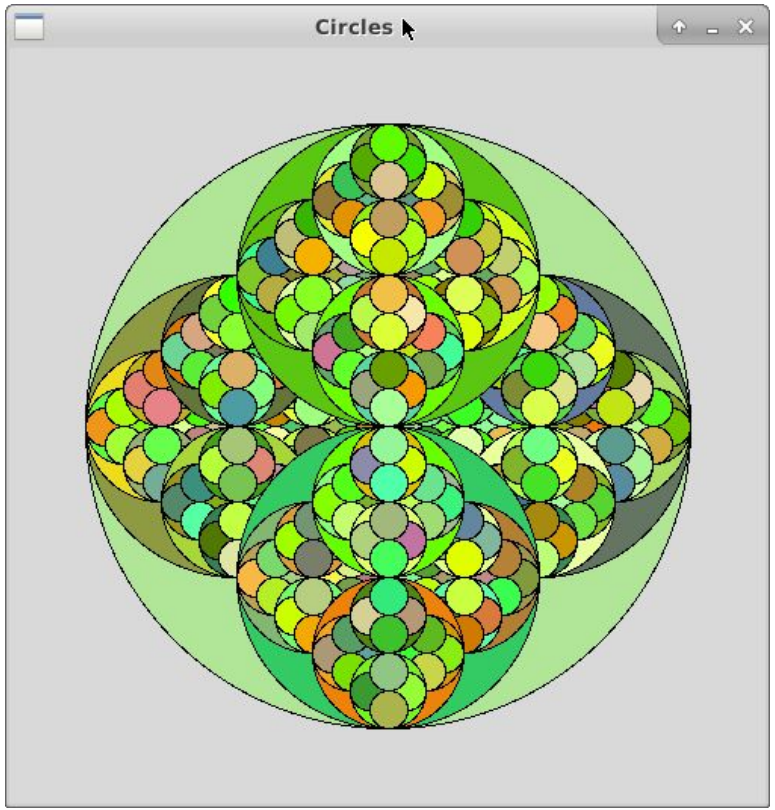
Recursive bullseye

```
"""
Draw a bullseye using both iterative and recursive functions
Challenge: Draw the circles with different colors
Challenge: Draw the circles with alternating colors

"""

from graphics import *

def bullseye(win, center, radius):
    while radius > 10:
        c = Circle(center, radius)
        c.draw(win)
        radius = radius - 10

def rbullseye(win, center, radius):
    if radius < 10:
        return

    c = Circle(center, radius)
    c.draw(win)
    rbullseye(win, center, radius - 10)

def main():
    win = GraphWin("Tower", 500, 500)
    rbullseye(win, Point(250,250), 200)
    win.getMouse()

main()
```



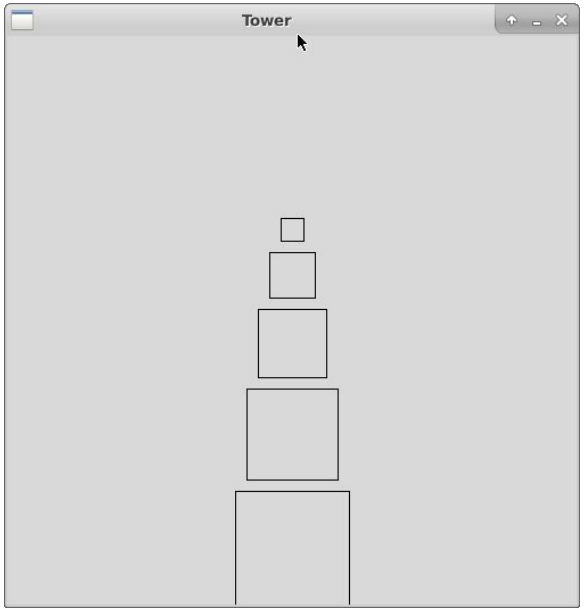
Recursive stained glass window

```
def rbullseye(win, color, center, radius):
    if radius < 10:
       return

    # jitter color
    jitter = [0,0,0]
    for i in range(len(color)):
        jitter[i] = color[i] + random.randrange(-100,100)
        jitter[i] = min(max(0, jitter[i]), 255)

    c = Circle(center, radius)
    c.setFill(color_rgb(jitter))
    c.draw(win)

    p1 = Point(center.getX() - radius/2, center.getY())
    rbullseye(win, color, p1, radius/2)

    p2 = Point(center.getX() + radius/2, center.getY())
    rbullseye(win, color, p2, radius/2)

    p3 = Point(center.getX(), center.getY() - radius/2)
    rbullseye(win, color, p3, radius/2)

    p4 = Point(center.getX(), center.getY() + radius/2)
    rbullseye(win, color, p4, radius/2)

def main():
    win = GraphWin("Circles", 500, 500)
    red = random.randrange(255)
    green = random.randrange(255)
    blue = random.randrange(255)
    rbullseye(win, [red,blue,green], Point(250,250), 200)
    win.getMouse()

main()
```



Example: Recursive tower

```
18  def rtower(win, center, size):
19      if size < 10:
20          return
21
22      px = center.getX()
23      py = center.getY()
24      tl = Point(px - size, py - size)
25      br = Point(px + size, py + size)
26      rectangle = Rectangle(tl, br)
27      rectangle.draw(win)
28      py = py - 2*size
29      size = size - 10
30      rtower(win, Point(px, py), size)
```



Recursive binary search: rbinsearch(x, L)

Base cases:

- if len(L) is 0, return False
- if L[mid] == x, return True

Recursion case:

- if x < L[mid], look in the left sublist
- if x > L[mid], look in the right sublist

recursive binary search

What is the function stack for

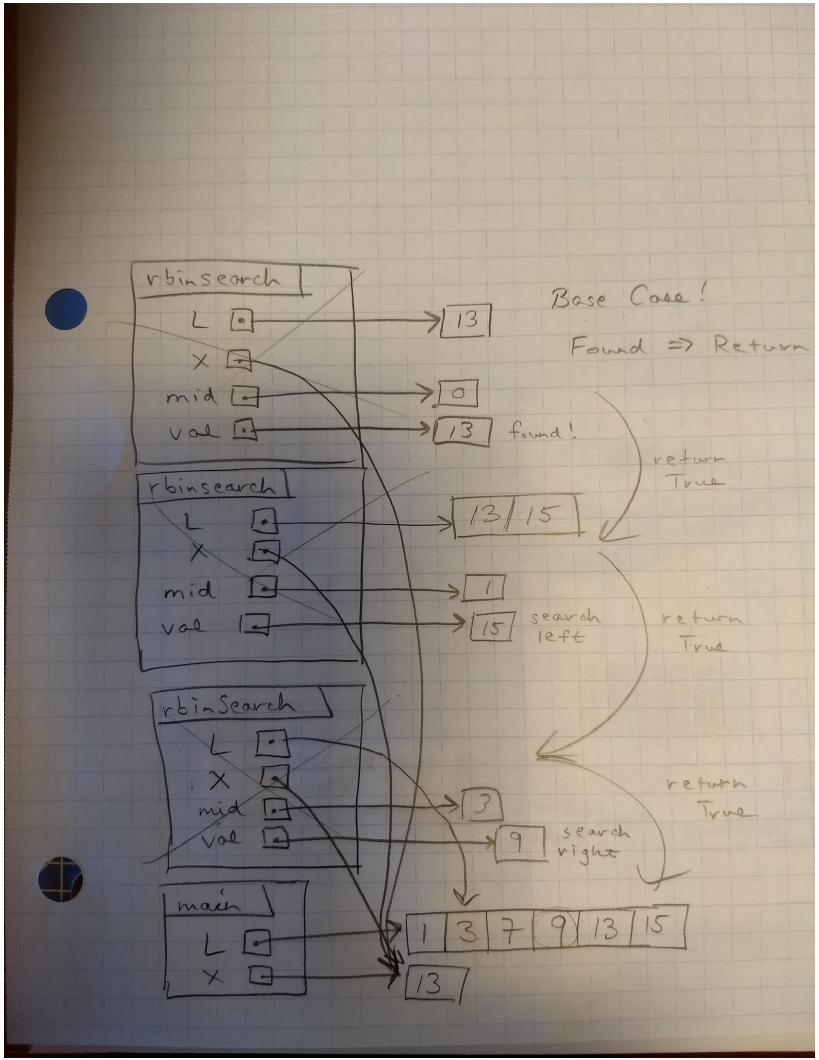
L = [1,3,7,9,13, 15] and x = 13?

What is the function stack for

L = [1,3,7,9,13, 15] and x = 2?

```
15
16  def rbinsearch(x, L):
17      if len(L) == 0:
18          return False
19
20      mid = int(len(L)/2)
21      val = L[mid]
22      if x < val:
23          return rbinsearch(x, L[:mid])
24      elif x > val:
25          return rbinsearch(x, L[mid+1:])
26      else:
27          return True
28
29
30  if __name__ == '__main__':
31
32      L = list(range(1,10,2))
33      print(L)
34
35      for i in range(10):
36          print(i, " in L?", binsearch(i, L))
37          print(i, " in L?", rbinsearch(i, L))
```

L = [1,3,7,9,13, 15] and x = 13



L = [1,3,7,9,13, 15] and x = 2