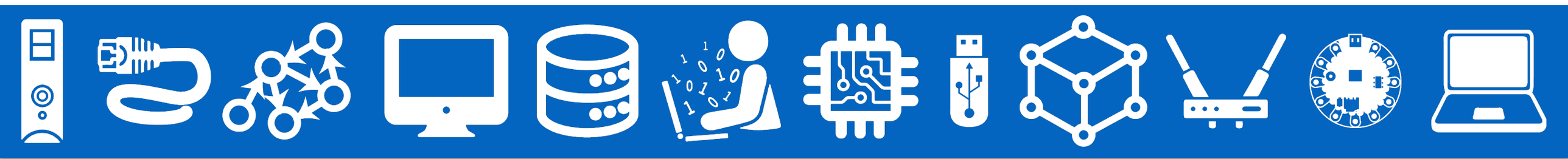
CS134:

Graphical Recursion



Announcements & Logistics

- **Lab 7** today and tomorrow: focuses on recursion
  - Please write/print the **pre lab** before you come to lab
    - We will be collecting it! The prelab is an *individual* assignment
- **HW 6** due @ 10 pm
  - We made a mistake on one question — Glow is now fixed. The question no longer counts against your quiz score.

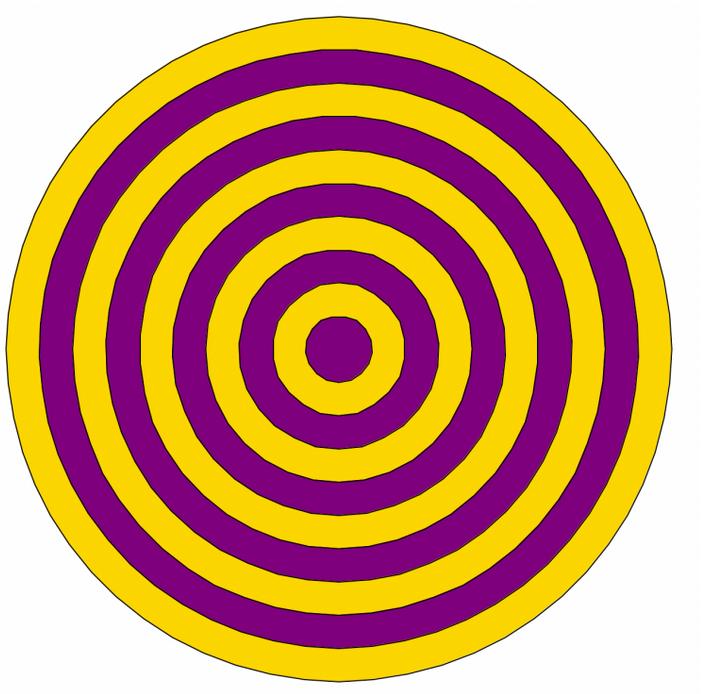
Do You Have Any Questions?

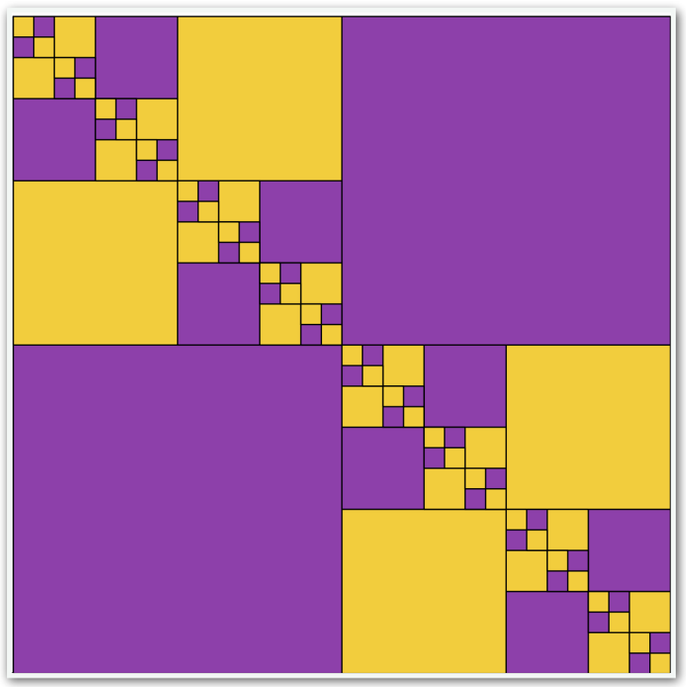
Last Time: Recursive Approach to Problem Solving

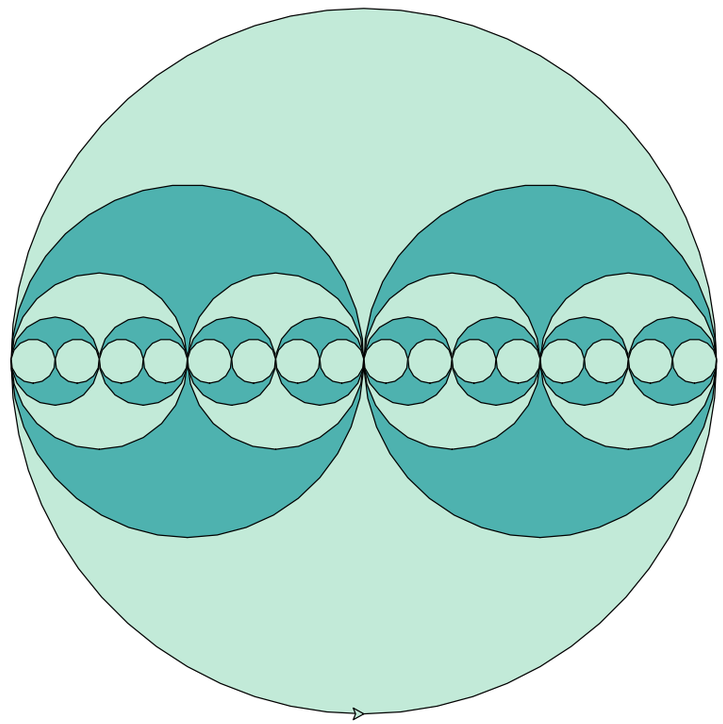
- A recursive function is a function **that calls itself**
- A recursive approach to problem solving has two main parts:
  - **Base case(s).** When the problem is **so small**, we solve it directly, without having to reduce it any further
  - **Recursive step.** Does the following things:
    - Performs an action that contributes to the solution
    - **Reduces** the problem to a smaller version of the same problem, and calls the function on this **smaller subproblem**
- The recursive step is a form of "wishful thinking" (also called the inductive hypothesis)

Today's Plan

- Introduction to Turtle
- Graphical recursion examples
- Understanding function **invariants** and why they matter when doing recursion







The Turtle Module

- Turtle is a **graphics module** first introduced in the 1960s by computer scientists Seymour Papert, Wally Feurzig, and Cynthia Solomon.
- It uses a programmable cursor — fondly referred to as the "turtle" — to draw on a Cartesian plane (x and y axis.)

```
pen down
```

Turtle In Python

- **`turtle`** is available as a built-in module in Python. See the Python turtle module API for details.
- Basic turtle commands:

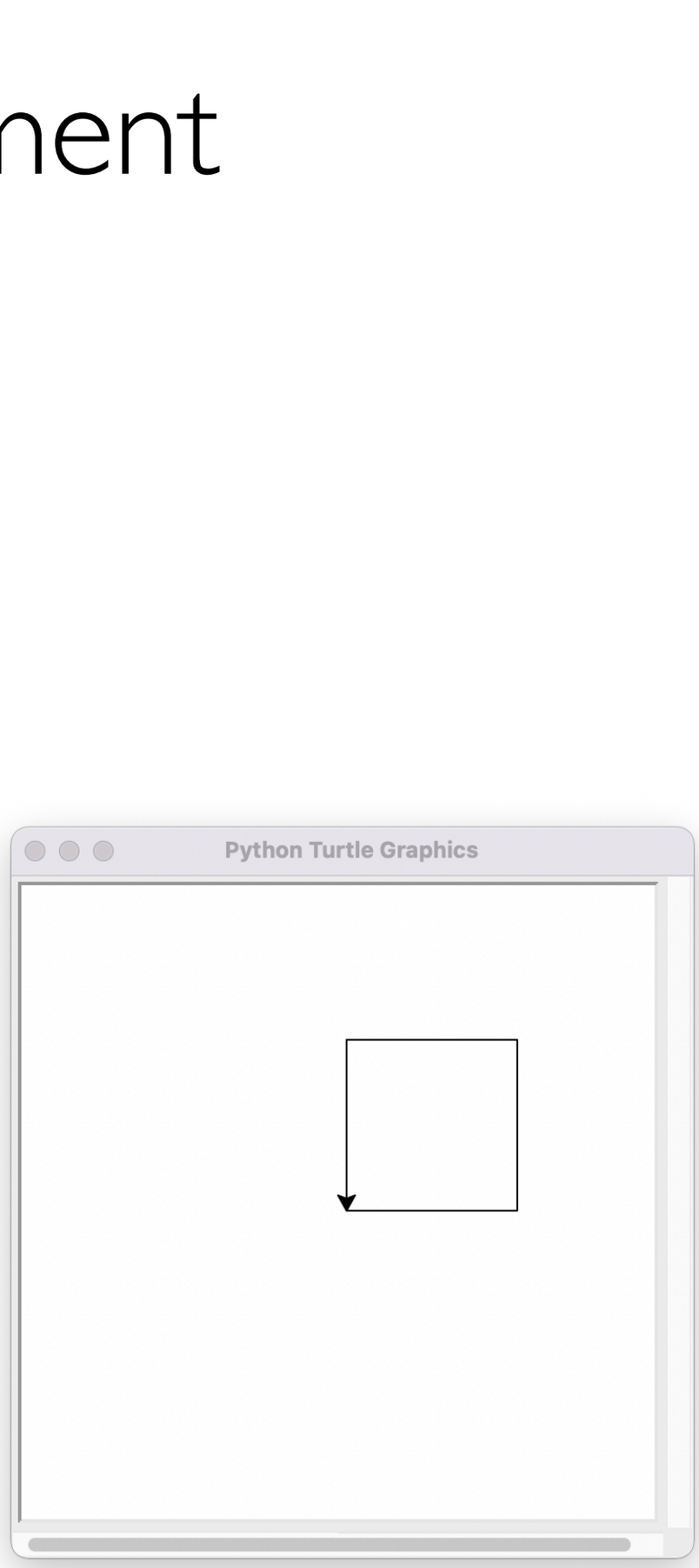
Use `from turtle import *` to use these commands

`fd(dist)`	turtle moves forward by dist
`bk(dist)`	turtle moves backward by dist
`lt(angle)`	turtle turns left angle degrees
`rt(angle)`	turtle turns right angle degrees
`up()`	(pen up) turtle raises pen in belly
`down()`	(pen down) turtle lowers pen from belly
`shape(shp)`	sets the turtle's shape to shp
`speed(spd)`	sets the turtle's speed 1-10 (slow-fast). 0 skips animation.
`home()`	turtle returns to (0,0) (center of screen)
`clear()`	delete turtle drawings; no change to turtle's state
`reset()`	delete turtle drawings; reset turtle's state
`setup(width, height)`	create a turtle window of given width and height

Basic Turtle Movement

- forward(dist) or fd(dist),
  left(angle) or lt(angle),
  right(angle) or rt(angle),
  backward(dist) or bk(dist)

```
# set up a 400x400 turtle window
setup(400, 400)
reset()

fd(100) # move the turtle forward 100 pixels

lt(90) # turn the turtle 90 degrees to the left

fd(100) # move forward another 100 pixels

# complete a square
lt(90)
fd(100)
lt(90)
fd(100)
done()
```

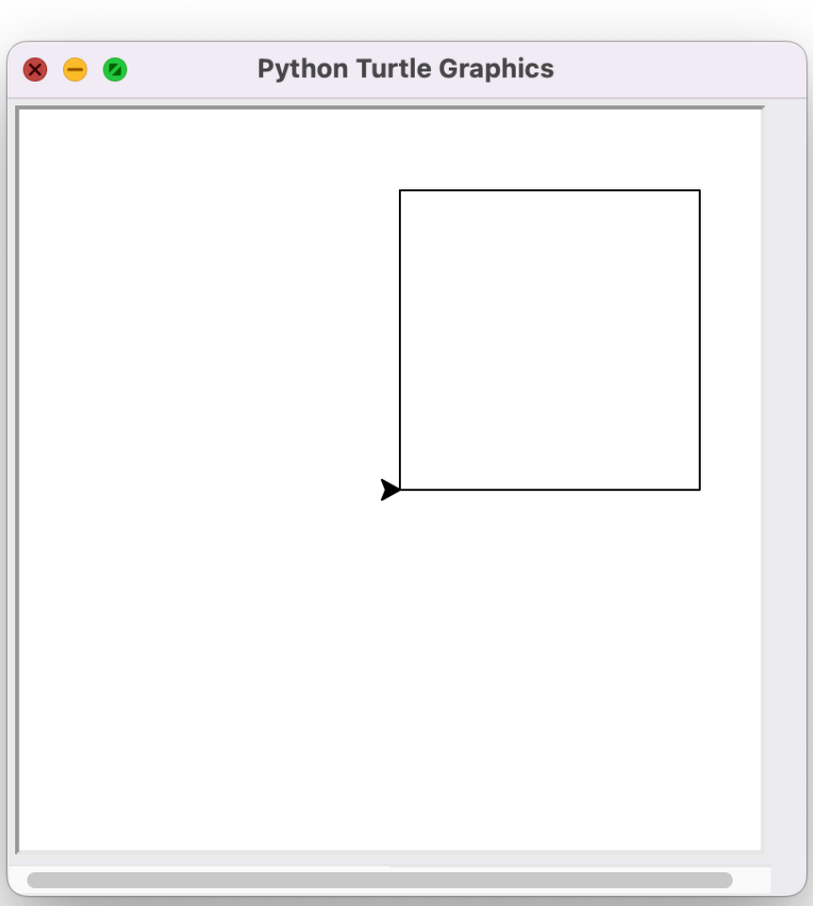


Drawing Basic Shapes With Turtle

- We can write functions that use turtle commands to draw shapes.
- For example, here's a function that draws a square of the desired size

```
def draw_square(length):
    # a loop that runs 4 times
    # and draws each side of the square
    for i in range(4):
        fd(length)
        lt(90)
    done()
```

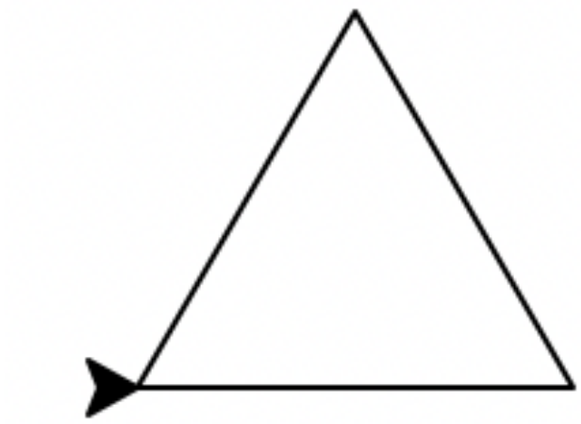
```
setup(400, 400)
reset()
draw_square(150)
```



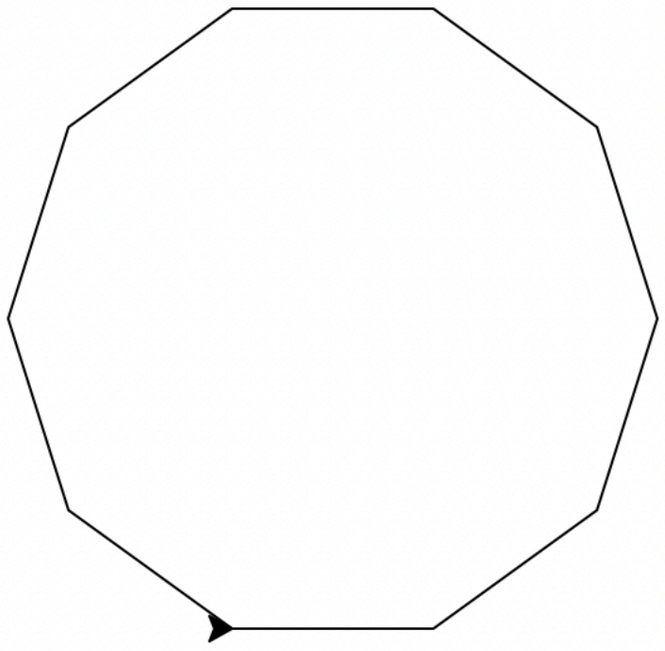
Drawing Basic Shapes With Turtle

- How about drawing polygons with different numbers of sides?

```
def draw_polygon(length, num_sides):
    for i in range(num_sides):
        fd(length)
        lt(360/num_sides)
    done()
```



`draw_polygon(80, 3)`

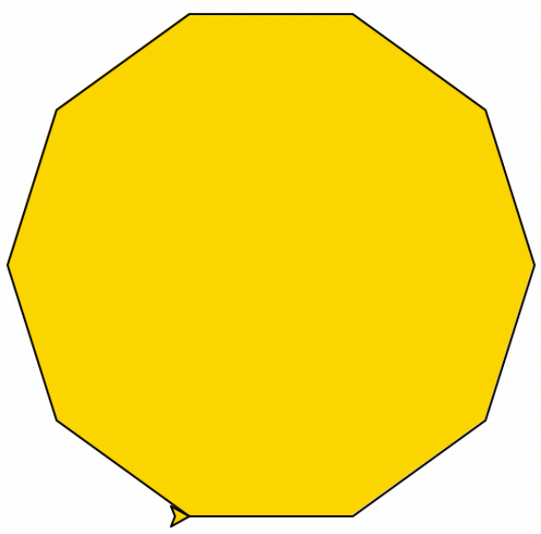


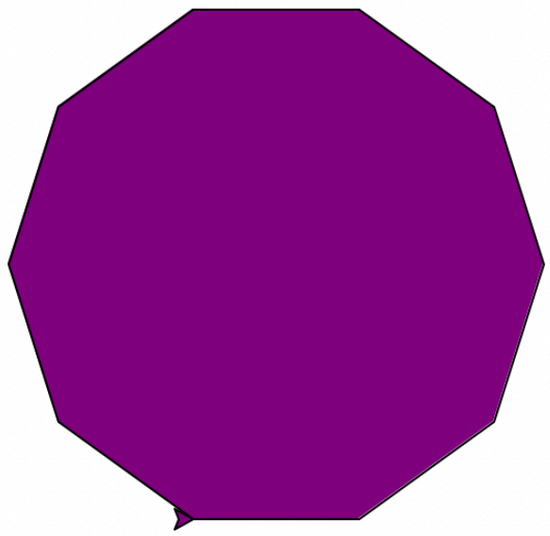
`draw_polygon(80, 10)`

Adding Color!

- What if we wanted to add some color to our shapes?

```
def draw_polygon_color(length, num_sides, color):
    # set the color we want to fill the shape with
    # color is a string
    fillcolor(color)

    begin_fill()
    for i in range(num_sides):
        fd(length)
        lt(360/num_sides)
    end_fill()
    done()
```



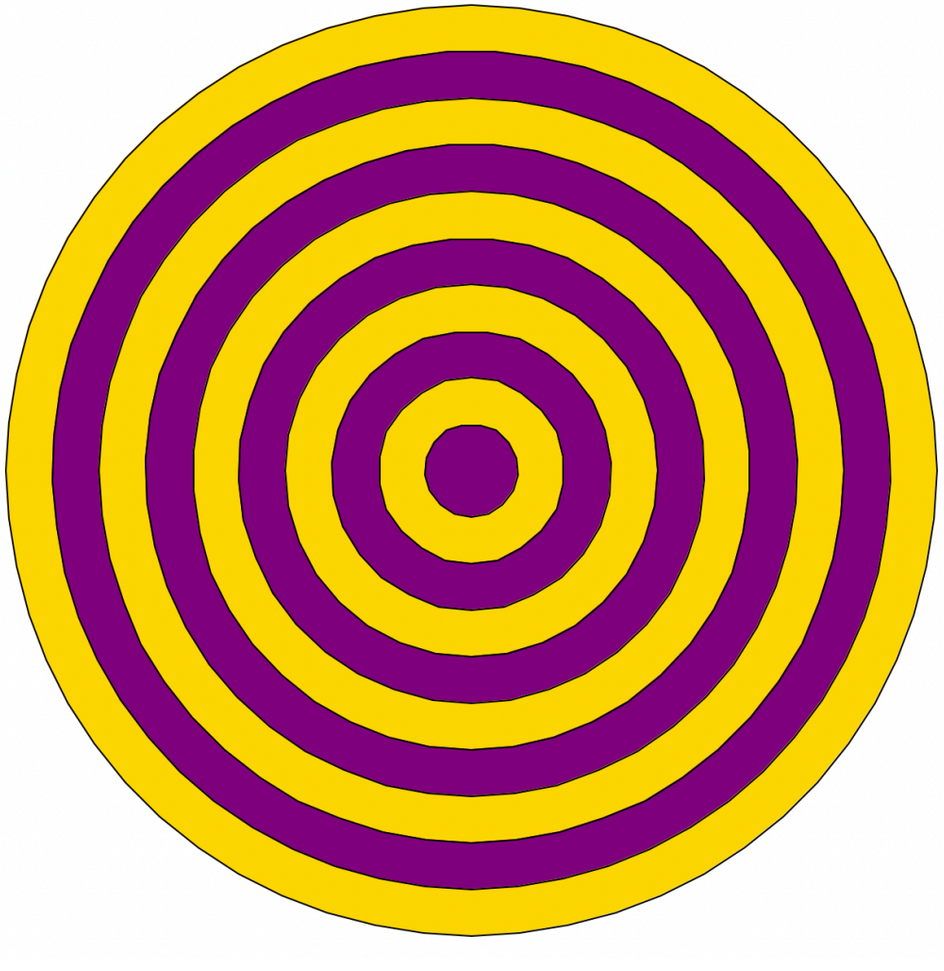


```
draw_polygon_color(80, 10, "gold")
```

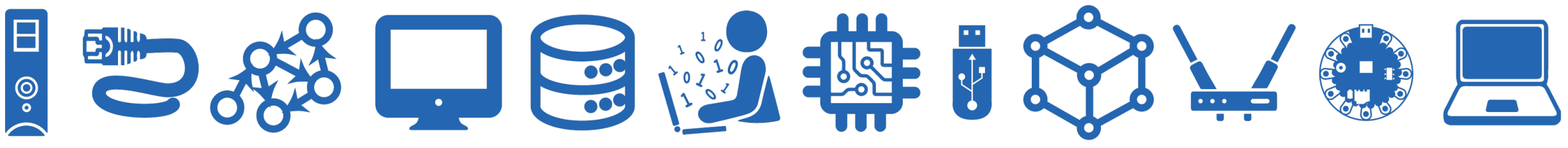
```
draw_polygon_color(80, 10, "purple")
```

Recursive Figures With Turtle

- Let's explore how to draw pictures recursively using Turtle
- Below we have a set of concentric circles of alternating colors
- How can this task be viewed recursively?



Example:
Concentric Circles



Concentric Circles With No Colors

- **Recursive idea**: we have circles within circles, and each circle becomes successively smaller. In addition to drawing the circles, let's keep track of the **number of circles** we draw.
- Let's first think about the circles without colors.
- **Base case**: radius of the circle is so small it's not worth drawing, return 0
- **Recursive step**:
  - Draw a single circle of radius `r`, increment total by 1
  - Recursively draw concentric circles starting with an outer circle of a slightly smaller radius `r-g` (where `g` is any positive number you want to shrink the radius by, or the "gap" between the circles)
  - Let's also count the number of circles we draw, so add one to our count!

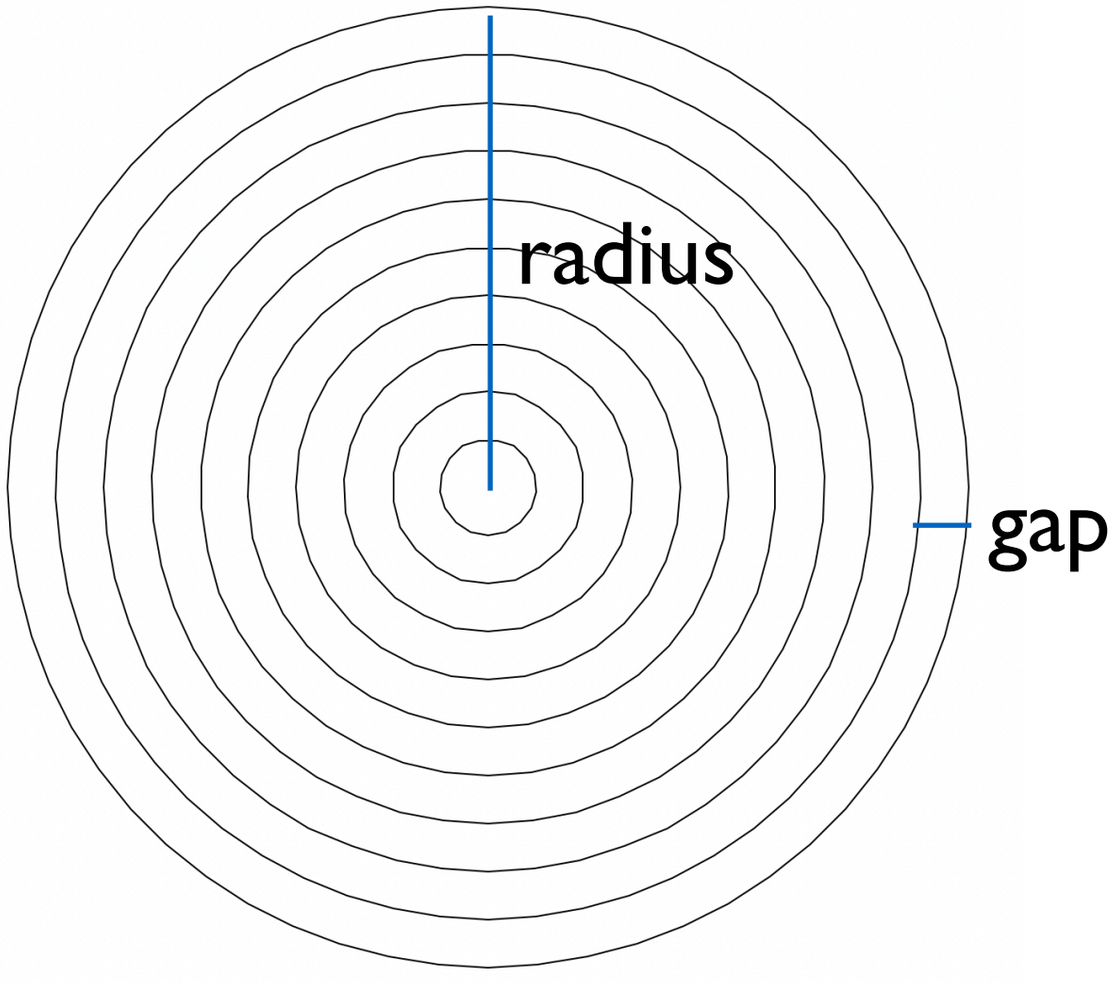
Counting the number of circles isn't necessary for drawing pictures, but it does make debugging easier!

Concentric Circles

- Function definition

  `concentric_circles(radius, gap)`

  - `radius`: radius of the outermost circle
  - `gap`: width of gap between circles



Concentric Circles

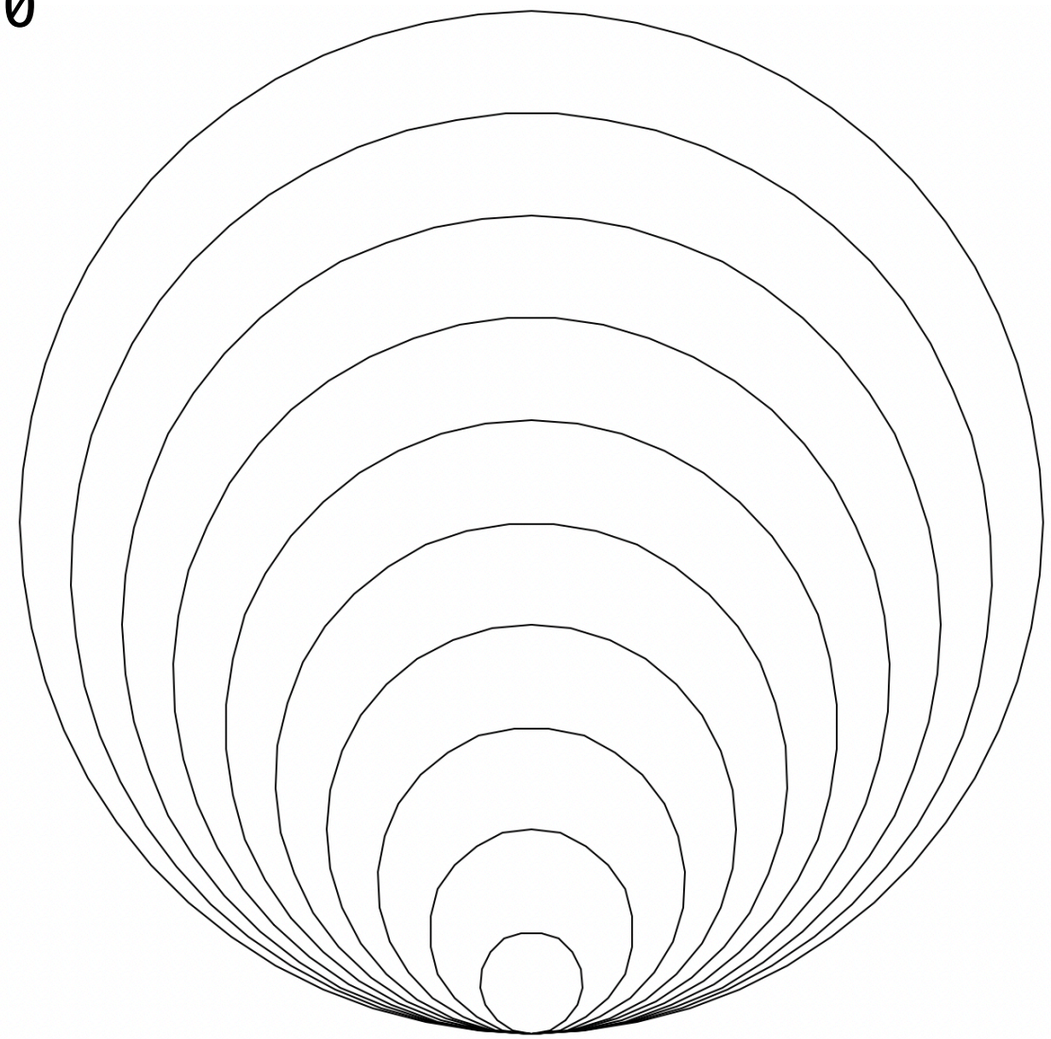
```
def concentric_circles(radius, gap):
    # base case, don't draw anything, return 0
    if radius < gap:
        return 0
    else:
        # tell the turtle draw a circle
        circle(radius)

        # recursive function call; draw smaller circles
        num = concentric_circles(radius-gap, gap)

        # we drew one circle in this step, plus however many we
        # drew recursively, so return 1 + num
        return 1 + num
```

- Are we done?

Concentric Circles

```
print("Num Circles:", concentric_circles(300, 30))
```

```
Num Circles: 10
```

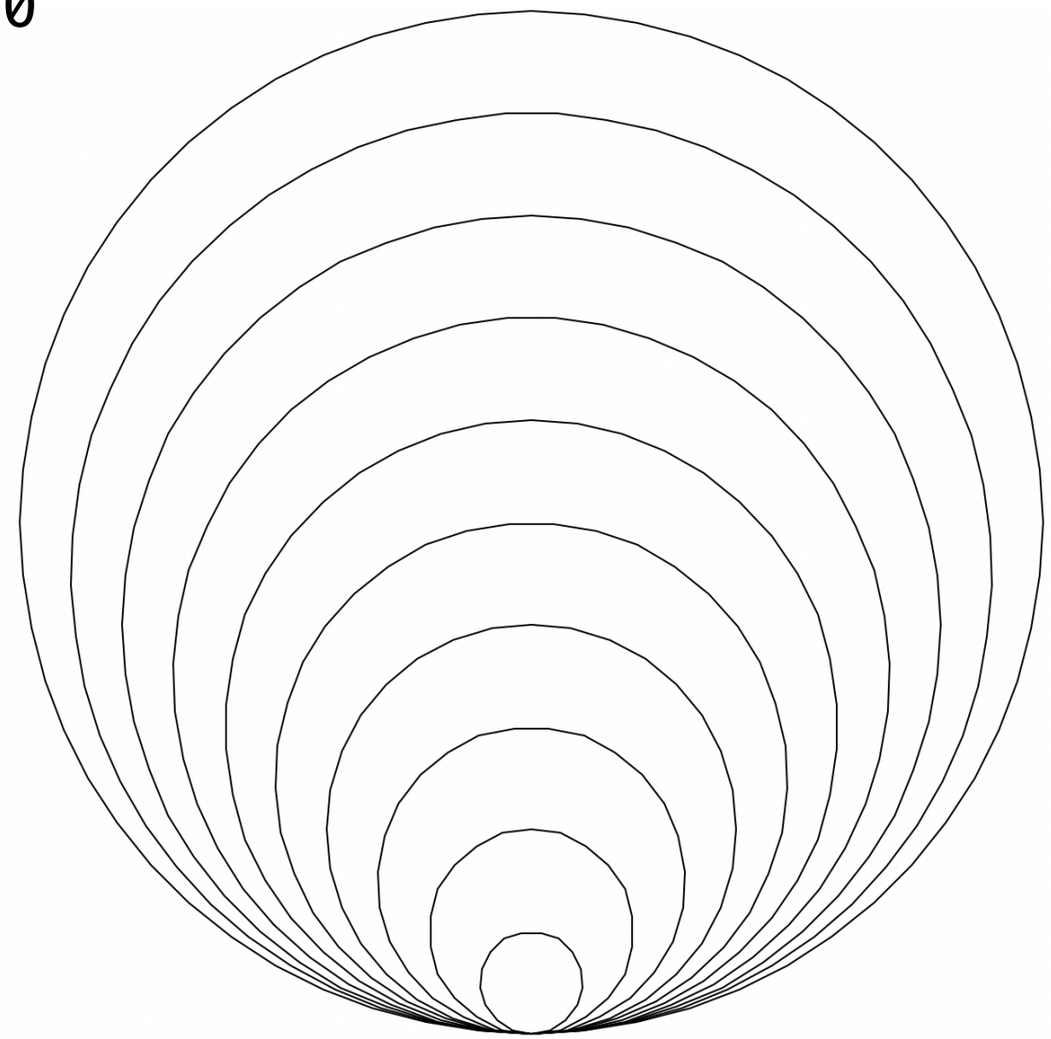


- Pretty picture, and almost there! But not quite right. What happened?

Concentric Circles

```
print("Num Circles:", concentric_circles(300, 30))
```

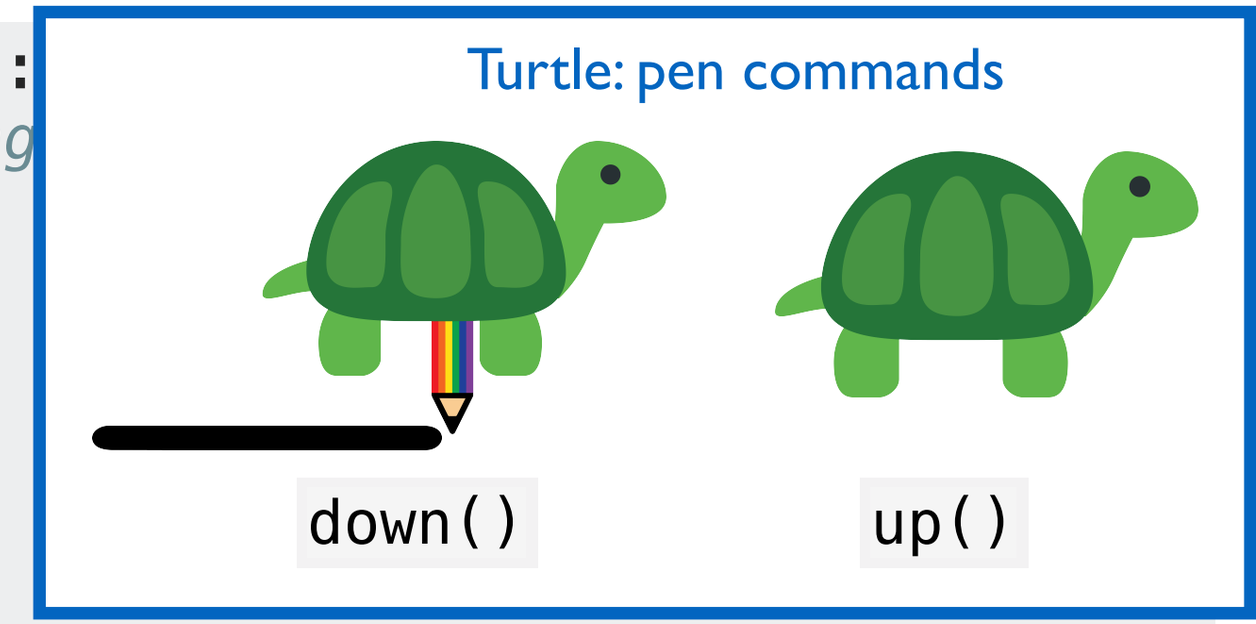
```
Num Circles: 10
```



- We need to reposition the turtle after each recursive call.

Concentric Circles

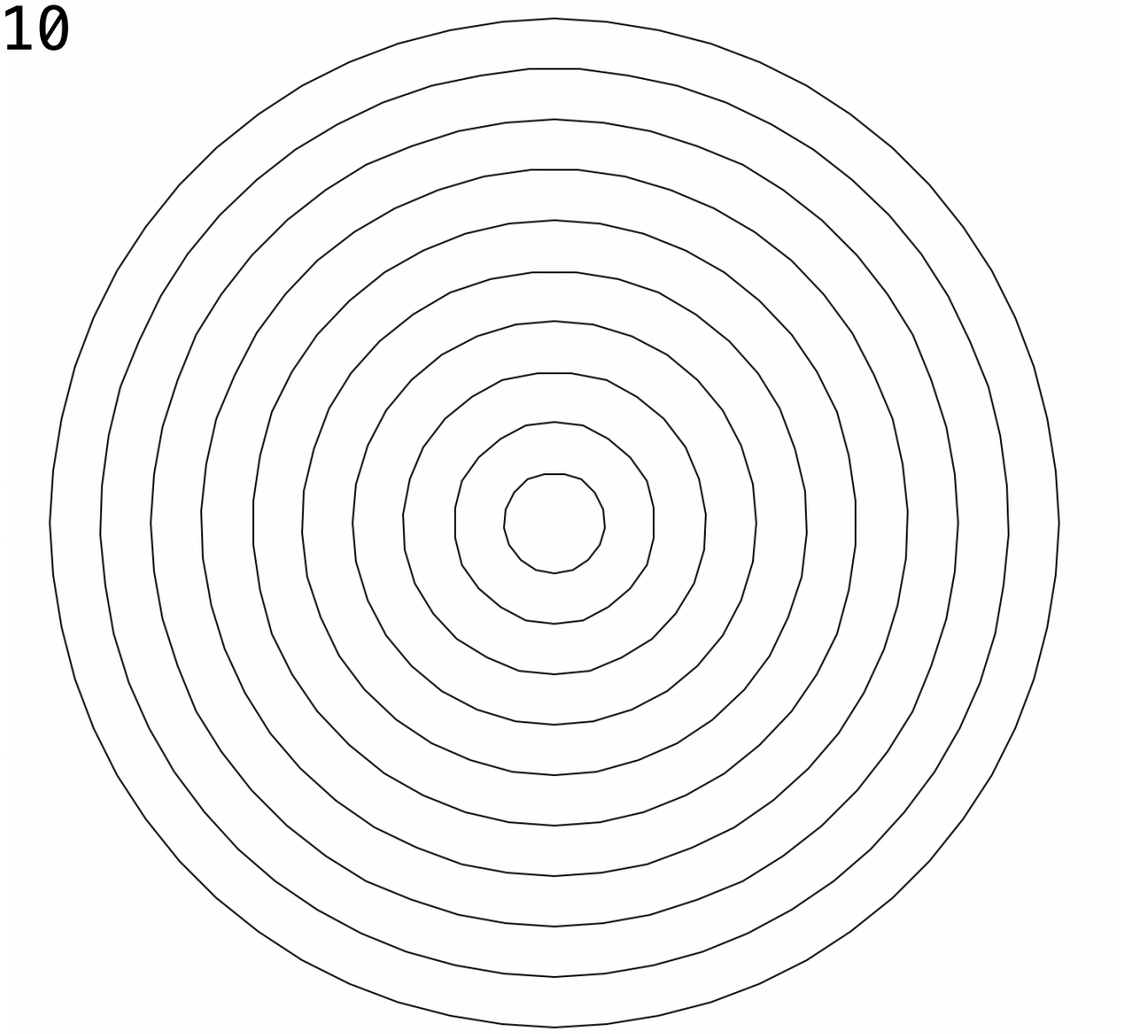
```
def concentric_circles(radius, gap):
    # base case, don't draw anything
    if radius < gap:
        return 0
    else:
        # pen down, draw circle
        down()
        circle(radius)

        # pen up, ensure the turtle doesn't draw while repositioning
        up()

        # reposition the turtle for the next circle
        lt(90)
        fd(gap)
        rt(90)

        # recursive function call; draw smaller circles
        num = concentric_circles(radius-gap, gap)

        # we drew one circle in this step, plus however many we
        # drew recursively, so return 1 + num
        return 1 + num
```



Concentric Circles

```
print("Num Circles:", concentric_circles(300, 30))
```

```
Num Circles: 10
```



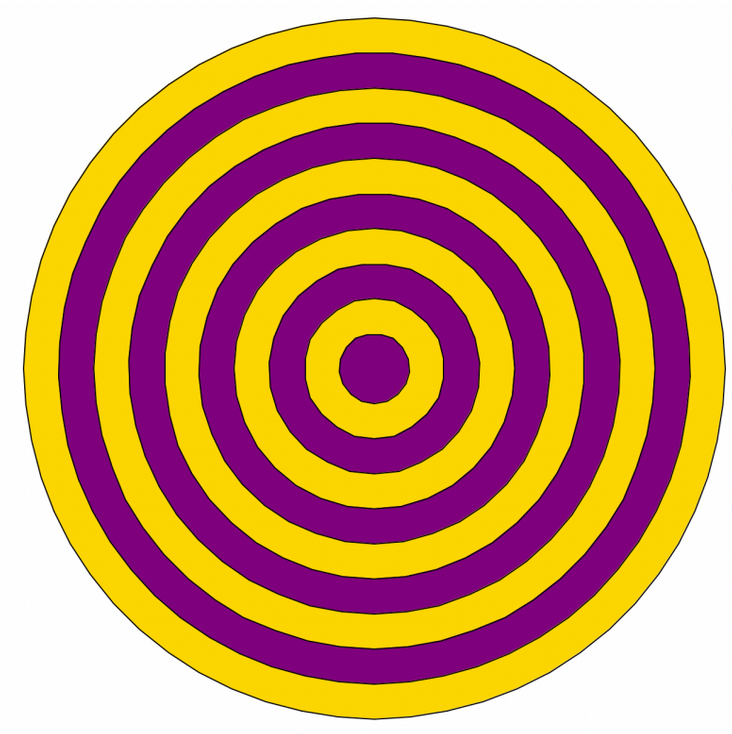
- Great! Now let's add some color.

Concentric Circles With Colors

- Function definition

```
concentric_circles(radius, gap, color_outer, color_inner)
```

  - `radius`: radius of the outermost circle
  - `gap`: width of the gap between circles
  - `color_outer`: color of the outermost circle
  - `color_inner`: color that alternates with color_outer



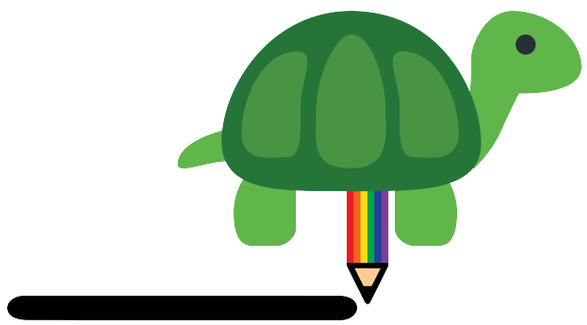
Concentric Circles: Adding Color

- Base case and recursive case stay the same
- How do we achieve the alternating colors?
- Just swap the order of the arguments in the recursive call
  - `color_outer` becomes `color_inner` and vice versa
- Let's also write a helper function to draw a circle filled in with some color to clean up the recursive function itself

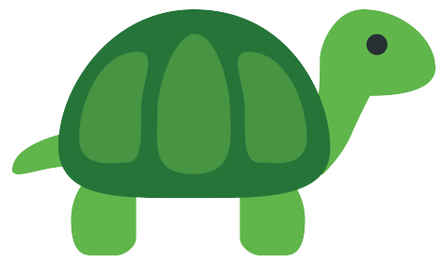
Helper Function

```
def draw_disc(radius, color):
    """
    Draw circle of a given radius
    and fill it with color
    """

    # put the pen down
    down()

    # set the color
    fillcolor(color)

    # draw the circle
    begin_fill()
    circle(radius)
    end_fill()

    # put the pen up
    up()
```

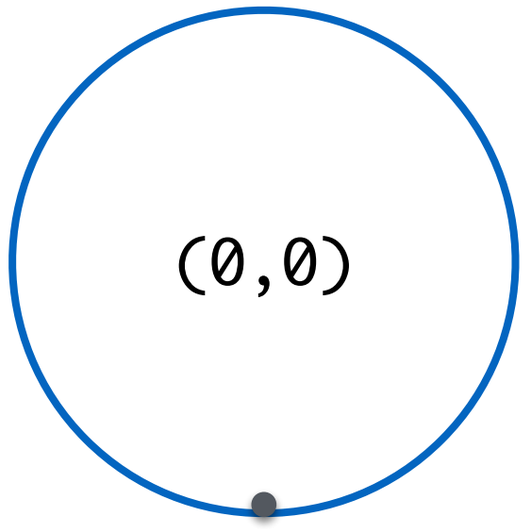
Turtle: pen commands



down()



up()

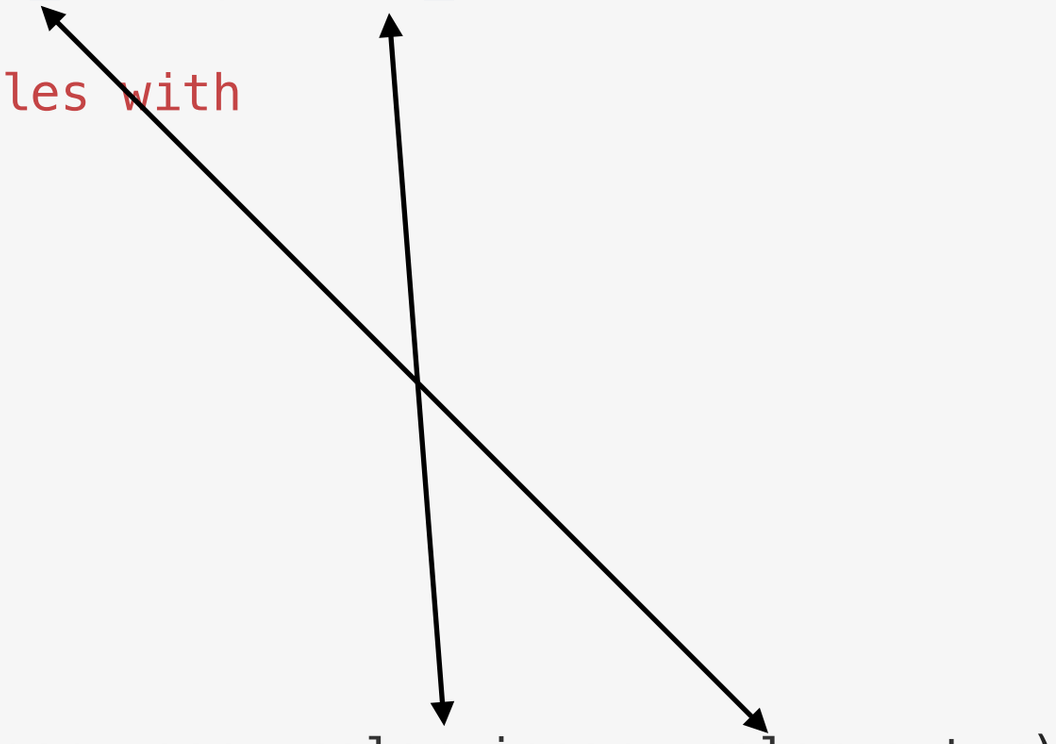


Starting position of turtle

(0, -radius)

The Recursive Function

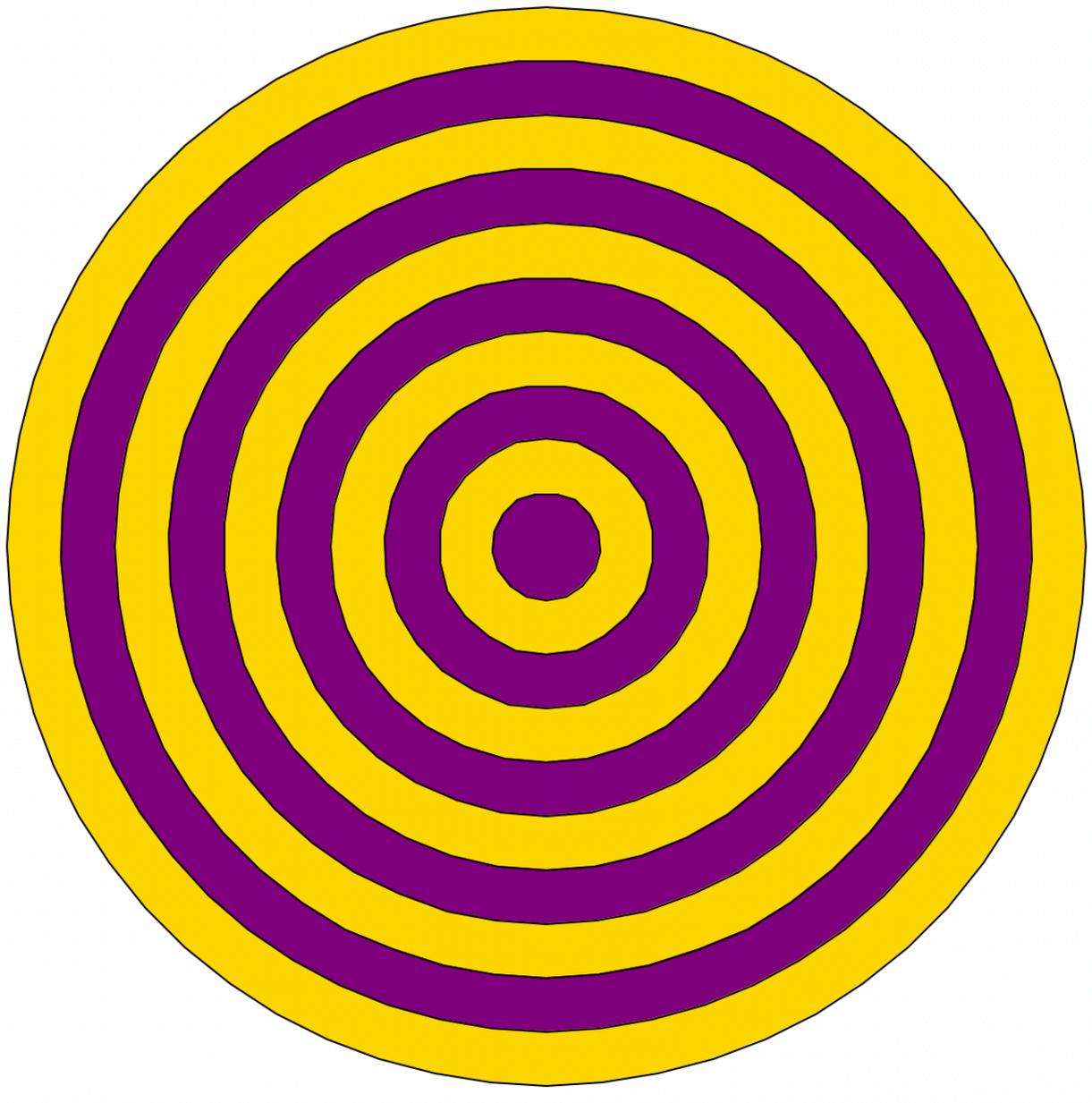
```
def concentric_circles_color(radius, gap, color_outer, color_inner):
    """
    Recursive function to draw concentric circles with
    alternating colors
    """
    # base case, don't draw anything, return 0
    if radius < gap:
        return 0
    else:
        draw_disc(radius, color_outer)
        lt(90)
        fd(gap)
        rt(90)
        num = concentric_circles_color(radius-gap, gap, color_inner, color_outer)
        return 1 + num
```



Concentric Circles

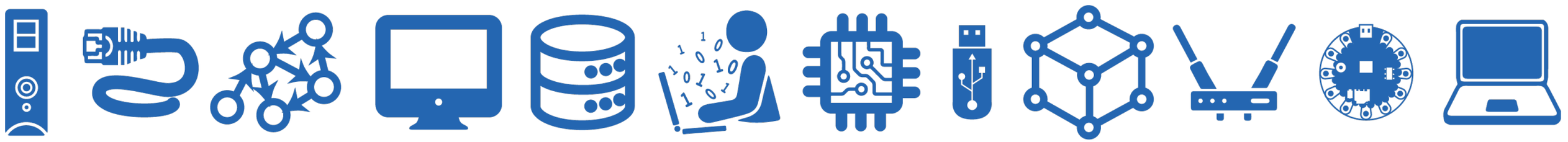
```
print("Num circles:", concentric_circles_color(300, 30, "gold", "purple"))
```

```
Num Circles: 10
```

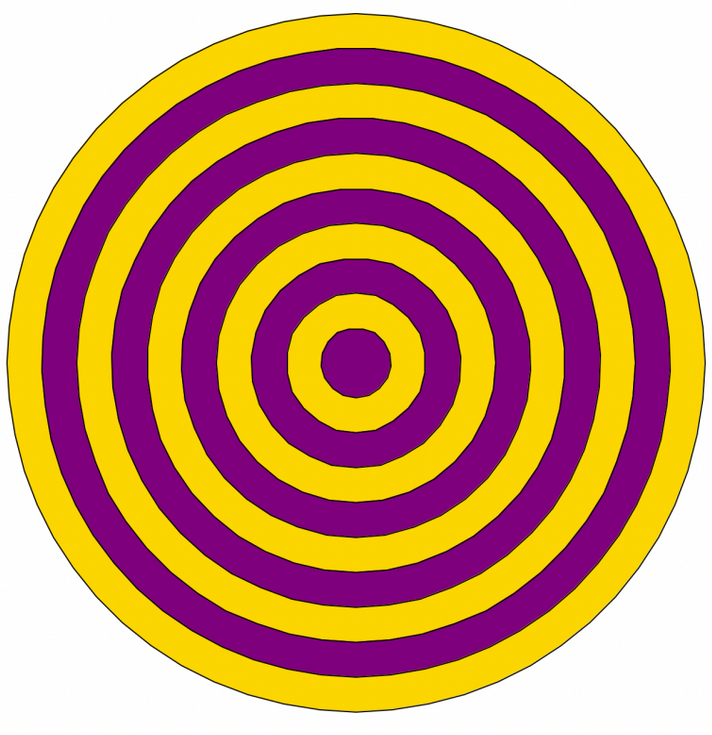


Function Frame Model:

concentric_circles



```
def concentric_circles(radius, gap, color_outer, color_inner):
  """Recursive function to draw concentric circles"""
 if radius < gap:
  return 0
 else:
  draw_disc(radius, color_out)
   lt(90); fd(gap); rt(90)
  num = concentric_circles(radius-gap, gap, color_in, color_out)
  return 1 + num
```



```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

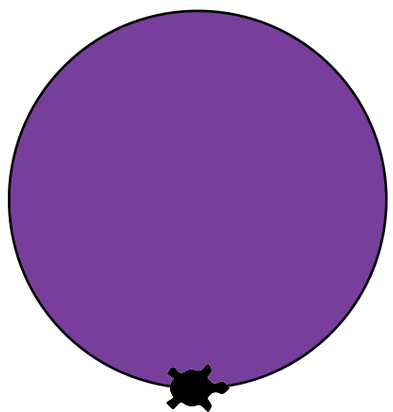
```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```



```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

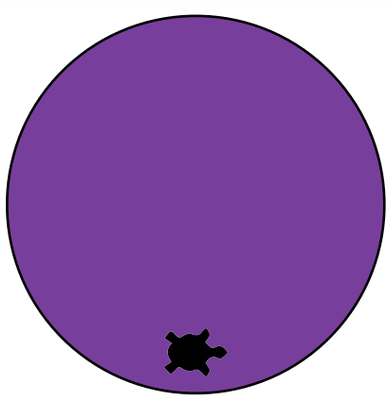
radius 18 gap 5

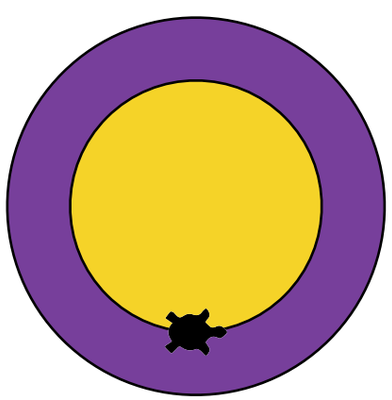
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```





```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

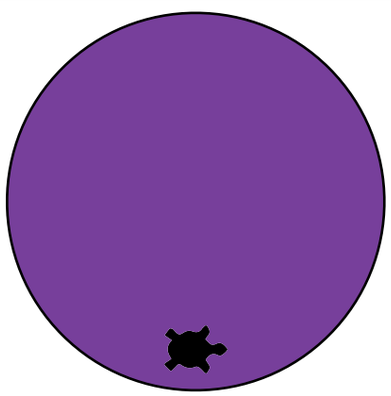
radius 13 gap 5

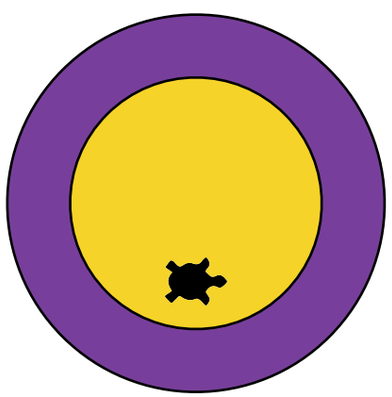
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

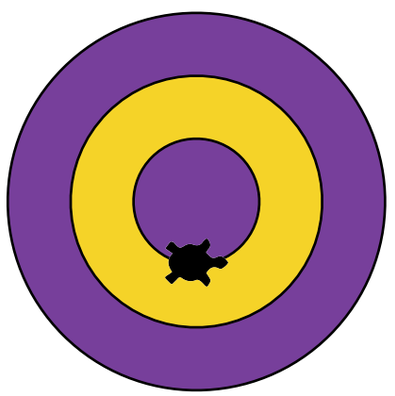
contrc_circles(8,5,'p','g')

radius 8 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

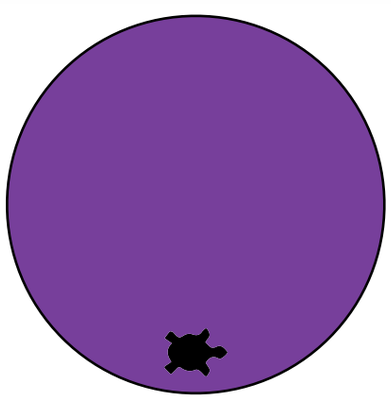
radius 13 gap 5

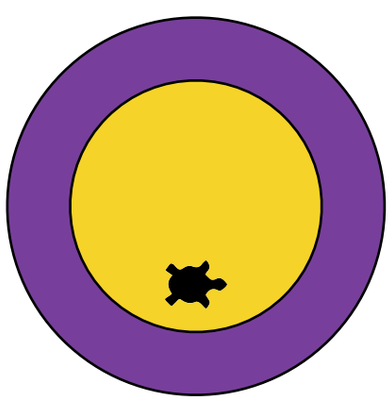
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

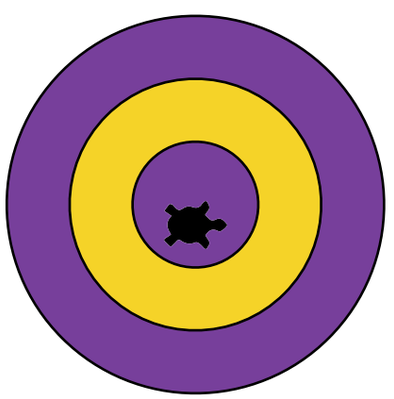
contrc_circles(8,5,'p','g')

radius 8 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(8,5,'p','g')

radius 8 gap 5

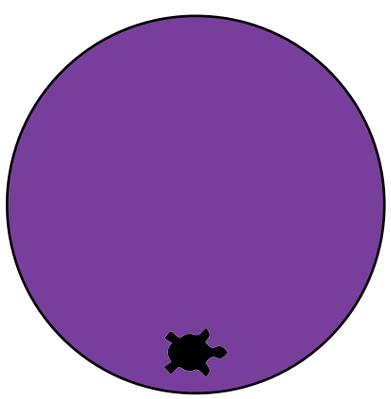
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

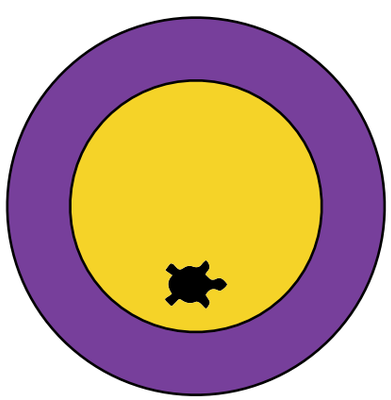
0

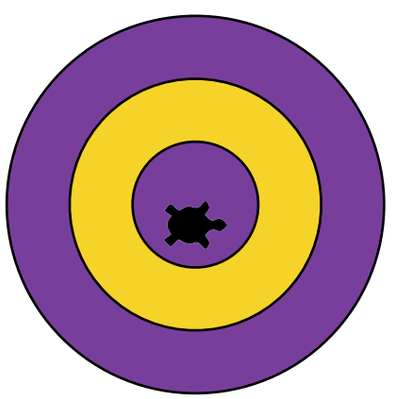
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(8,5,'p','g')

radius 8 gap 5

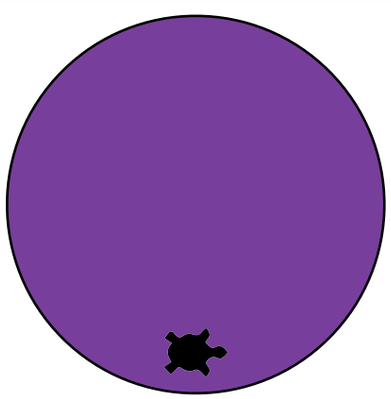
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

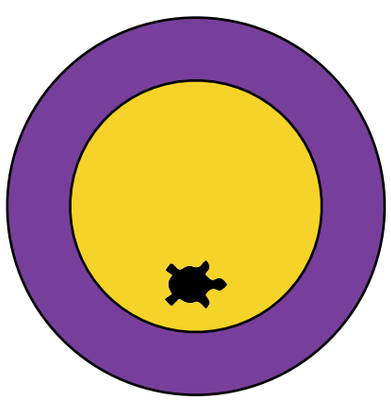
0

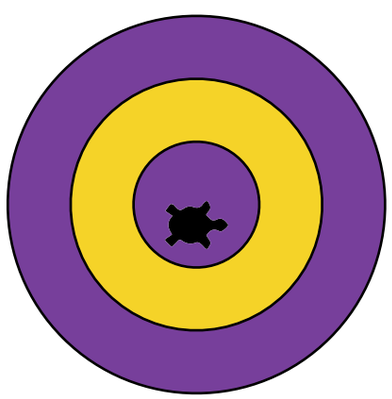
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 1 entric_circles
        -g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(8,5,'p','g')

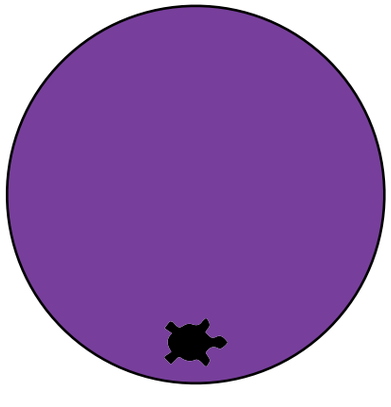
radius 8 gap 5

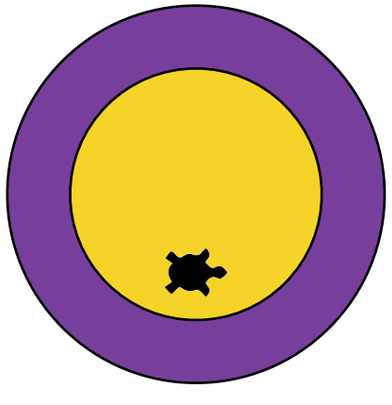
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); ap); rt(90)
  num = c 0 tric_circles
        (  , g, clr_i, clr_o)
  return 1 + num
```

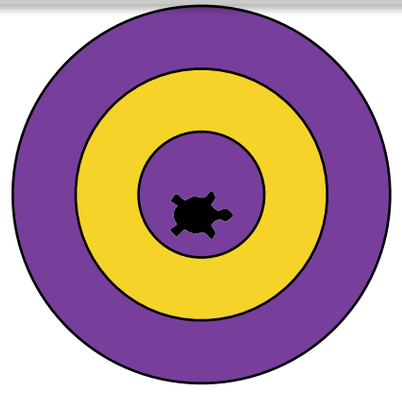
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

1

contrc_circles(8,5,'p','g')

radius 8 gap 5

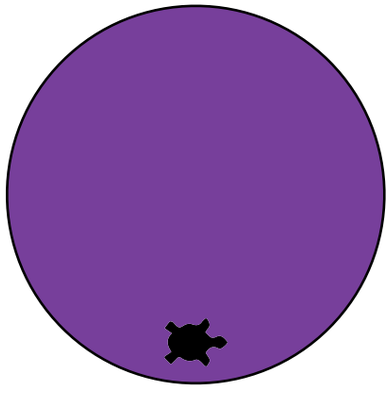
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

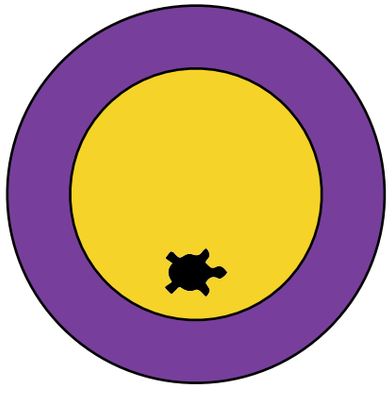
0

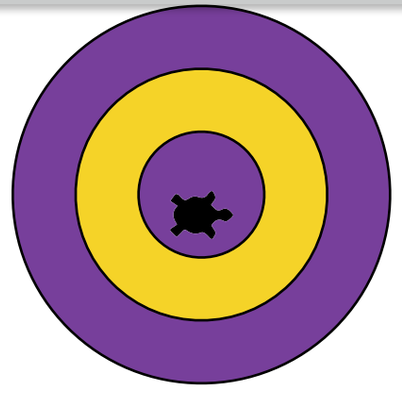
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 2 ntric_circles
        g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 1 entric_circles
        -g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(8,5,'p','g')

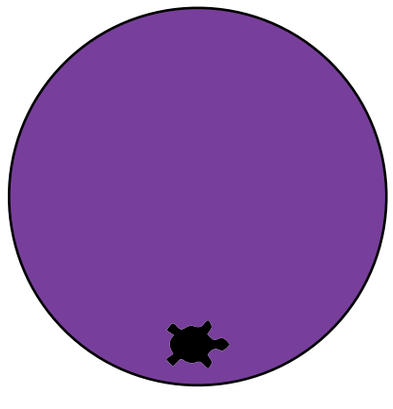
radius 8 gap 5

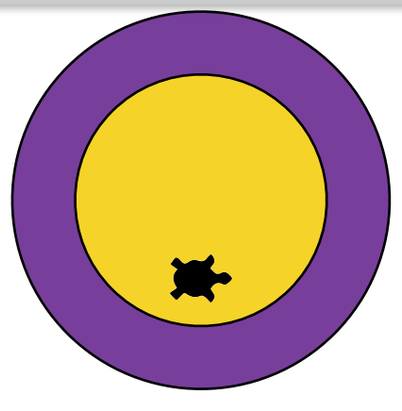
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); ap); rt(90)
  num = c 0 tric_circles
        ( , g, clr_i, clr_o)
  return 1 + num
```

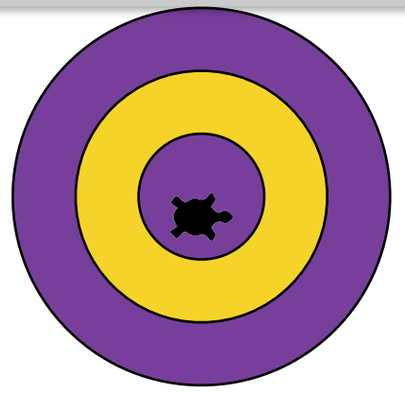
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> concentric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 2 ntric_circles
        g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 1 entric_circles
        -g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(8,5,'p','g')

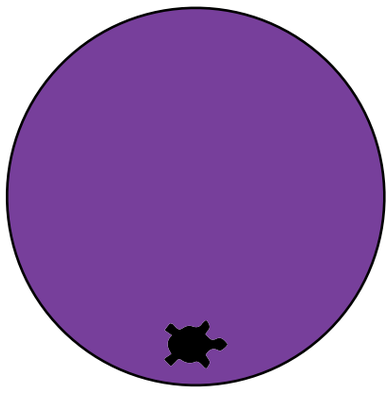
radius 8 gap 5

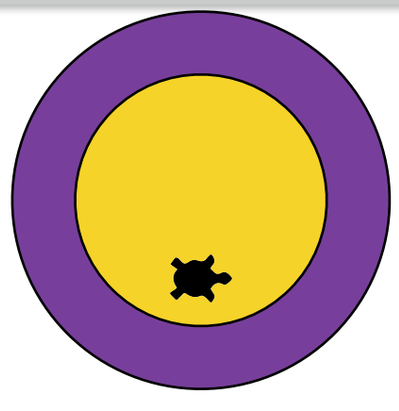
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); ap); rt(90)
  num = c 0 tric_circles
        ( , g, clr_i, clr_o)
  return 1 + num
```

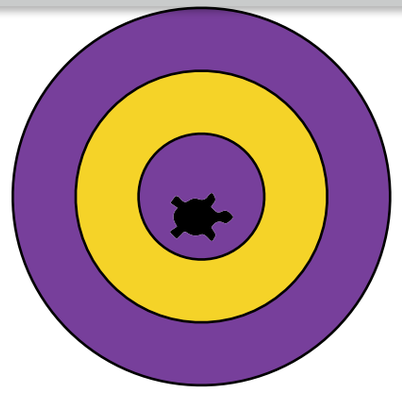
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```







```
def concentric_circles(radius, gap, color_out, color_in):
  """Recursive function to draw concentric circles"""
  if radius < gap:
    return 0
  else:
    draw_disc(radius, color_out)
    lt(90); fd(gap); rt(90)
    num = concentric_circles(radius-gap, gap, color_in, color_out)
    return 1 + num
```

```
>>> co 3 ntric_circles(18, 5, "purple", "gold")
```

contrc_circles(18,5,'p','g')

radius 18 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 2 ntric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(13,5,'g','p')

radius 13 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = 1 entric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```

contrc_circles(8,5,'p','g')

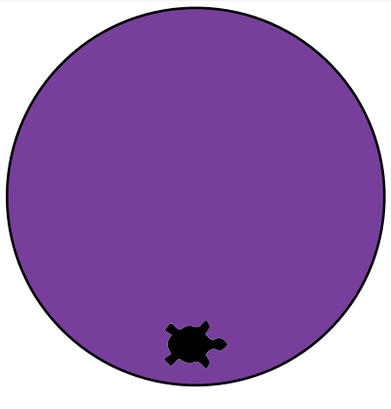
radius 8 gap 5

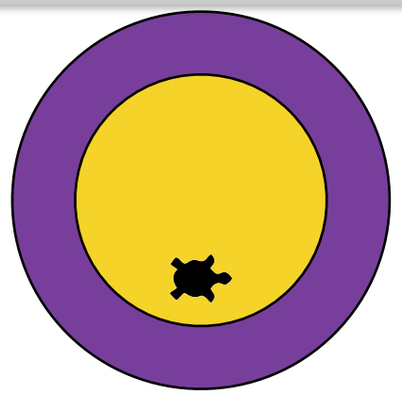
```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = c 0 tric_circles
        ( , g, clr_i, clr_o)
  return 1 + num
```

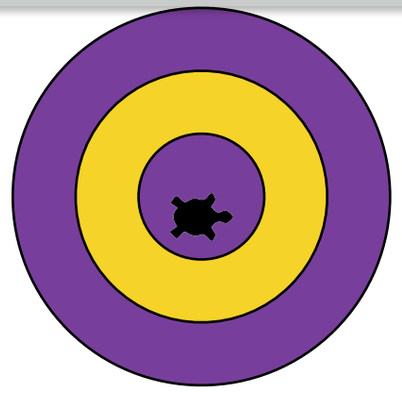
contrc_circles(3,5,'g','p')

radius 3 gap 5

```
if rad < gap:
  return 0
else:
  draw_disc(rad, clr_o)
  lt(90); fd(gap); rt(90)
  num = concentric_circles
        (rad-g, g, clr_i, clr_o)
  return 1 + num
```



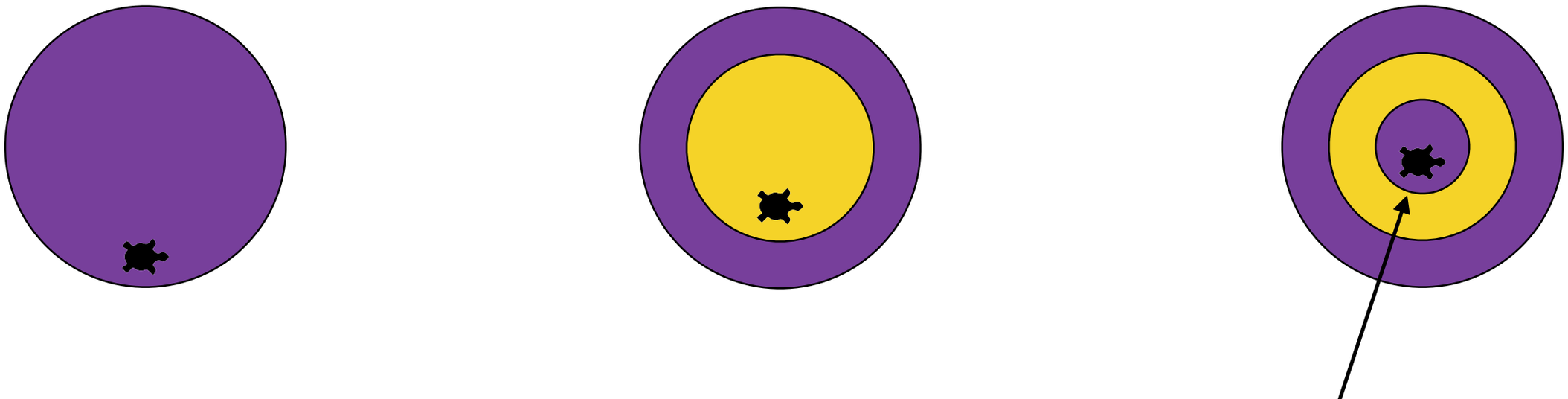




Function Invariants

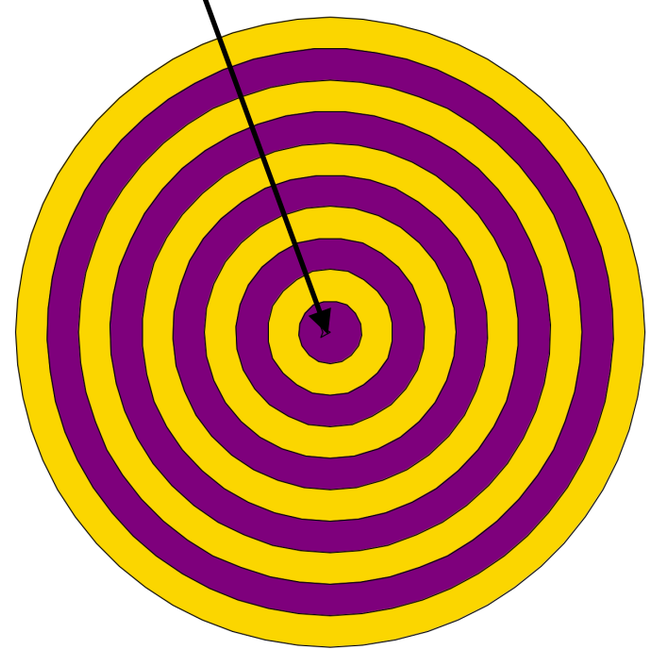
- Where does the turtle end up in this example with `concentric_circles_color`?

```
concentric_circles(18, 5, 'purple', 'gold')
```

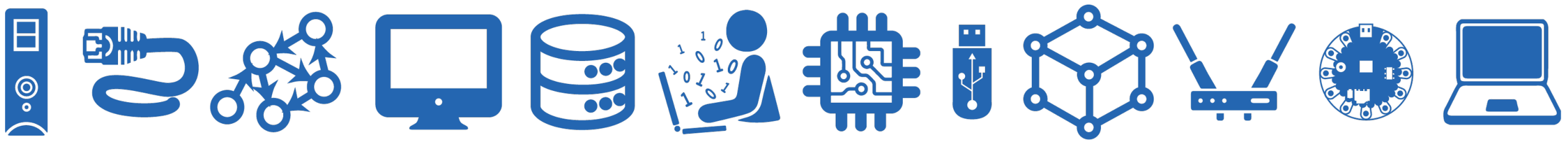


- The turtle does not end where it starts

```
def concentric_circles_color(radius, gap, color_outer, color_inner):
    """
    Recursive function to draw concentric circles with
    alternating colors
    """
    # base case, don't draw anything, return 0
    if radius < gap:
        return 0
    else:
        draw_disc(radius, color_outer)
        lt(90)
        fd(gap)
        rt(90)
        num = concentric_circles_color(radius-gap, gap, color_inner, color_outer)
        return 1 + num
```

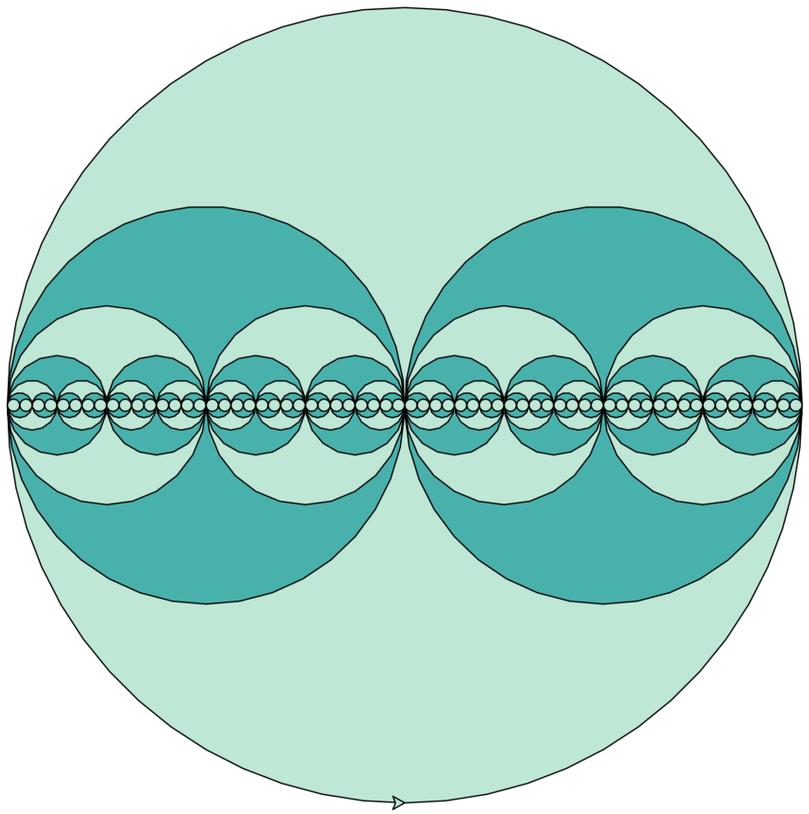


Example: Nested Circles



Invariance of Recursive Functions

- Let's do an example with multiple recursive calls: nested circles

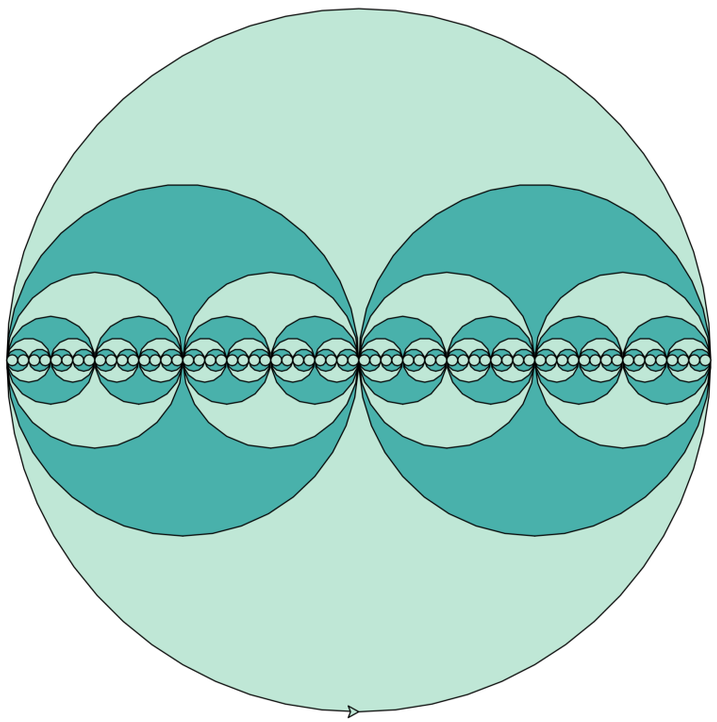


Multiple Recursive Calls

- **Example:** Nested circles function definition

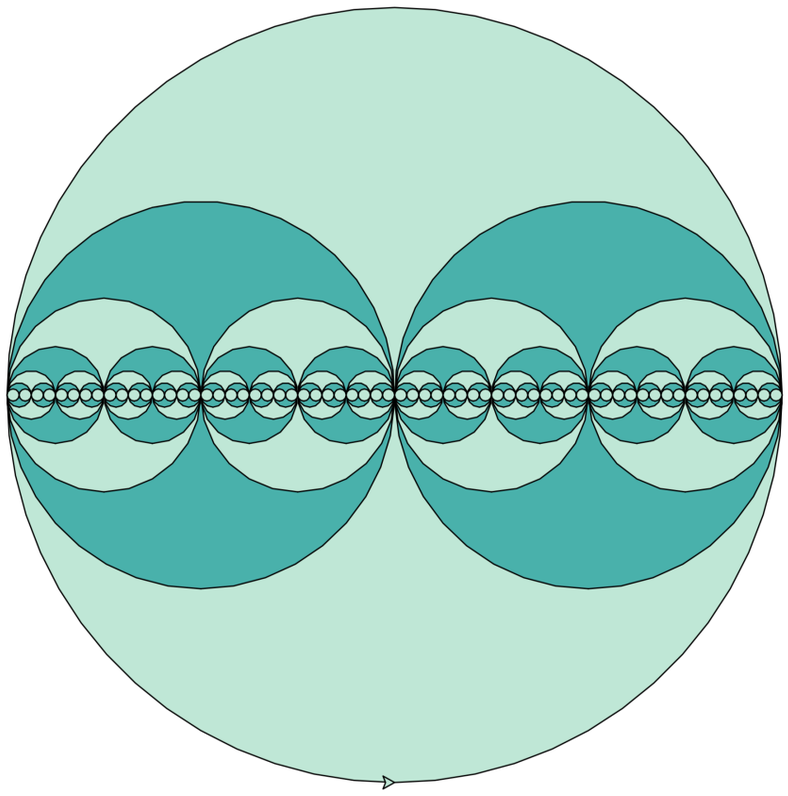
```
nested_circles(radius, min_radius, color_out, color_alt)
```

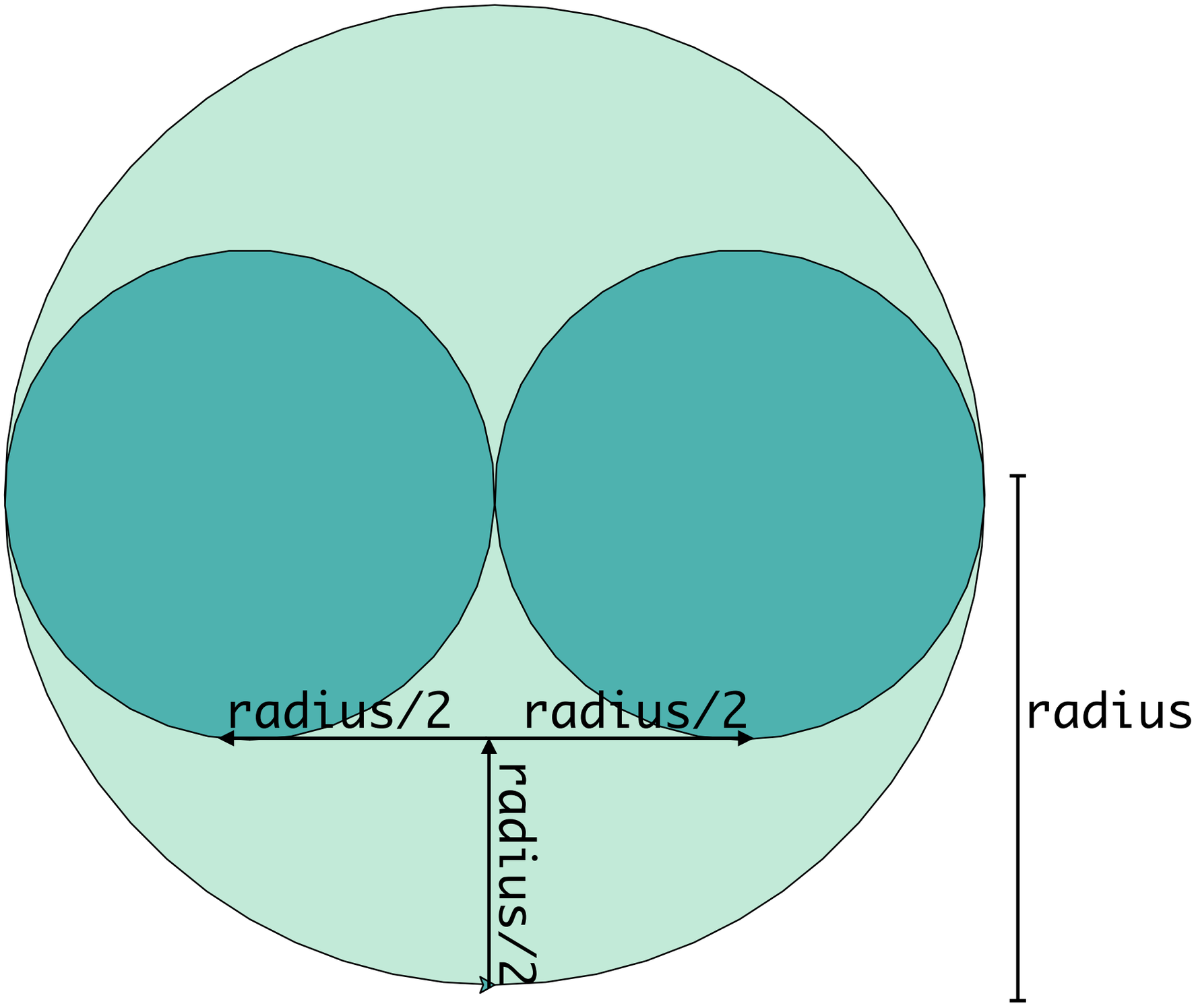
  - `radius`: radius of the outermost circle
  - `min_radius`: minimum radius of any circle
  - `color_out`: color of the outermost circle
  - `color_alt`: color that alternates with colorOut



Nested Circles

- **Base case?**
  - When radius becomes less than min_radius
  - Don't draw anything return 0
- **Recursive case**
  - Draw the outer circle, add one to total
  - Position turtle for recursive calls



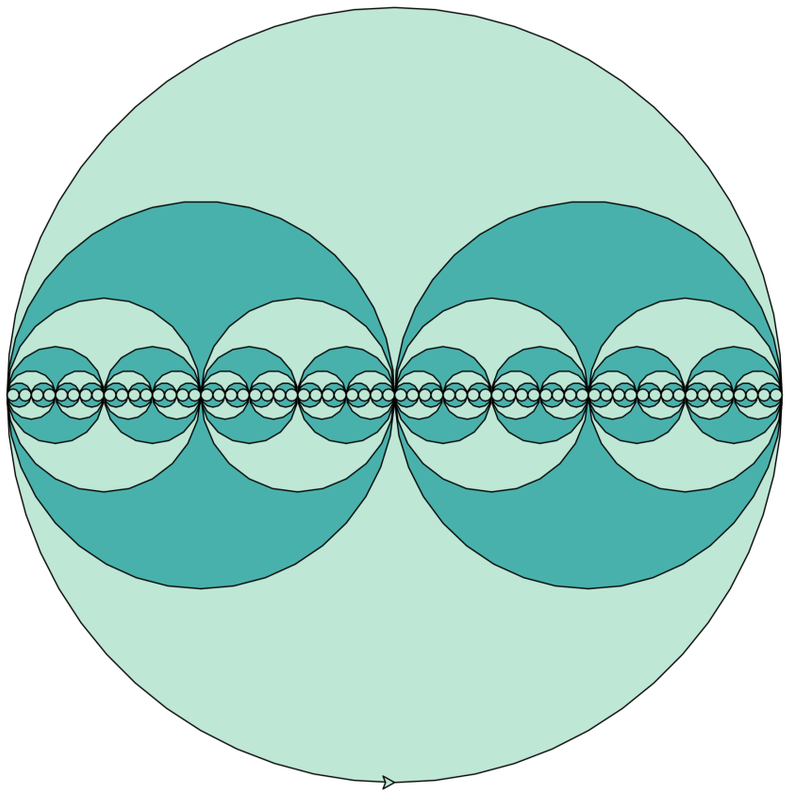


Starting position of turtle

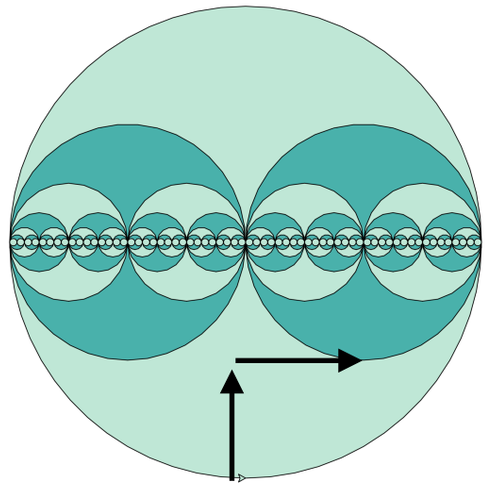
```
nested_circles(300, 150)
```

Nested Circles

- **Base case?**
  - When radius becomes less than minRadius
  - Don't draw anything return 0
- **Recursive case**
  - Draw the outer circle, add one to total
  - Position turtle for recursive calls
  - How many recursive calls do we need?
    - Two! Right subcircle and left subcircle



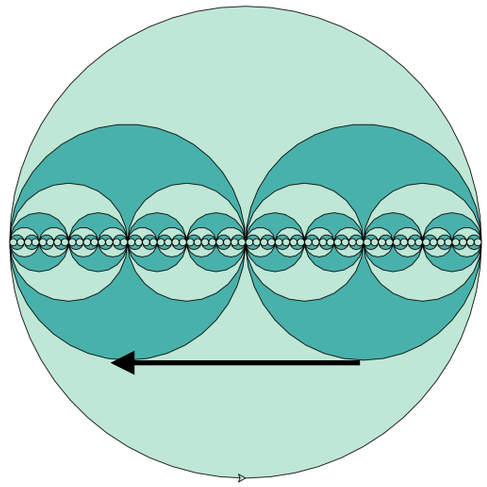
Nested Circles



- **Recursive case**
  - Draw the outer circle, add one to total
  - Position turtle for right recursive subcircle

```
def nested_circles(radius, min_radius, color_out, color_alt):
    if radius < min_radius:
        return 0
    else:
        # contribute to the solution
        draw_disc(radius, color_out)

        # save half of radius
        half_radius = radius/2

        # position the turtle to draw right subcircle
        lt(90); fd(half_radius); rt(90); fd(half_radius)

        # draw right subcircle recursively
        right = nested_circles(half_radius, min_radius, color_alt, color_out)
```

Nested Circles



- **Recursive case**
  - Move the turtle to draw left subcircle recursively
  - (continued from previous slide)

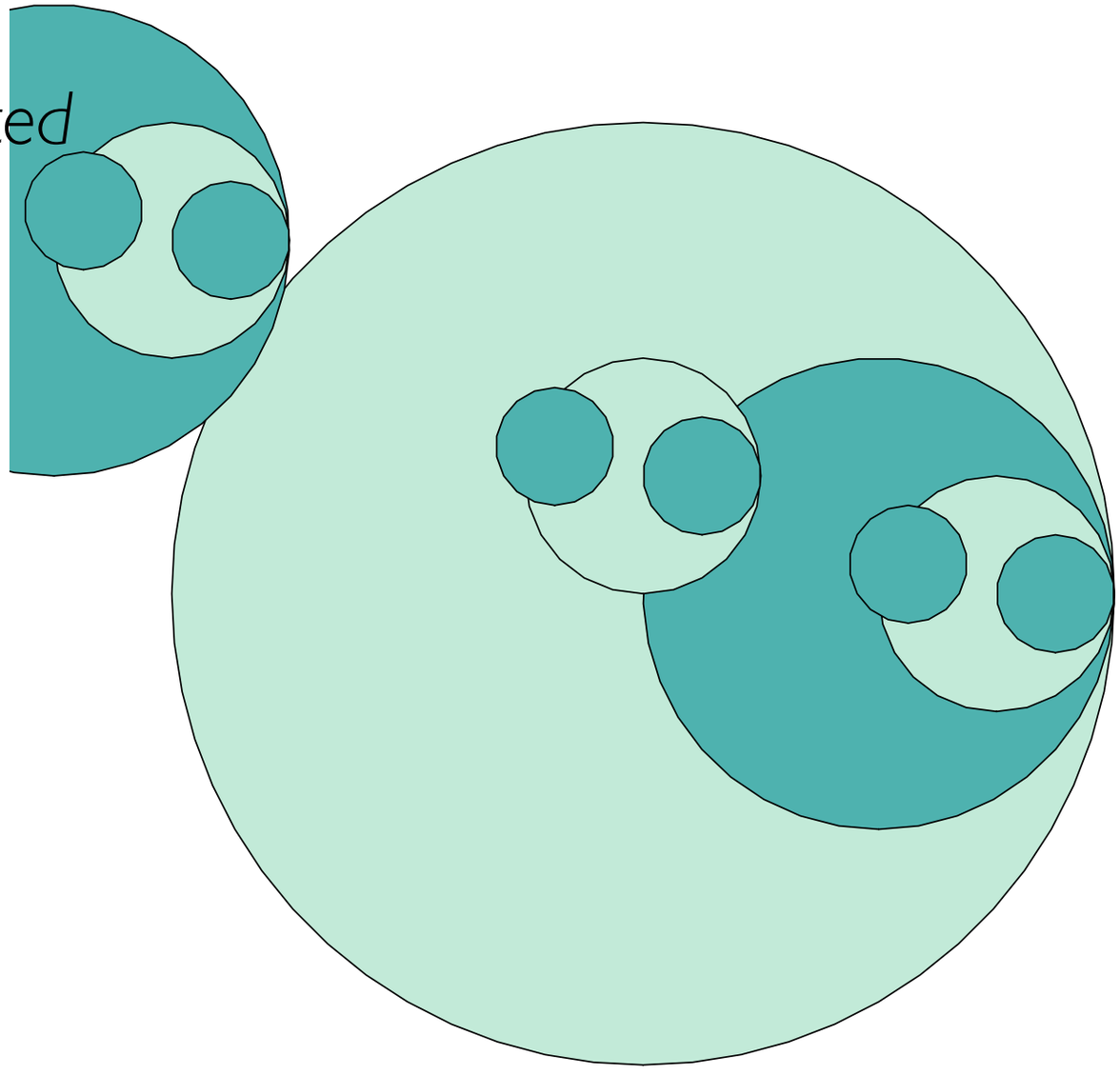
```
    # draw right subcircle recursively
    right = nested_circles(half_radius, min_radius, color_alt, color_out)

    # position turtle for left subcircle
    bk(radius)

    # draw left subcircle recursively
    left = nested_circles(half_radius, min_radius, color_alt, color_out)

    # add our count of subcircles
    return 1 + left + right
```

Nested Circles

- **Recursive case**
    - Are we done? Let's try it!

Nested Circles

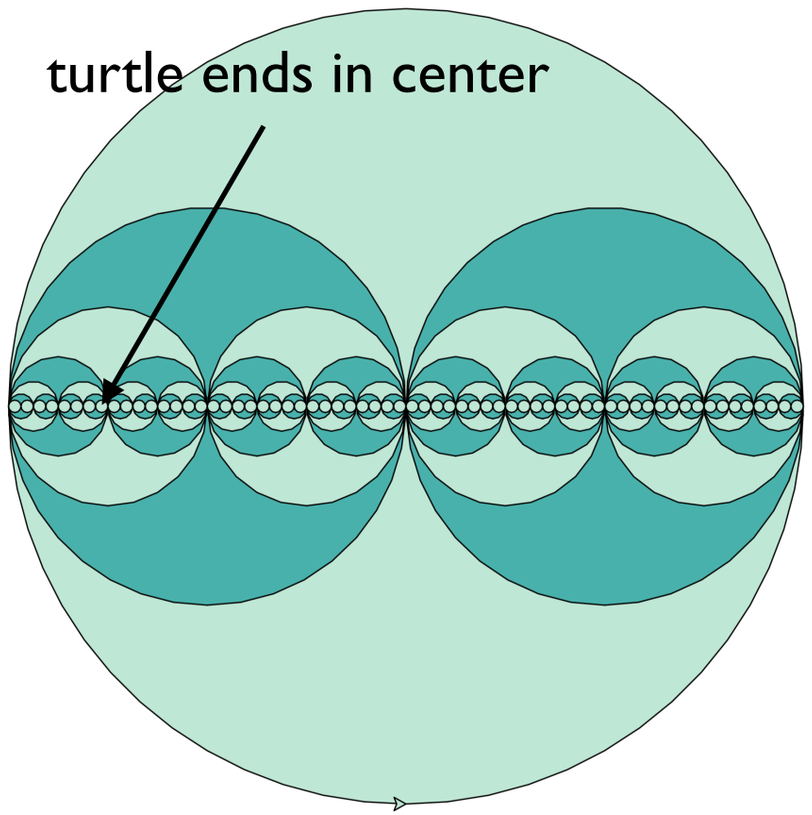
- **Recursive case**
  - What happened?!
  - We made assumptions about where the turtle started, that wasn't true!
  - Need turtle to *end* where it *started*



Function Invariants

- A function **invariant** is a property that is true *before* and *after* the function is invoked
- Right now the turtle's position in our `nested_circles` function is not invariant
  - That is, the turtle does not always end were it starts
- How can we preserve this invariant? By returning the turtle to starting position!

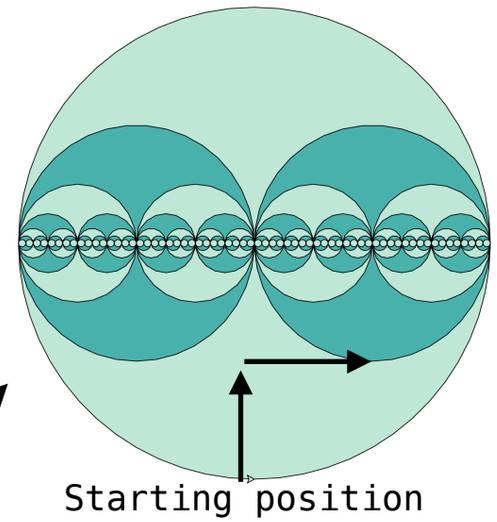
```
def nested_circles(radius, min_radius, color_out, color_alt):
    if radius < min_radius:
        return 0
    else:
        draw_disc(radius, color_out)
        h_r = radius/2

        lt(90); fd(h_r); rt(90); fd(h_r)

        right = nested_circles(h_r, min_radius, color_alt, color_out)

        bk(radius)

        left = nested_circles(h_r, min_radius, color_alt, color_out)

        fd(h_r);  lt(90);  bk(h_r);  rt(90)
        return 1 + right + left
```

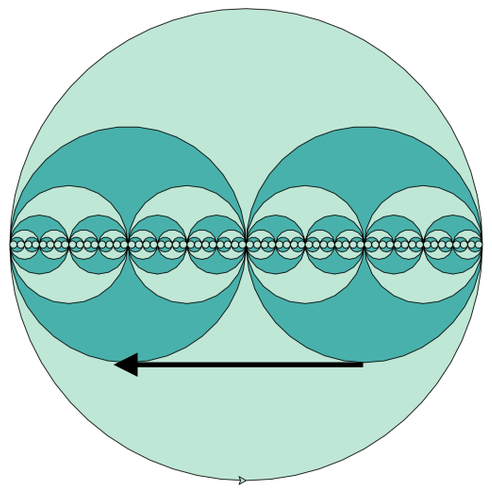


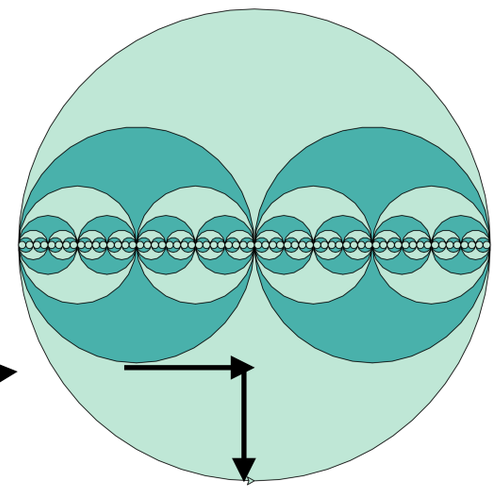
Maintaining Invariants

- Any turtle movements that happen before the recursive call should be "undone" after the recursive call to maintain our "position" invariant
- **Rule of thumb:** always return turtle to its starting position

```
def nested_circles(radius, min_radius, color_out, color_alt):
    if radius < min_radius:
        return 0
    else:
        # contribute to the solution
        draw_disc(radius, color_out)

    # save half of radius
    half_radius = radius/2

    # position the turtle to draw right subcircle
    lt(90); fd(half_radius); rt(90); fd(half_radius)

    # draw right subcircle recursively
    right = nested_circles(half_radius, min_radius, color_alt, color_out)

    # position turtle for left subcircle
    bk(radius)

    # draw left subcircle recursively
    left = nested_circles(half_radius, min_radius, color_alt, color_out)

    # bring turtle back to start position
    fd(half_radius);  lt(90);  bk(half_radius);  rt(90)

    # return total number of circles drawn
    return 1 + right + left
```

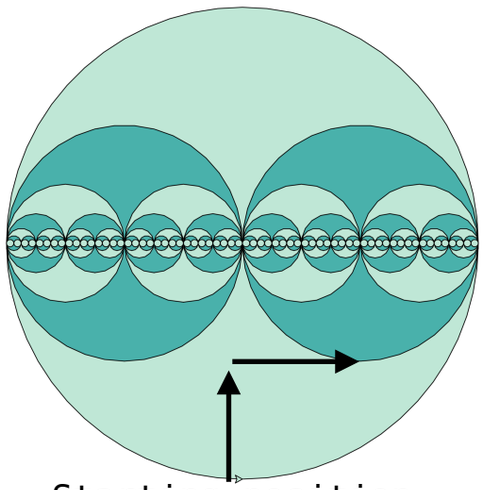




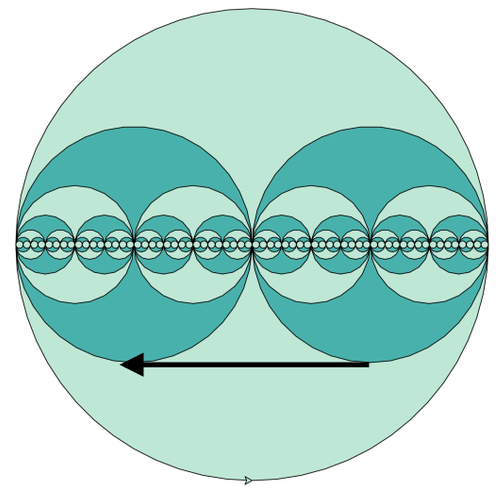


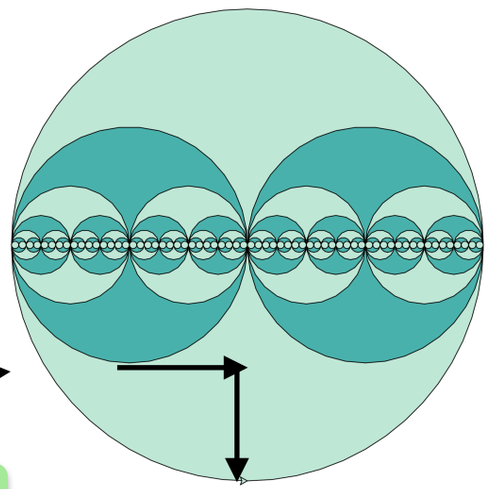
Maintaining Invariants

- Move turtle back to starting position to maintain `nested_circle`'s **invariants**

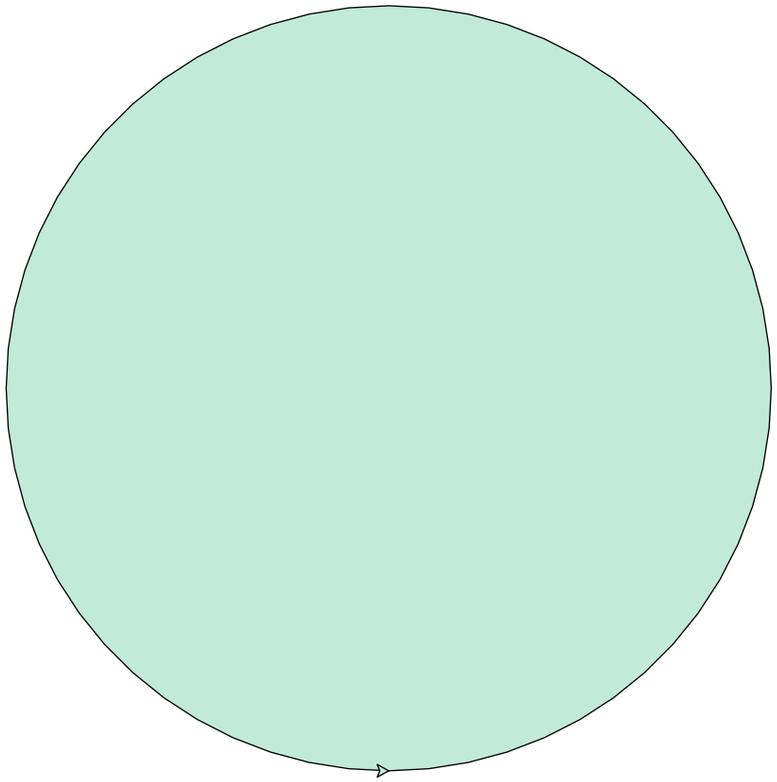



```
def nested_circles(radius, min_radius, color_out, color_alt):
    if radius < min_radius:
        return 0
    else:
        # contribute to the solution
        draw_disc(radius, color_out)

    # save half of radius
    half_radius = radius/2

    # position the turtle to draw right subcircle
    lt(90); fd(half_radius); rt(90); fd(half_radius)

    # draw right subcircle recursively
    right = nested_circles(half_radius, min_radius, color_alt, color_out)

    # position turtle for left subcircle
    bk(radius)

    # draw left subcircle recursively
    left = nested_circles(half_radius, min_radius, color_alt, color_out)

    # bring turtle back to start position
    fd(half_radius);  lt(90);  bk(half_radius);  rt(90)

    # return total number of circles drawn
    return 1 + right + left
```

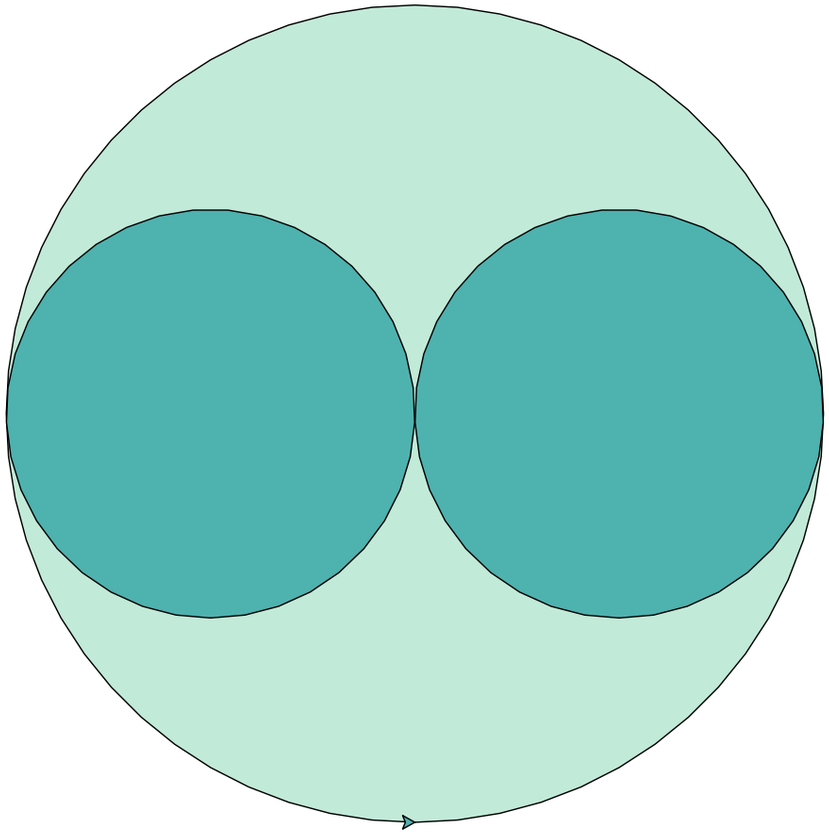




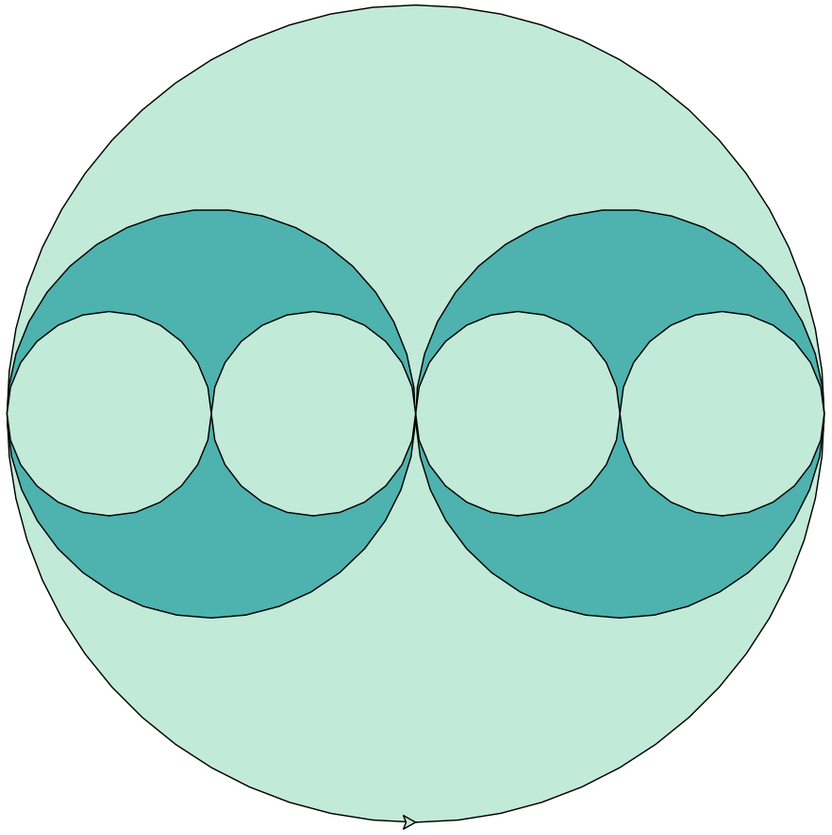
Maintain invariance



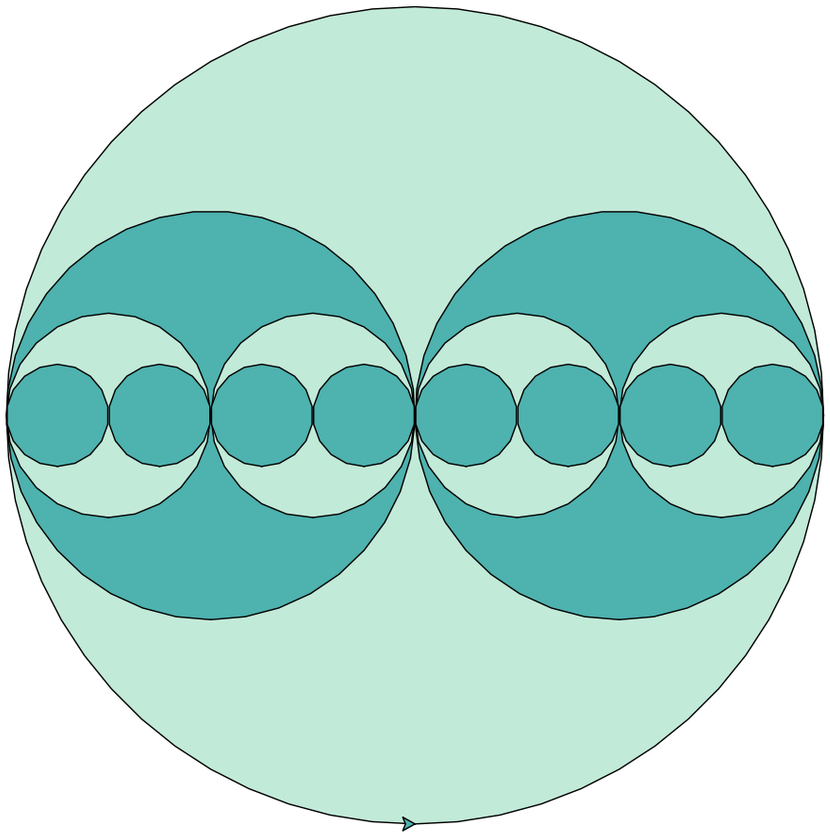
`nestedCircles(300, 300)`



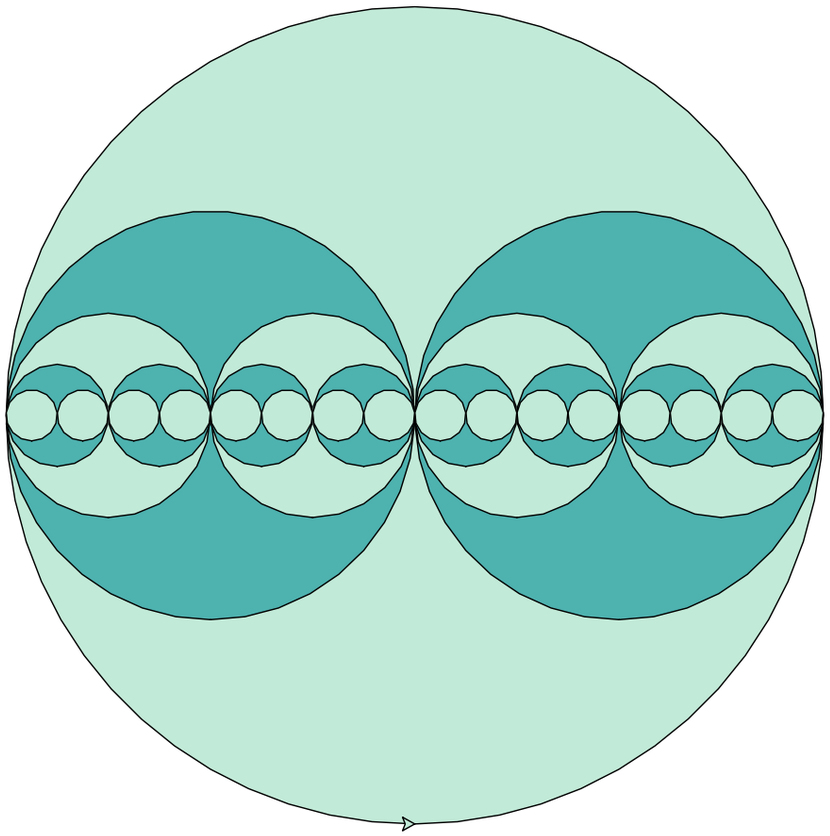
`nestedCircles(300, 150)`



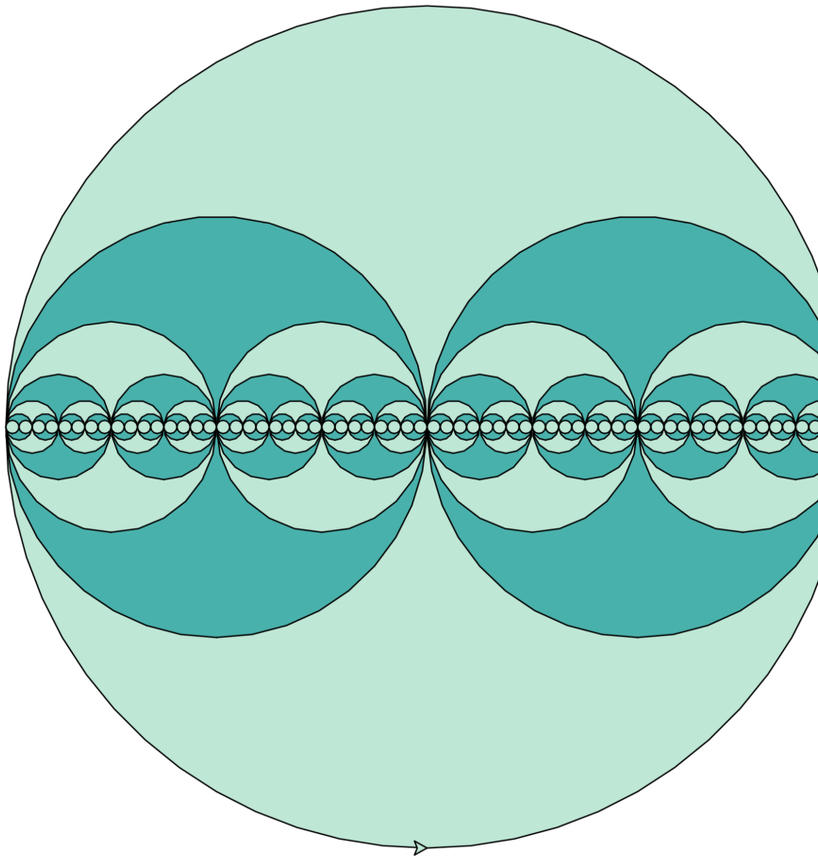
`nestedCircles(300, 75)`



`nestedCircles(300, 37)`



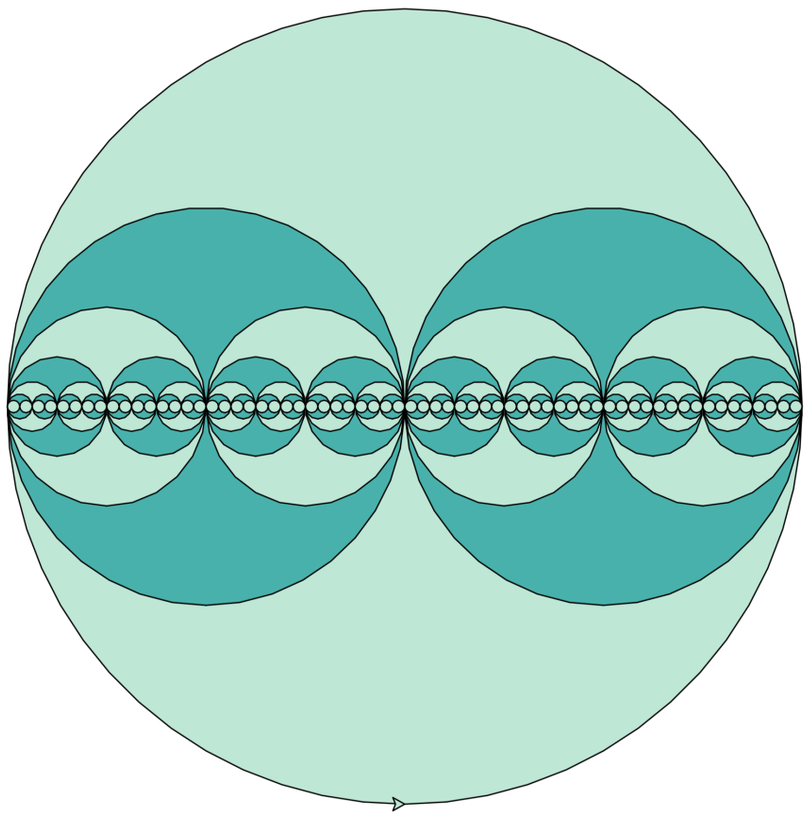
`nestedCircles(300, 9)`



`nestedCircles(300, 2)`

Invariants of Recursive Functions

- Why do we care about **invariants**?
  - Though not all functions have invariants that are necessary for correctness, not uncommon for recursive functions
  - In lab, our graphical functions will not always work properly if we do not maintain the function's "turtle position invariant"



The end!