

5. Introduction to Procedures

Topics:

The module SimpleGraphics

Creating and Showing figures

Drawing Rectangles, Disks, and Stars

Optional arguments

Application Scripts

Procedures

We continue our introduction to functions with a focus on procedures.

Procedures are functions that do not return a value.

Instead, they “do something.”

Graphics is a good place to illustrate the idea.

The Module SimpleGraphics Has Eight Procedures

SimpleGraphics.py

MakeWindow

ShowWindow

DrawRect

DrawDisk

DrawStar

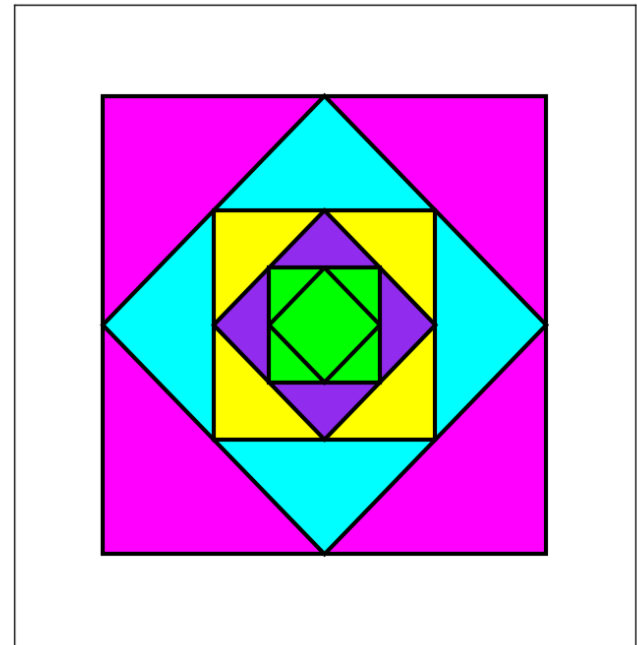
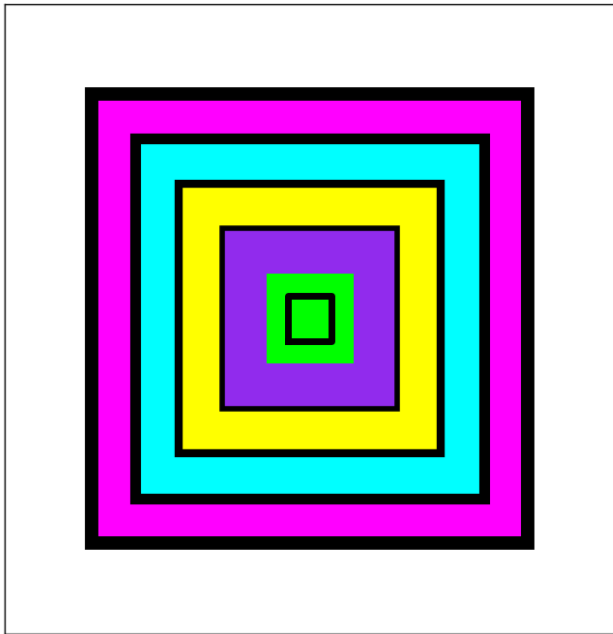
DrawLineSeg

DrawText

Title

We will use this module to make designs that involve rectangles, disks, stars, etc

Examples



Looks like we will be able to draw tilted rectangles

Example

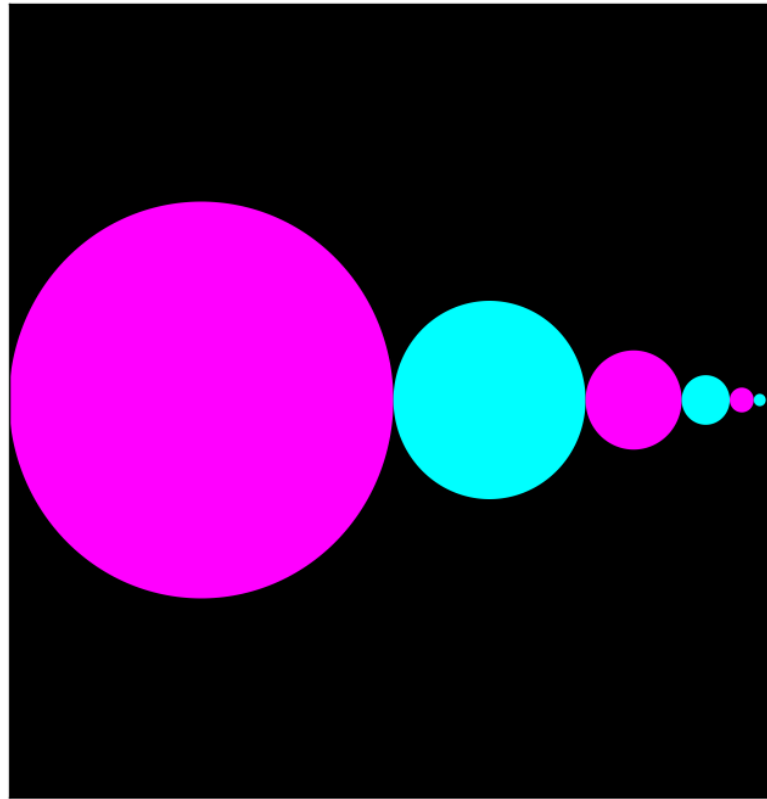
How
does
color
work?



What
if we
had
100 rows
each with
100 stars ?

Anticipating loops.

Example



Xeno's Paradox: Will we ever reach the right edge?

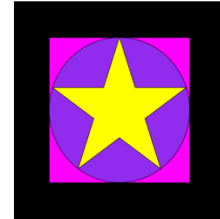
Example



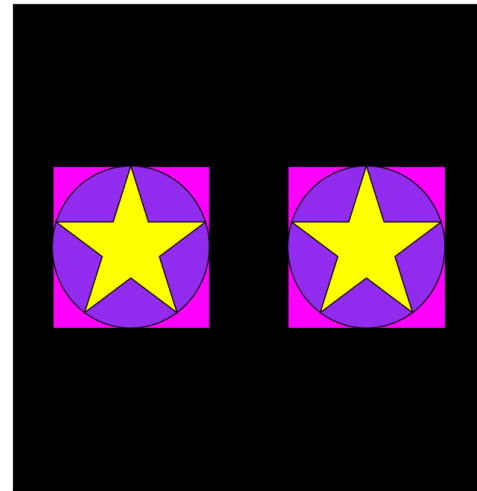
White Rectangle + Red Rectangle + White Disk + Red Disk + Tilted White Star

Example

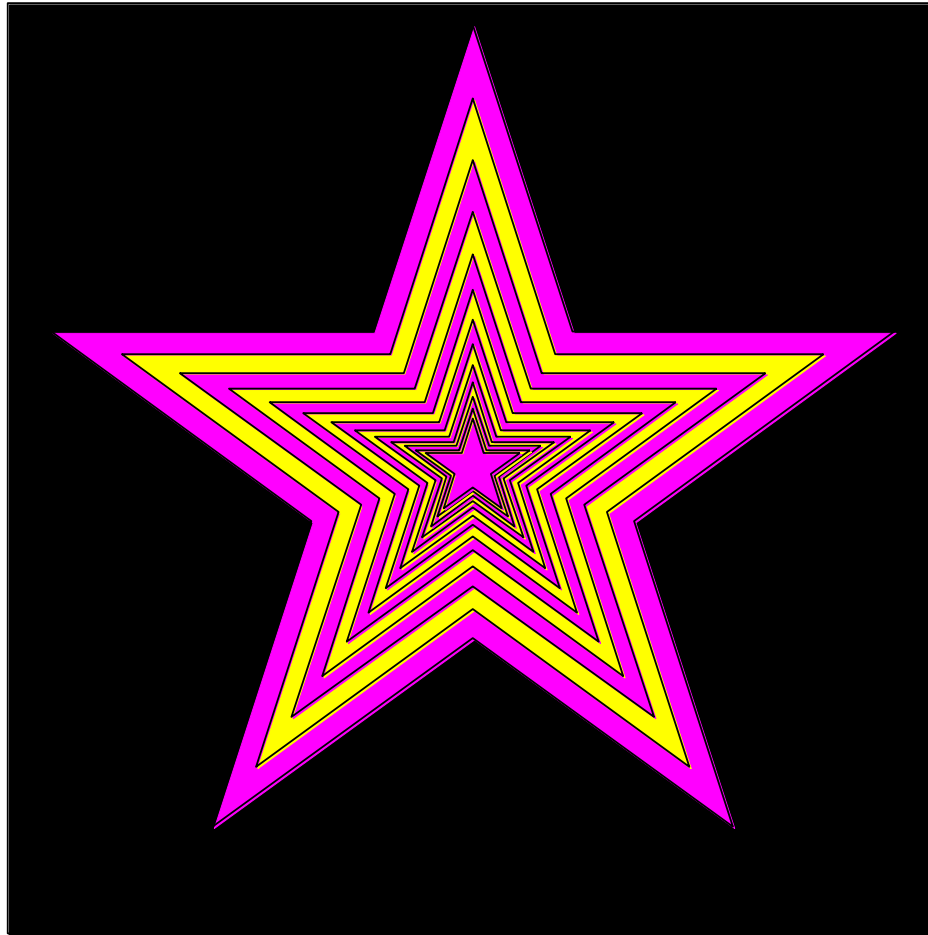
Let's write a function to draw this:



And then apply it two times:

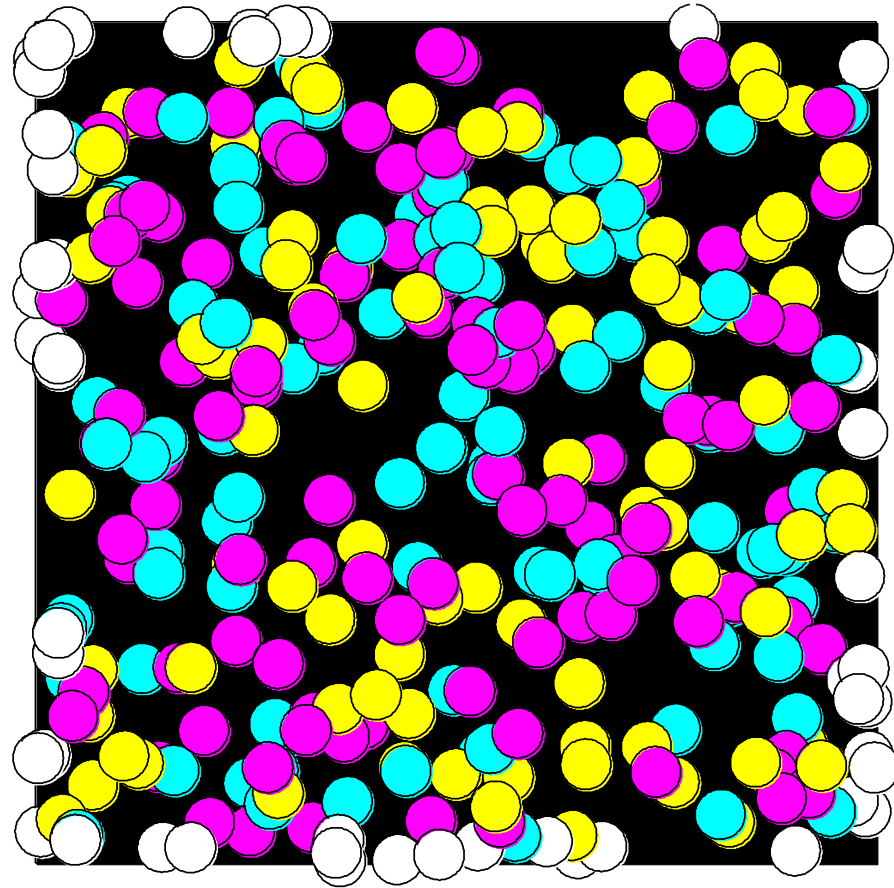


After We Learn About Iteration...



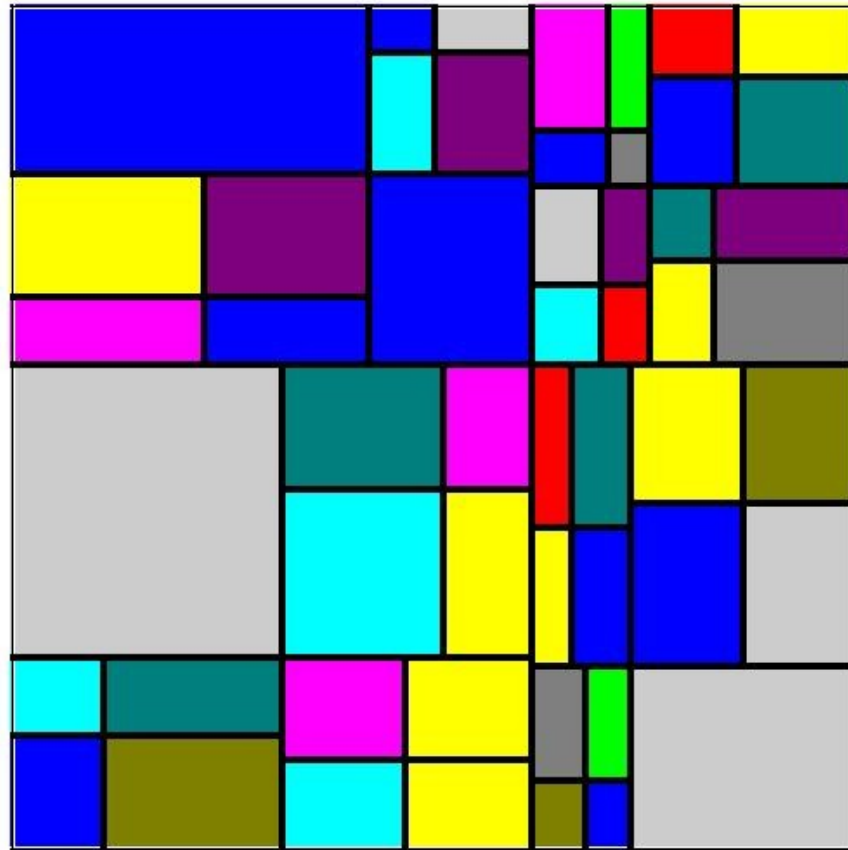
What if there were billions and billions of stars? Will need loops.

After We Learn About Iteration...



How long before the square is covered? Need loops.

After We Learn About Recursion...



Random Mondrian. Repeatedly cut a rectangle into 4 smaller rectangles.

A Quick Tour Through the SimpleGraphics Module

We now show how to use the eight procedures in SimpleGraphics:

MakeWindow

ShowWindow

DrawRect

DrawDisk

DrawStar

DrawLineSeg

DrawText

Title

Each of these procedures has several "options." We do not cover everything in the lecture slides. Labs and demo scripts cover these procedures in greater detail.

First: Create a Figure Window

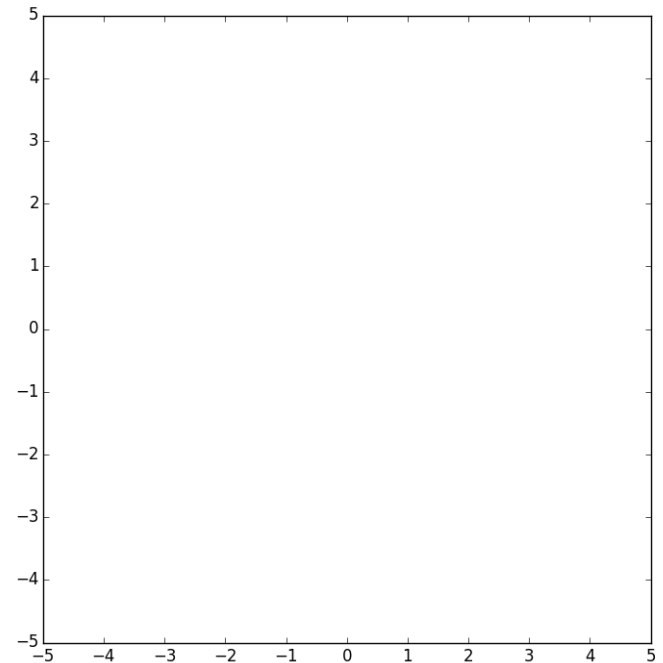
You cannot create any designs until you have a figure into which you can "drop" rectangles, disks, and stars.

MakeWindow

```
from SimpleGraphics import *  
n = 5  
MakeWindow(n)
```

Here we have created
a figure with labeled axes
that is ready to display
things in the square defined
by

$$-5 \leq x \leq +5, -5 \leq y \leq 5$$

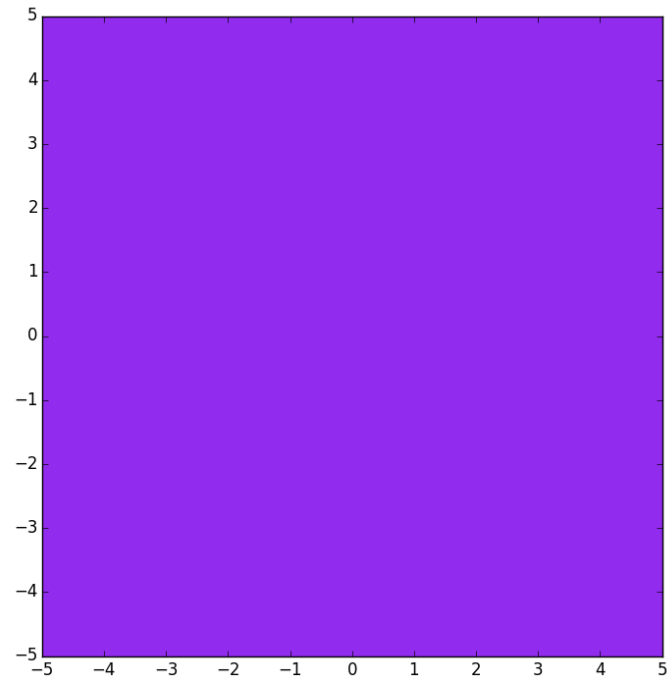


MakeWindow

```
from SimpleGraphics import*  
n = 5  
MakeWindow(n,bgcolor=PURPLE)
```

The "default" is to "paint"
the figure window white.

So this is what you must
do to set the background
color to something
different.

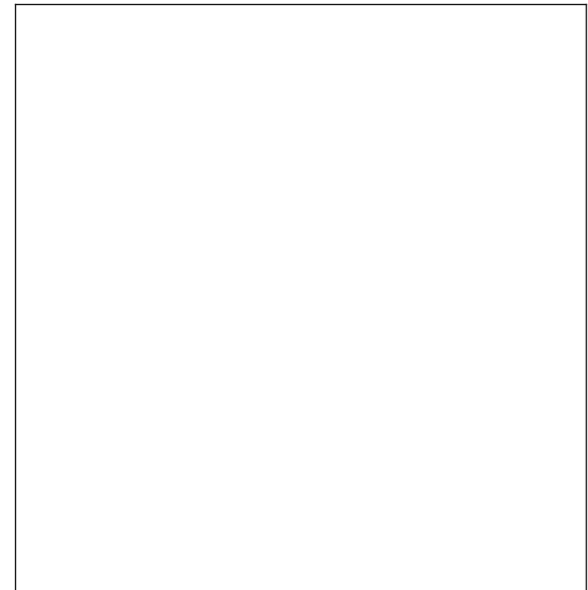


MakeWindow

```
from SimpleGraphics import*  
n = 5  
MakeWindow(n, labels=False)
```

The "default" is to label the axes.

So this is what you must do to suppress the labeling.



We are using import * to save space and because it is such a tiny module.

Color in simpleGraphics

The module has thirteen "built-in" colors.

If a SimpleGraphics procedure wants a color, just "hand over" one of these:

YELLOW

PURPLE

CYAN

ORANGE

RED

BLUE

GREEN

MAGENTA

PINK

WHITE

BLACK

LIGHTGRAY

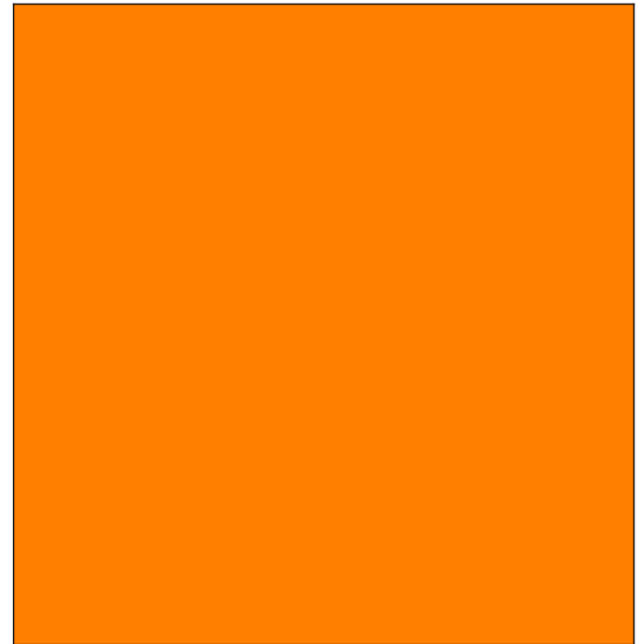
DARKGRAY

There is more flexibility than this. More later.

MakeWindow

```
from SimpleGraphics import*  
n = 5  
MakeWindow(n, labels=False, bgcolor=ORANGE)
```

You can turn off labeling
and specify a color
in the same call to
MakeWindow.



Optional Arguments

The function `MakeWindow` has four arguments.

Three of the arguments are "optional".

When there are several optional arguments, their order is immaterial. These are equivalent:

```
MakeWindow(n, labels=False, bgcolor=ORANGE)
```

```
MakeWindow(n, bgcolor=ORANGE, labels=False)
```

Note: You need the "assignment" for an optional argument.

This is illegal: `MakeWindow(5, False, ORANGE)`

Let's Draw a Rectangle with DrawRect

You must tell DrawRect

- the center of the rectangle.
- the horizontal dimension of the rectangle
- the vertical dimension of the rectangle

You have the option of telling DrawRect

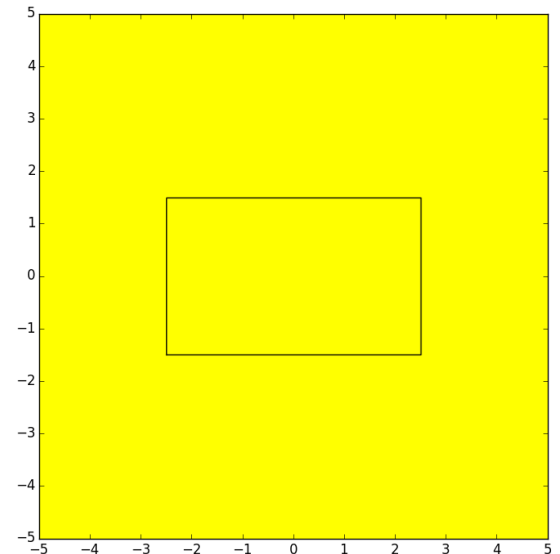
- the fill color
- the width of the perimeter highlight
- the color of the perimeter highlight
- the rotation angle

DrawRect

```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; L=5; W=3  
DrawRect(x,y,L,W)  
ShowWindow()
```

Nothing is actually displayed until this command is executed. More later.

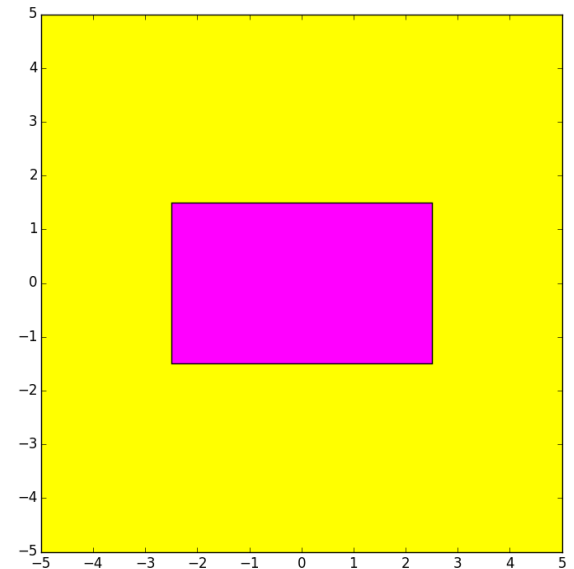
The default is a rectangle with no fill color. So all you get is the perimeter.



DrawRect

```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; L=5; W=3  
DrawRect(x,y,L,W,FillColor=MAGENTA)  
ShowWindow()
```

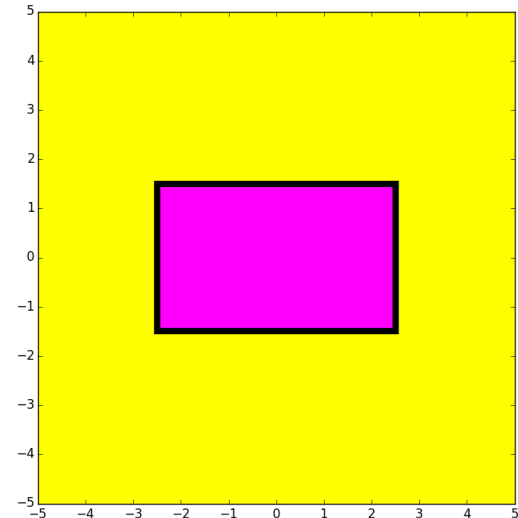
Use the optional color argument to specify a fill color.



DrawRect

```
from simpleGraphics import*
MakeWindow(5,bgcolor=YELLOW)
x=0; y=0; L=5; W=3
DrawRect(x,y,L,W,FillColor=MAGENTA,EdgeWidth=6)
ShowWindow()
```

Use the optional `EdgeWidth` argument to specify the boldness of the perimeter highlight. The default is `EdgeWidth = 1`

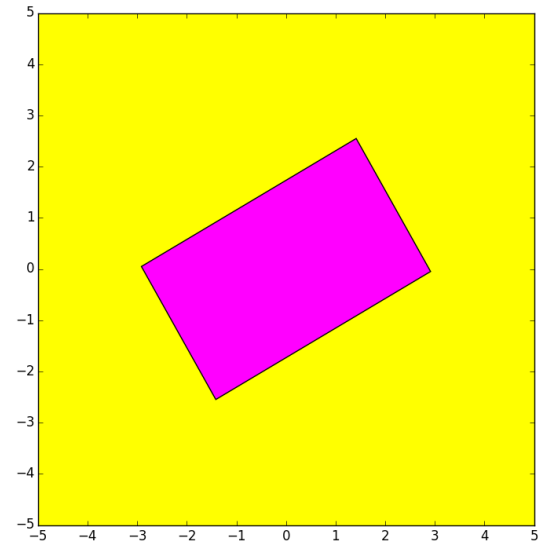


If you don't want any perimeter highlight, set `EdgeWidth=0`

DrawRect

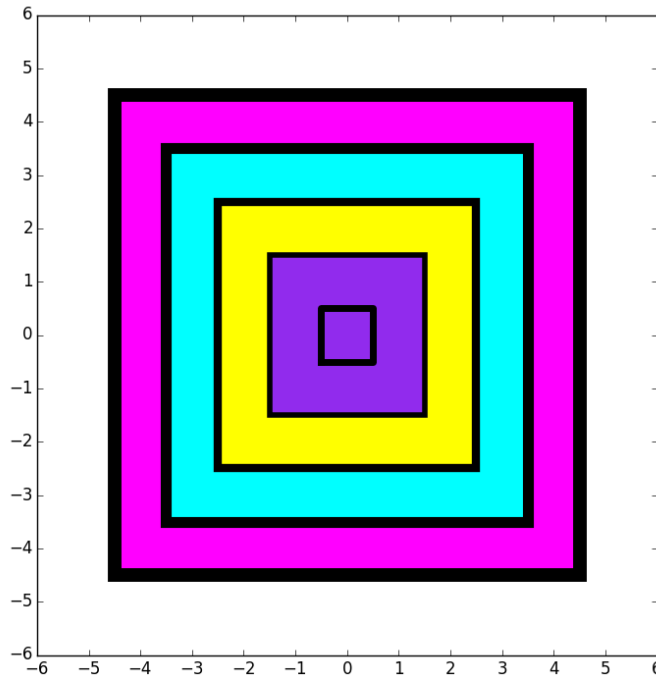
```
from SimpleGraphics import*
MakeWindow(5,bgcolor=YELLOW)
x=0; y=0; L=5; W=3
DrawRect(x,y,L,W,color=MAGENTA,theta=30)
ShowWindow()
```

Use the optional `theta` argument to specify the counterclockwise rotation of the rectangle about its center. (Angle in degrees.)



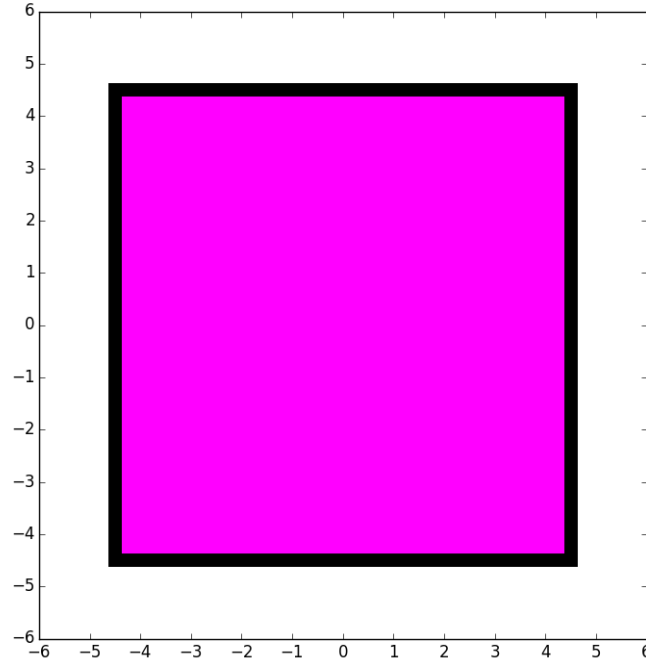
The default rotation angle is zero.

Let's Write a Script to Do This



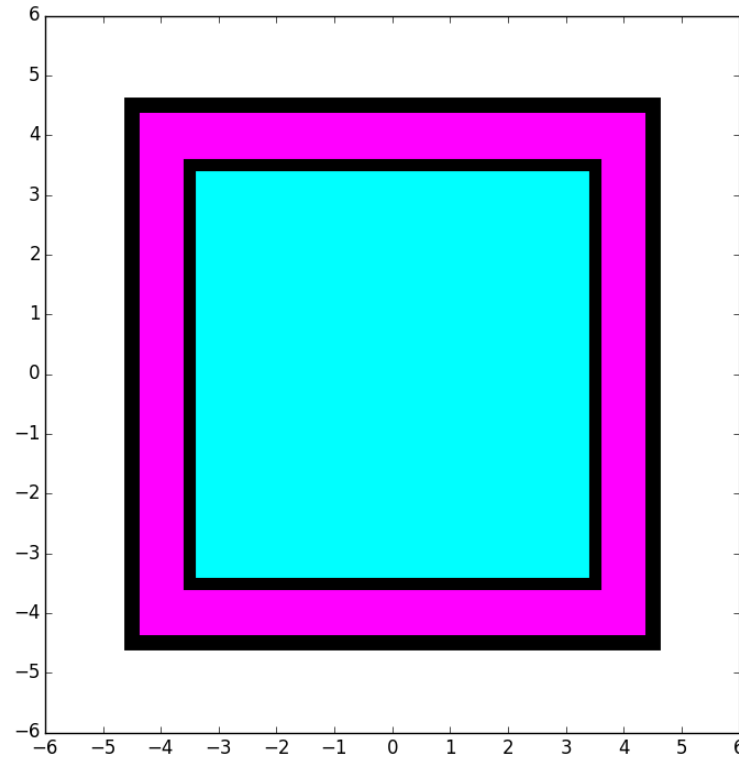
The squares are 9×9 , 7×7 , 5×5 , 3×3 , and 1×1 .

Nested Squares



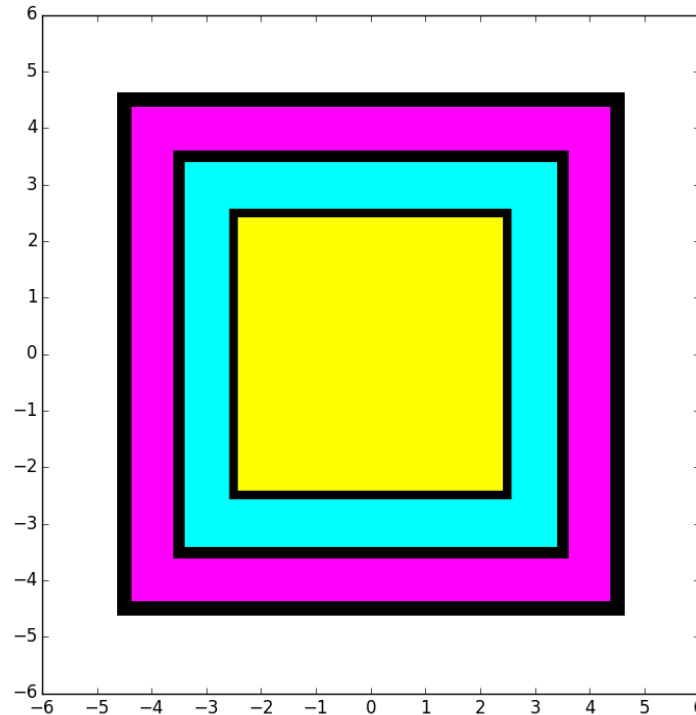
```
DrawRect(0,0,9,9,FillColor=MAGENTA,  
EdgeWidth=10)
```

DrawRect



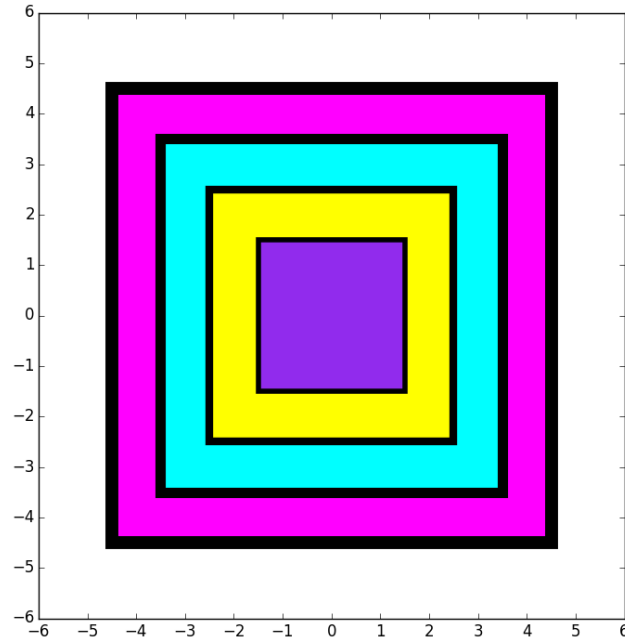
```
DrawRect(0,0,7,7,FillColor=CYAN,  
EdgeWidth=8)
```

Nested Squares



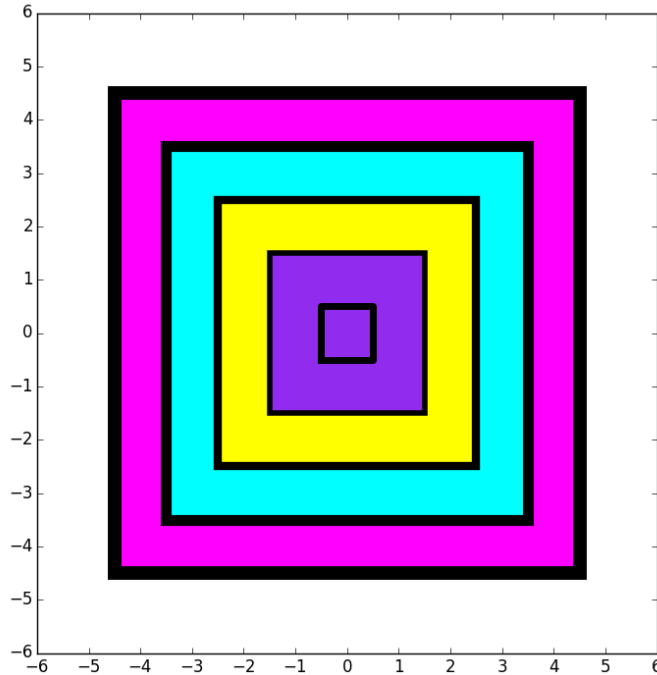
```
DrawRect(0, 0, 5, 5, FillColor=YELLOW,  
EdgeWidth=6)
```

DrawRect



```
DrawRect(0, 0, 3, 3, FillColor=PURPLE,  
EdgeWidth=4)
```

Nested Squares



`DrawRect (0 , 0 , 1 , 1 , EdgeWidth=5)`

Nested Squares

```
MakeWindow(6,bgcolor=WHITE)

DrawRect(0,0,9,9,FillColor=MAGENTA,
          EdgeWidth=10)
DrawRect(0,0,7,7,FillColor=CYAN,
          EdgeWidth=8)
DrawRect(0,0,5,5,FillColor=YELLOW,
          EdgeWidth=6)
DrawRect(0,0,3,3,FillColor=PURPLE,
          EdgeWidth=4)
DrawRect(0,0,1,1,EdgeWidth=5)

ShowWindow()
```

Let's Draw a Disk with DrawDisk

You must tell DrawDisk

- the center of the disk.
- the radius of the disk

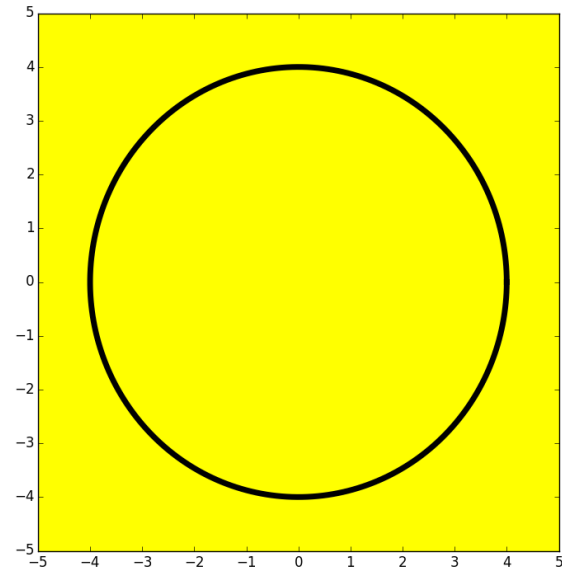
You have the option of telling DrawDisk

- the fill color
- the width of the perimeter highlight
- the color of the perimeter highlight

DrawDisk

```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; r=4  
DrawDisk(x,y,r)  
ShowWindow()
```

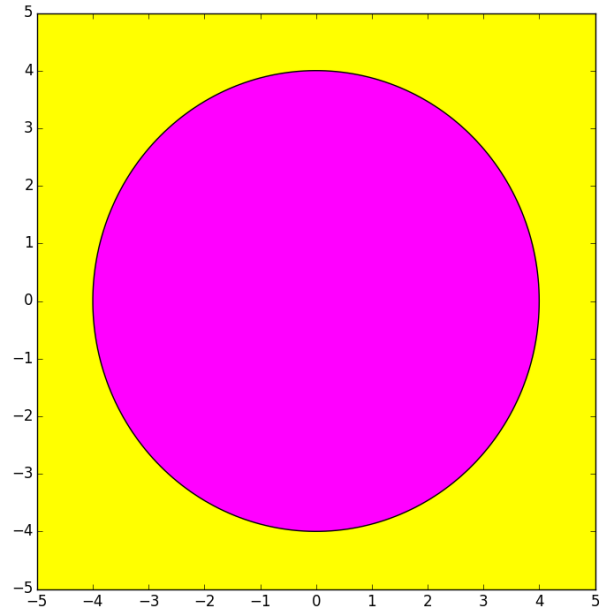
The default is a circle with no fill color. So all you get is the perimeter.



DrawDisk

```
from SimpleGraphics import*
MakeWindow(5,bgcolor=YELLOW)
x=0; y=0; r=4
DrawDisk(x,y,r,FillColor=MAGENTA)
ShowWindow()
```

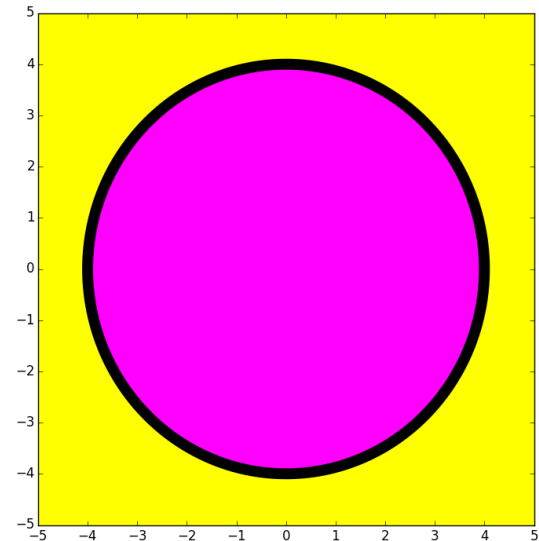
Use the optional color argument to specify a fill color.



DrawDisk

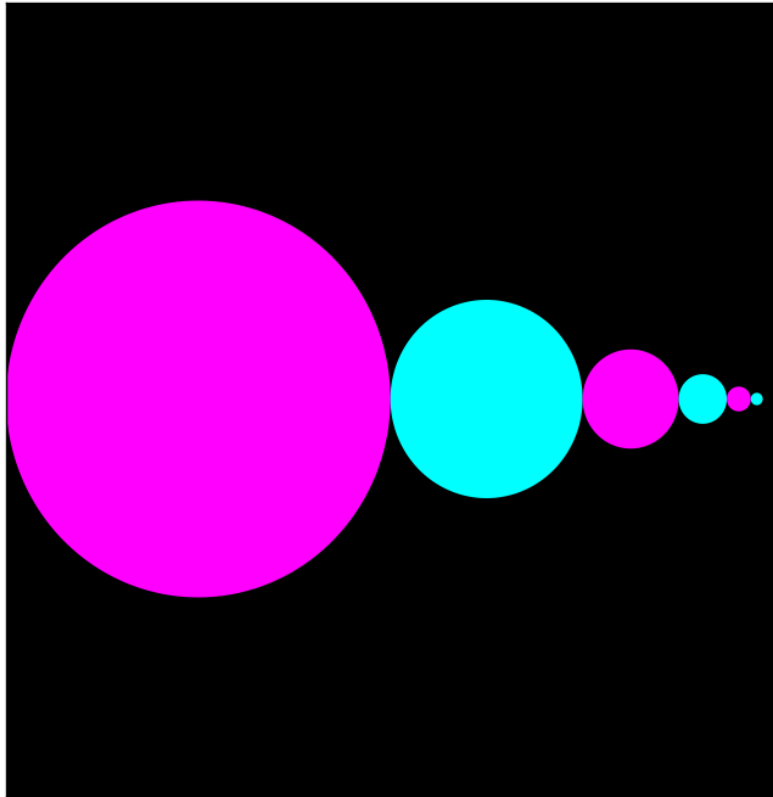
```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; r=4  
DrawDisk(x,y,r,FillColor=MAGENTA,EdgeWidth=6)  
ShowWindow()
```

Use the optional `EdgeWidth` argument to specify the boldness of the perimeter highlight. The default is `EdgeWidth = 1`



If you don't want any perimeter highlight, set `EdgeWidth=0`

Let's Draw This



Rules:

Big circle center at $(-4,0)$
with radius 4.

Circles are tangent to each
other. Centers on x-axis.

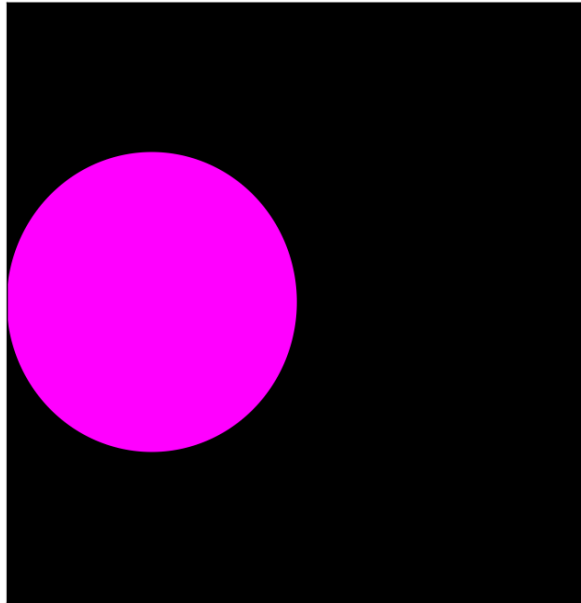
Each circle has half the
radius of its left neighbor.

Draw the First Disk

```
x = -4
```

```
r = 4
```

```
DrawDisk(x, 0, r, FillColor=MAGENTA, EdgeWidth=0)
```

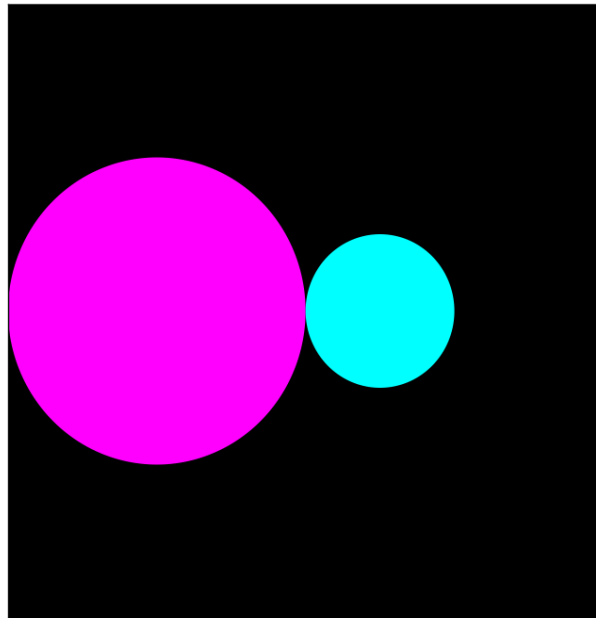


Draw the Second Disk

```
x = x + 1.5*r
```

```
r = r/2
```

```
DrawDisk(x, 0, r, FillColor=CYAN, EdgeWidth=0)
```

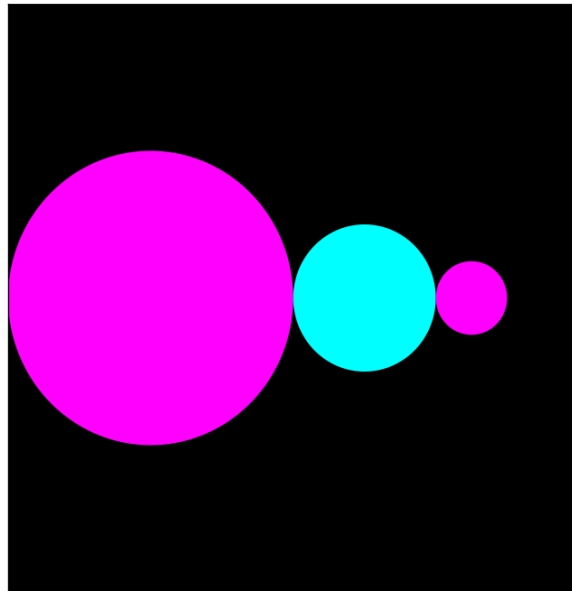


Draw the Third Disk

```
x = x + 1.5*r
```

```
r = r/2
```

```
DrawDisk(x, 0, r, FillColor=MAGENTA, EdgeWidth=0  
)
```



Overall

```
x = -4; r = 4
```

```
DrawDisk(x, 0, r, FillColor=MAGENTA, EdgeWidth=0)
```

```
x = x + 1.5*r; r = r/2
```

```
DrawDisk(x, 0, r, FillColor=CYAN, EdgeWidth=0)
```

```
x = x + 1.5*r; r = r/2
```

```
DrawDisk(x, 0, r, FillColor=MAGENTA, EdgeWidth=0)
```

```
x = x + 1.5*r; r = r/2
```

```
DrawDisk(x, 0, r, FillColor=CYAN, EdgeWidth=0)
```

Notice the repetition of the x and r updates. Simpler than figuring the centers and radii "by hand". Also gets us ready for loops.

Let's Draw a Star with DrawStar

You must tell DrawStar

- the center of the star.
- the radius of the star

You have the option of telling DrawStar

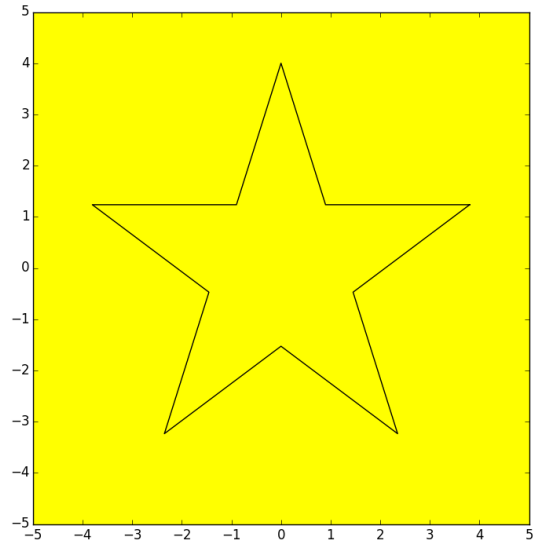
- the fill color
- the width of the perimeter highlight
- the color of the perimeter highlight
- the rotation angle

DrawStar

```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; r=4  
DrawStar(x,y,r)  
ShowWindow()
```

The default is a star with no fill color. So all you get is the perimeter.

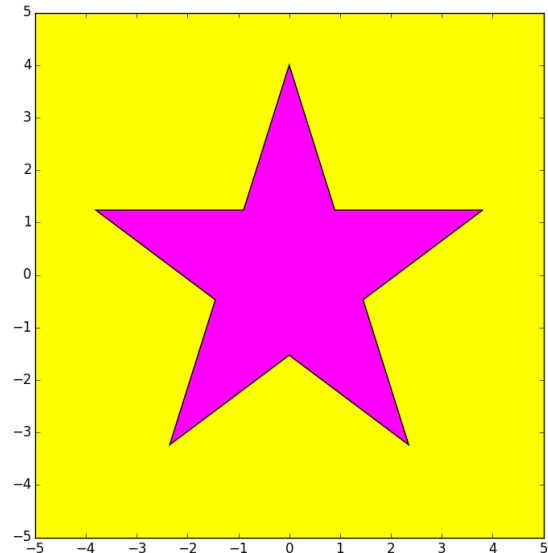
Note: the radius of a star is the distance from its center to any tip.



DrawStar

```
from SimpleGraphics import*
MakeWindow(5,bgcolor=YELLOW)
x=0; y=0; r=4
DrawStar(x,y,r,FillColor=MAGENTA)
ShowWindow()
```

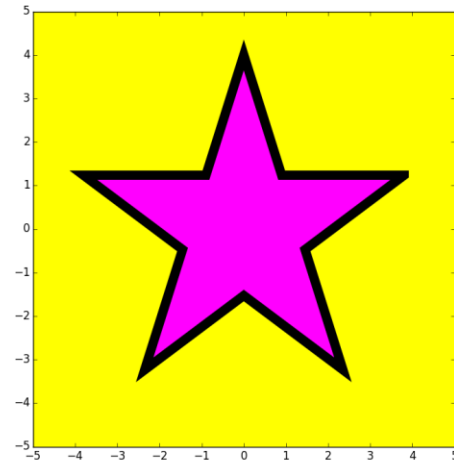
Use the optional color argument to specify a fill color.



DrawStar

```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; r=4  
DrawStar(x,y,r,FillColor=MAGENTA,EdgeWidth=6)  
ShowWindow()
```

Use the optional `EdgeWidth` argument to specify the boldness of the perimeter highlight. The default is `EdgeWidth = 1`

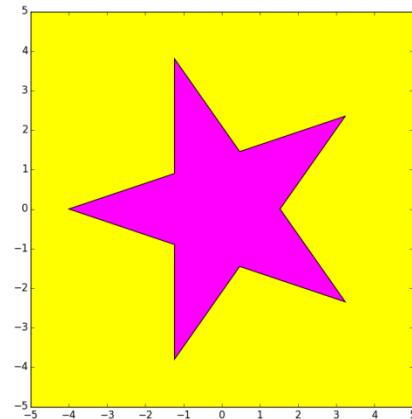


If you don't want any perimeter highlight, then set `EdgeWidth=0`

DrawStar

```
from SimpleGraphics import*  
MakeWindow(5,bgcolor=YELLOW)  
x=0; y=0; r=4  
DrawStar(x,y,r,FillColor=MAGENTA,theta=18)  
ShowWindow()
```

Use the optional `theta` argument to specify the counterclockwise rotation of the rectangle about its center. (Angle in degrees.)



The default rotation angle is zero.

Let's Draw a Line Segment with DrawLineSeg

You must tell DrawLineSeg

- the first endpoint of the segment
- the second endpoint of the segment

You have the option of telling DrawLineSeg

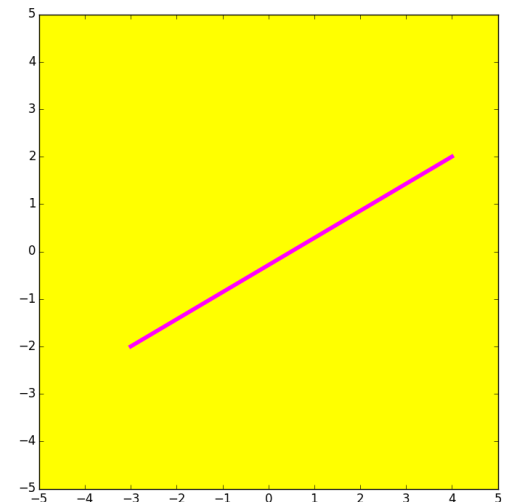
- the color of the segment
- the line width of the segment

DrawLineSeg

```
from SimpleGraphics import*
MakeWindow(5,bgcolor=YELLOW)
a = -3; b = -2; c = 4; d = 2
DrawLineSeg(a,b,c,d,LineWidth=4,
            LineColor='MAGENTA')
ShowWindow()
```

The default line color is BLACK.

The default line width is 1.



Let's "Draw" Text with DrawText

You must tell DrawText

- the location of the text.
- the text (a string) that is to be displayed

You have the option of telling DrawText

- the color of the text
- the size of the font

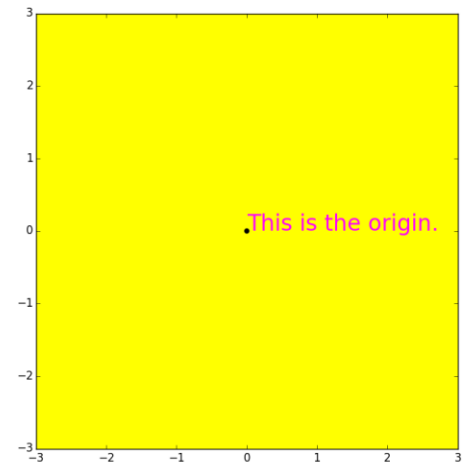
DrawText

```
from SimpleGraphics import*
MakeWindow(3,bgcolor=YELLOW)
x=0; y=0; s = 'This is the origin.'
DrawText(x,y,s,FontSize=24,FontColor='MAGENTA')
DrawDisk(0,0,.03,FillColor=BLACK)
ShowWindow()
```

The default text color is BLACK.

The default font size is 10.

The lower left corner of the first character is roughly at (x,y).



Let's Talk About Color



The rgb Representation

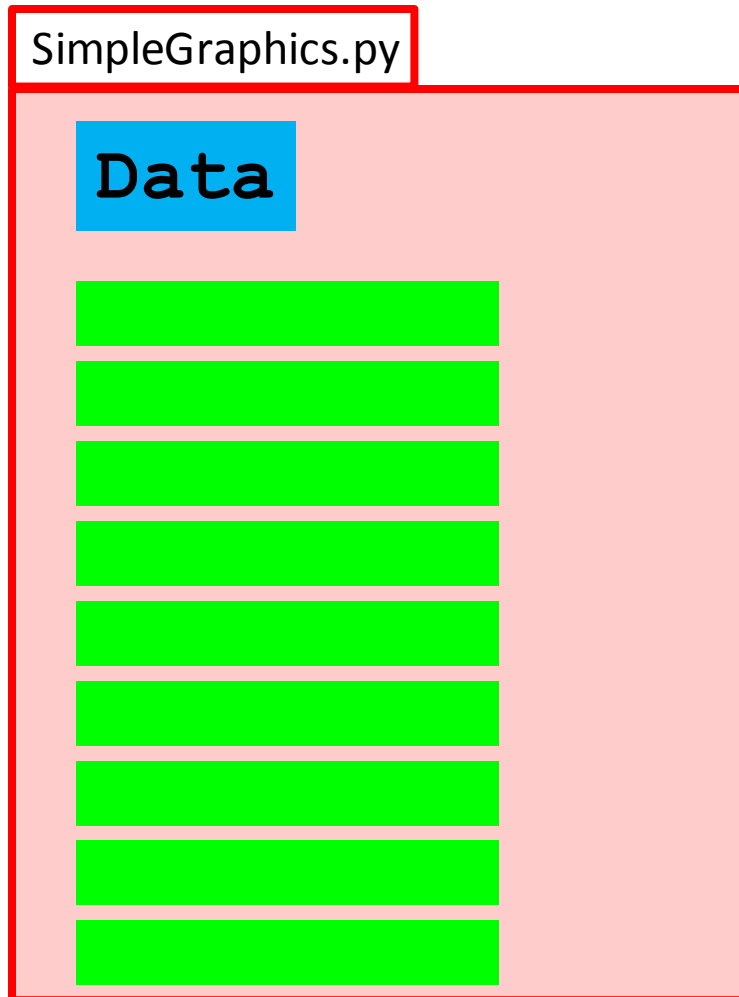
A color is a triple of numbers, each between zero and one.

The numbers represent the amount of red, green, and blue.

This is purple:

[0.57 , 0.17 , 0.93]

The Module SimpleGraphics Has 8 Procedures and Data



In this case
the data
encodes the
"rgb" values
of thirteen
colors

The SimpleGraphics Colors

YELLOW	=	[1.00, 1.00, 0.00]
CYAN	=	[0.00, 1.00, 1.00]
MAGENTA	=	[1.00, 0.00, 1.00]
RED	=	[1.00, 0.00, 0.00]
GREEN	=	[0.00, 1.00, 0.00]
BLUE	=	[0.00, 0.00, 1.00]
WHITE	=	[1.00, 1.00, 1.00]
BLACK	=	[0.00, 0.00, 0.00]
PURPLE	=	[0.57, 0.17, 0.93]
DARKGRAY	=	[0.33, 0.33, 0.33]
LIGHTGRAY	=	[0.67, 0.67, 0.67]
ORANGE	=	[1.00, 0.50, 0.00]
PINK	=	[1.00, 0.71, 0.80]

These are called
"Global Variables"

Convention: Global Variable Names should be UPPER CASE.

Access

```
from SimpleGraphics import*
MakeWindow(5,bgcolor=YELLOW)
x=0; y=0; L=5; W=3
DrawRect(x,y,L,W,FillColor=MAGENTA)
ShowWindow()
```

When a module is imported, it gives access to both its functions and its global variables.

rgb Lists

Things like `[0.74, 1.00, 0.34]` are called rgb lists.

Rules: Square brackets, 3 numbers separated by commas, each number between 0 and 1.

First number = red value

Second number = green value

Third number = blue value

The bigger numbers mean more of that color.

Using rgb Lists

Instead of using the predefined colors you can make up and use your own fill color, e.g.

```
c = [0.74, 1.00, 0.34]
```

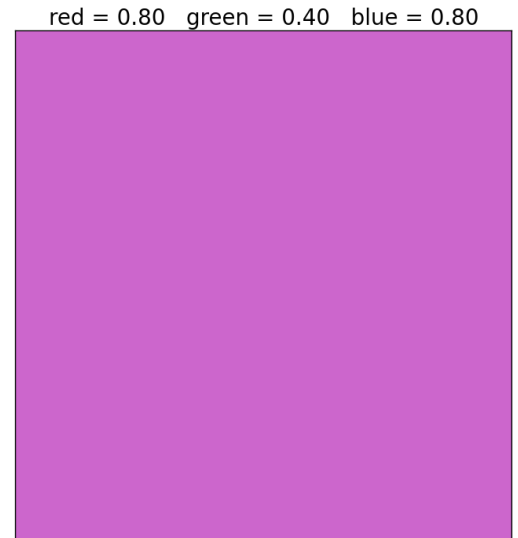
```
DrawDisk(0, 0, 1, FillColor=c)
```

Google "rgb values" to look at huge tables of colors and rgb values.

Title

```
from SimpleGraphics import*  
r=0.8,g=0.4, b=0.8  
MakeWindow(5,bgcolor=[r,g,b])  
s = `r=%4.2f  g=%4.2f  b = %4.2f` % (r,g,b)  
Title(s,FontSize=20)  
ShowWindow()
```

You can put a title
at the top of the
figure window.



A Note on Managing Figures

MakeWindow (etc)



MakeWindow (etc)



MakeWindow (etc)



ShowWindow ()

Three figure windows will be produced.

The green code defines what is in the first window.

The pink and blue code set up the second and third windows.

The ShowWindow says. "Show all the windows."

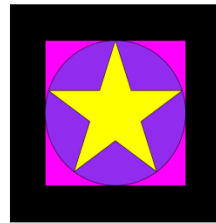
A Final Example

Shows two things.

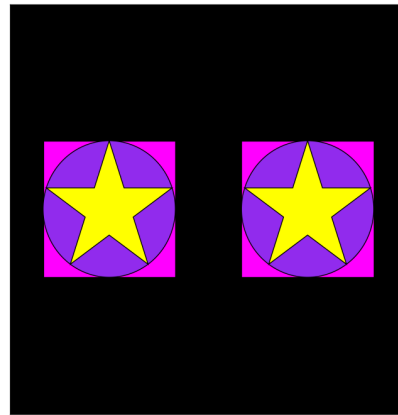
1. You can write a module that uses other modules that YOU have written.
2. You can have a module that has both function definitions and a script that can be executed.

A Final Example

We write a procedure to draw this



and a script that calls it twice:



We put them both in the *SAME* module....

A Final Example

Tile.py

```
from simpleGraphics import *

def DrawTile(x,y,r,c1,c2,c3):
    DrawRect(x,y,2*r,2*r,FillColor=c1)
    DrawDisk(x,y,r,FillColor=c2)
    DrawStar(x,y,r,FillColor=c3)

if __name__ == '__main__':

    MakeWindow(6,bgcolor=BLACK,labels=False)
    DrawTile(3,0,2,MAGENTA,PURPLE,YELLOW)
    DrawTile(-3,0,2,MAGENTA,PURPLE,YELLOW)
    ShowWindow()
```

See the Demo `Tile.py` In command mode, enter `python Tile.py`

A Final Example

Tile.py

```
from SimpleGraphics import *

def DrawTile(x,y,r,c1,c2,c3):
    DrawRect(x,y,2*r,2*r,color=c1)
    DrawDisk(x,y,r,color=c2)
    DrawStar(x,y,r,color=c3)

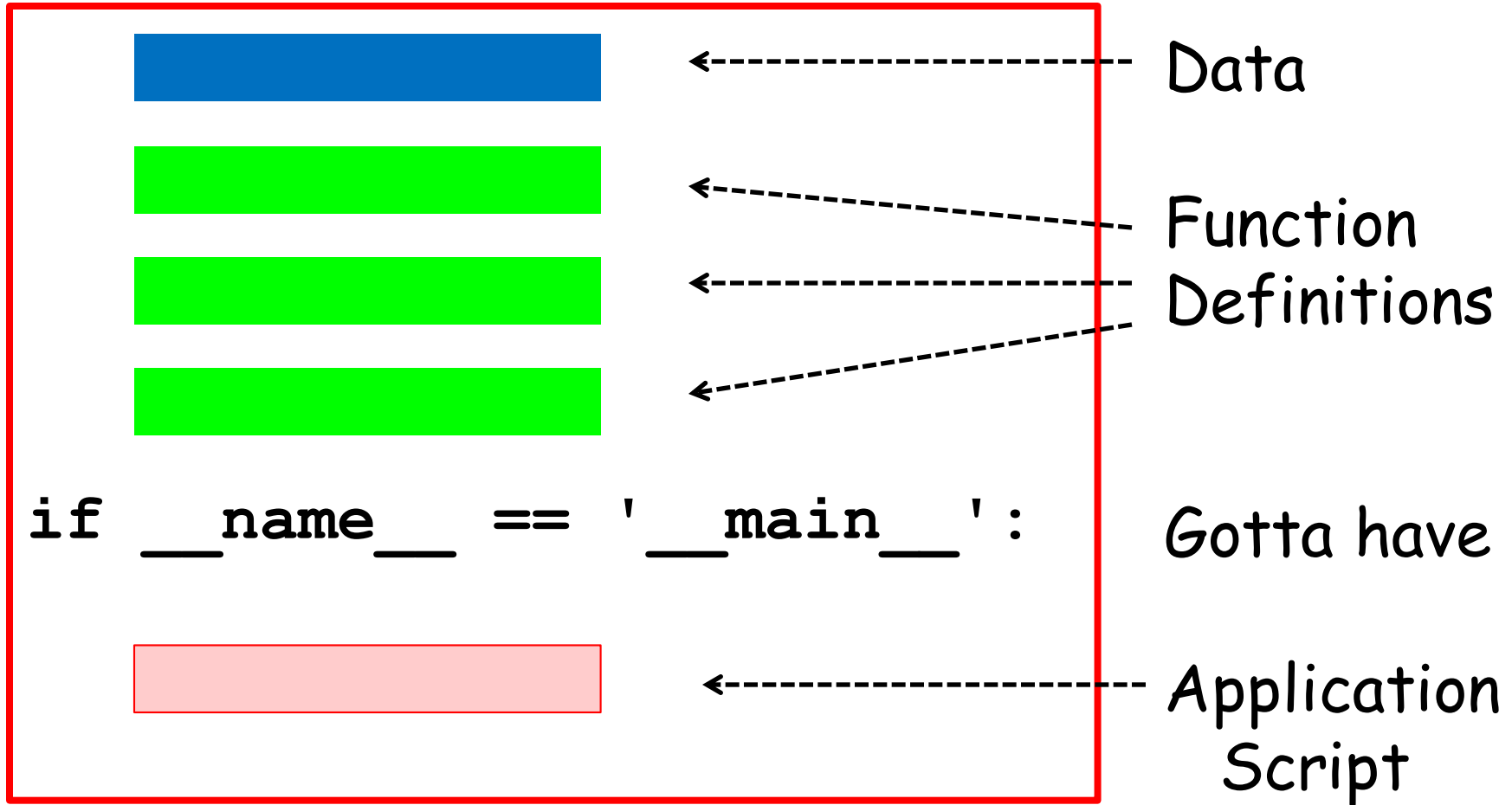
if __name__ == '__main__':
```

This is called
an “Application
Script”

```
    MakeWindow(6,bgcolor=BLACK,labels=False)
    DrawTile(3,0,2,MAGENTA,PURPLE,YELLOW)
    DrawTile(-3,0,2,MAGENTA,PURPLE,YELLOW)
    ShowWindow()
```

See the demo `Tile.py` In command mode, enter `python Tile.py`

So a Module Can Look Like This



Those are "double underscores" in the if statement.

Summary

1. Procedures "look like" functions without the "return." They "do stuff" but do not return values
2. Graphics procedures were used to illustrate the idea.
3. Color can be encoded with three numbers that indicate the amount of red, green, and blue.
4. A single module can house data, functions, and a script at the same time