Drawing with Turtle

Lecture 04.04

By Marina Barsky

All code examples:

turtle_setup.py
red_square.py
polygon.py
birthday_cake.py
star_turtle_algorithm.py
flower.py
circles.py
squares.py
nested_star_loop.py

Sketch Pad

- Want graphics? In Python, we give commands to a "turtle" to draw on a digital canvas!

```
import turtle
```

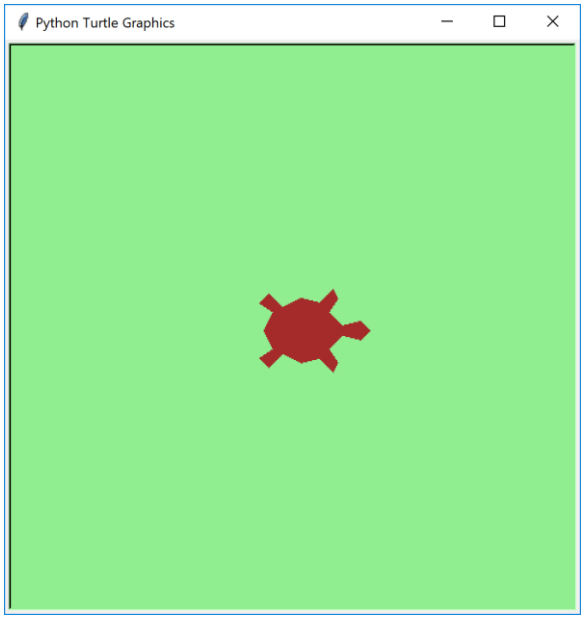
Make sure that your program is not called *turtle.py*, and there is no program called *turtle.py* in your project

Our first turtle

```
import turtle

screen = turtle.Screen()          # get a screen to draw on
screen.setup(500,500,40,50)       # width, hight, left, top
screen.bgcolor("lightgreen")

alex = turtle.Turtle()   # create artistic turtle
alex.shape("turtle")
alex.shapesize(4)
alex.color("brown")

turtle.done()
```

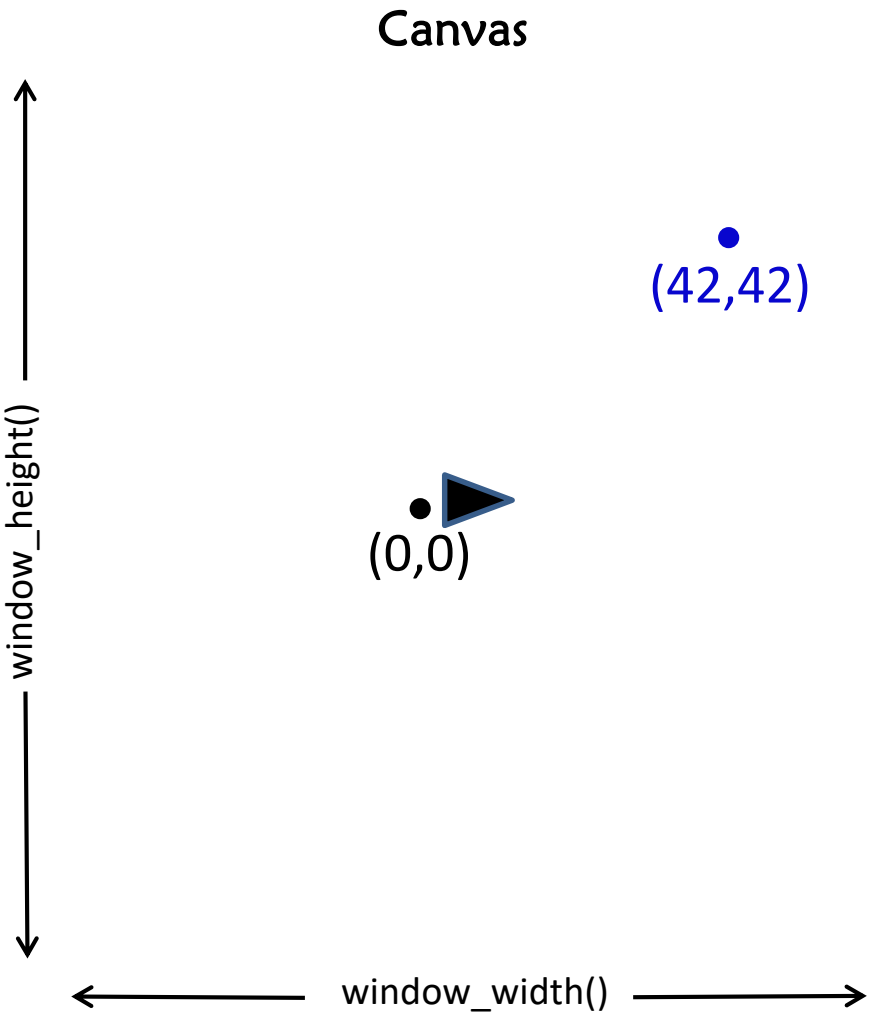
OR

```
turtle.exitonclick()
```



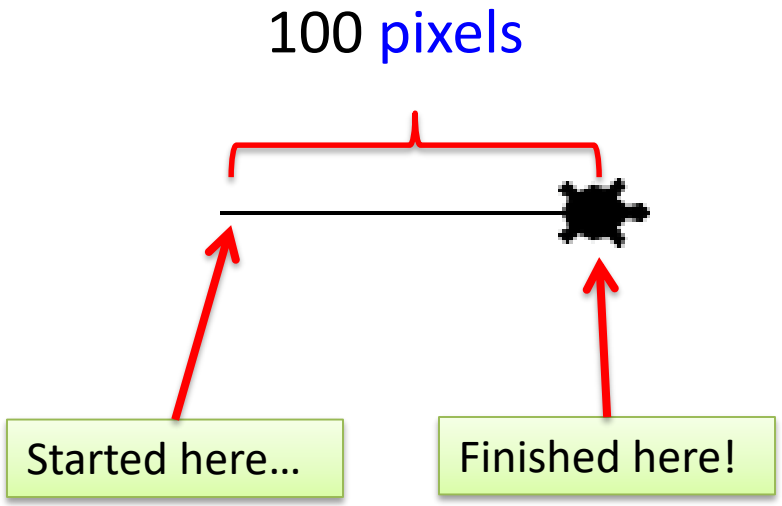
Coordinate system

- Canvas operates in x-y coordinate plane
  - (0,0) is the center
- **`alex.reset()`**
  - Delete any drawings, reset the screen, re-center the turtle
  - Turtle resets to face right (or east)



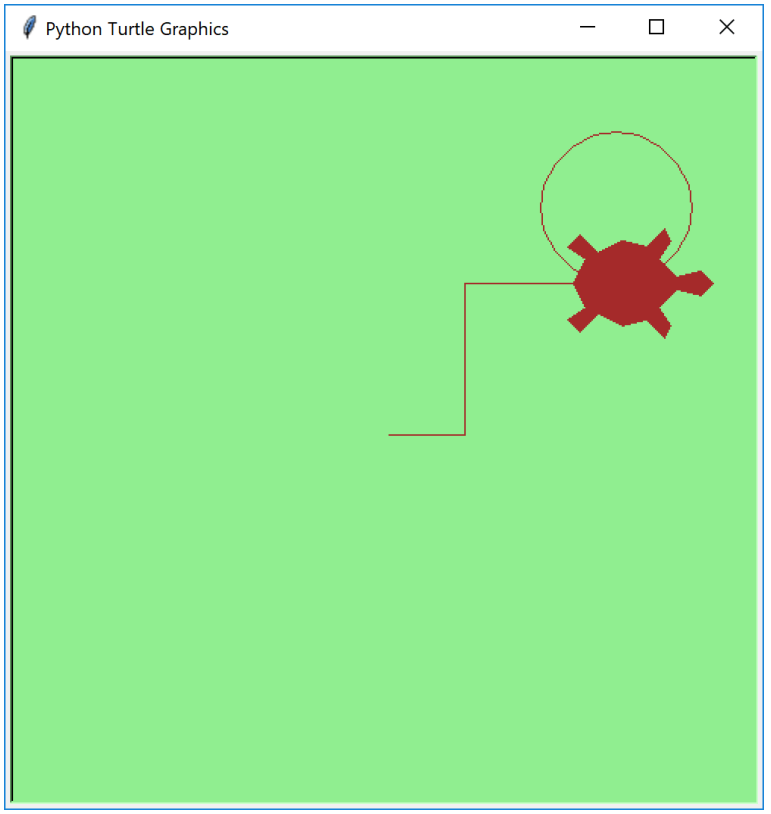
Moving turtle (and drawing)

```
# … setup
alex = turtle.Turtle()
alex.reset()  # home()
alex.forward(100)
```



Sketching turtle path

```
alex.forward(50)
alex.left(90)
alex.forward(100)
alex.right(90)
alex.forward(100)
alex.circle(50)
```



turtle_setup.py

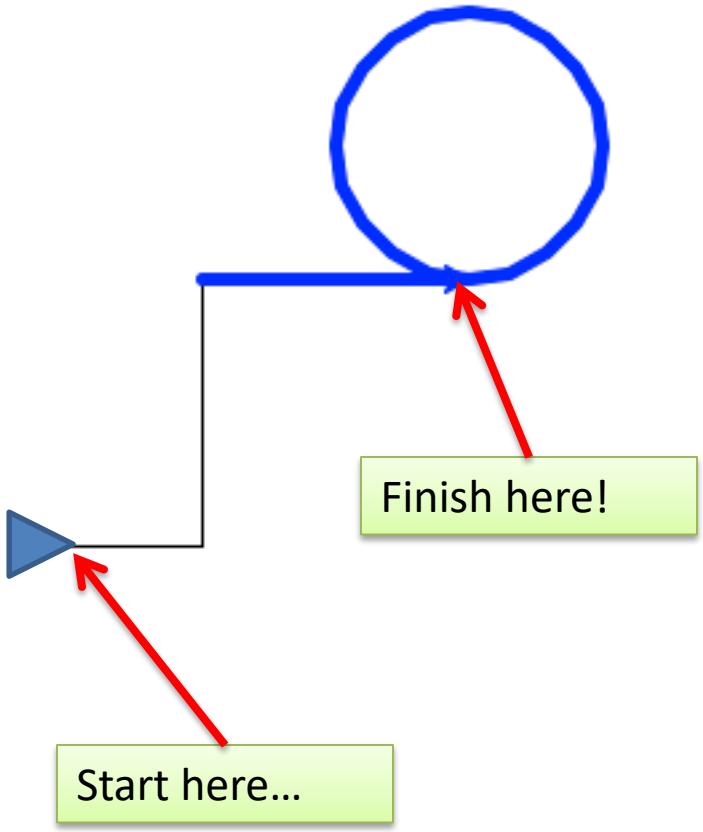
Explanation

```
alex.forward(50)
```
In pixels
NOTE: **`backward(n)`** moves the turtle back

```
alex.left(90)
```
In degrees

```
alex.forward(100)
alex.right(90)
```
In degrees

```
alex.forward(100)
alex.circle(50)
```
Starts drawing circle to the left of the turtle
Radius is specified In pixels

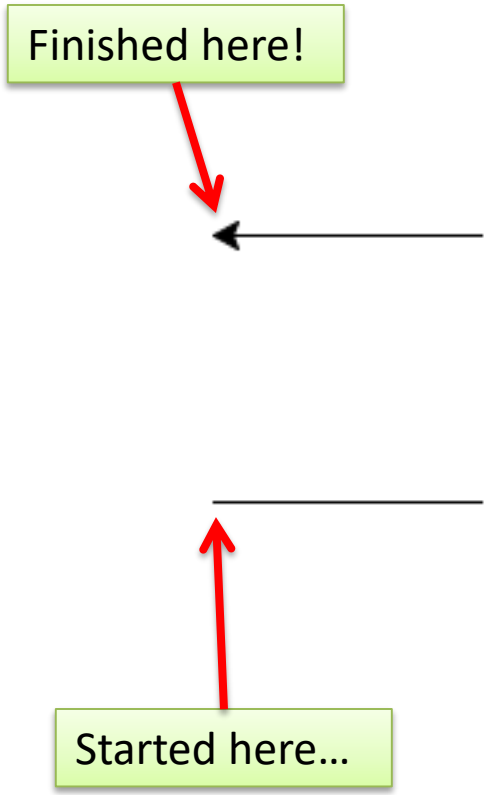


Pen up, Pen down

```
alex.reset()
alex.fd(100)
alex.lt(90)
alex.up()
alex.fd(100)
alex.lt(90)
alex.down()
alex.fd(100)
turtle.done()
```

Explanation

```
alex.reset()
alex.fd(100)
```
    Same as `forward()`
```
alex.lt(90)
```
    Same as `left()`
```
alex.up()
```
    Lifts the pen off the canvas
```
alex.fd(100)
alex.lt(90)
alex.down()
```
    Puts the pen down on the canvas
```
alex.fd(100)
turtle.done()
```



Turtle Graphics

```
import turtle
pen = turtle.Turtle()
```

You may need some more commands:

- **`pen.dot()`**
- **`pen.setheading(to_angle)`**
  - Set the orientation of the turtle to *to_angle*
- **`pen.tracer(0,0)`**
  - Turn turtle animation on/off
- **`pen.speed(0)`**

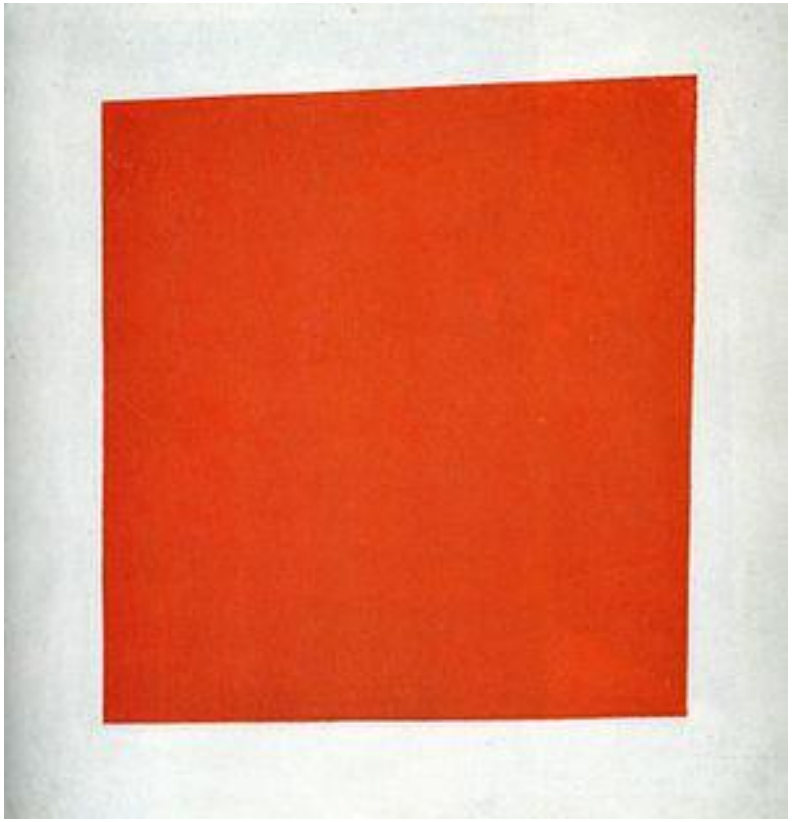
0 - east
90 - north
180 - west
270 - south

"fastest": 0
"fast": 10
"normal": 6
"slow": 3
"slowest": 1

Turtle reference https://docs.python.org/3.6/library/turtle.html

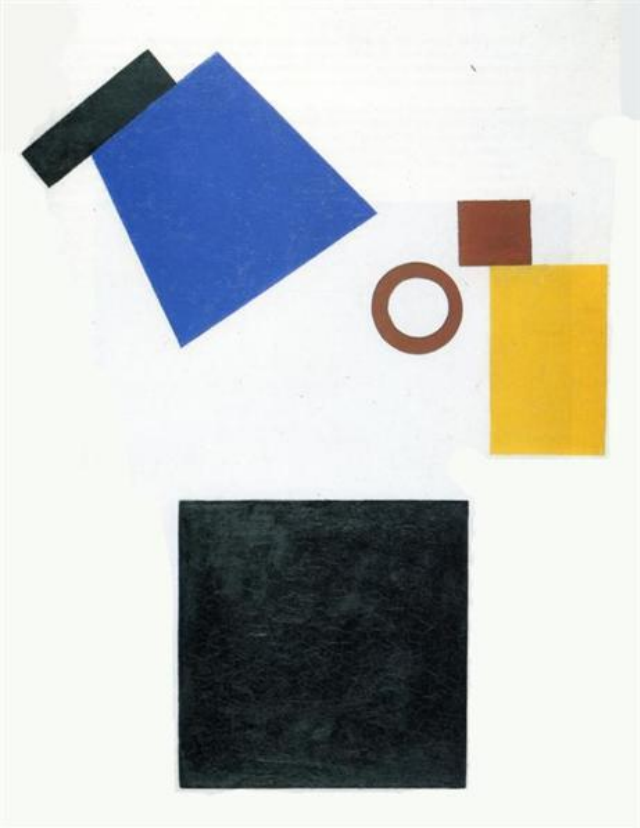
Red square by Kazimir Malevich, 1915

- *Suprematism* believed in the radical reduction of painting to nothing but shape and color
- Paintings would depict nothing, state nothing, **resist all aesthetic conventions**
- They would **spring free** as the revolution itself

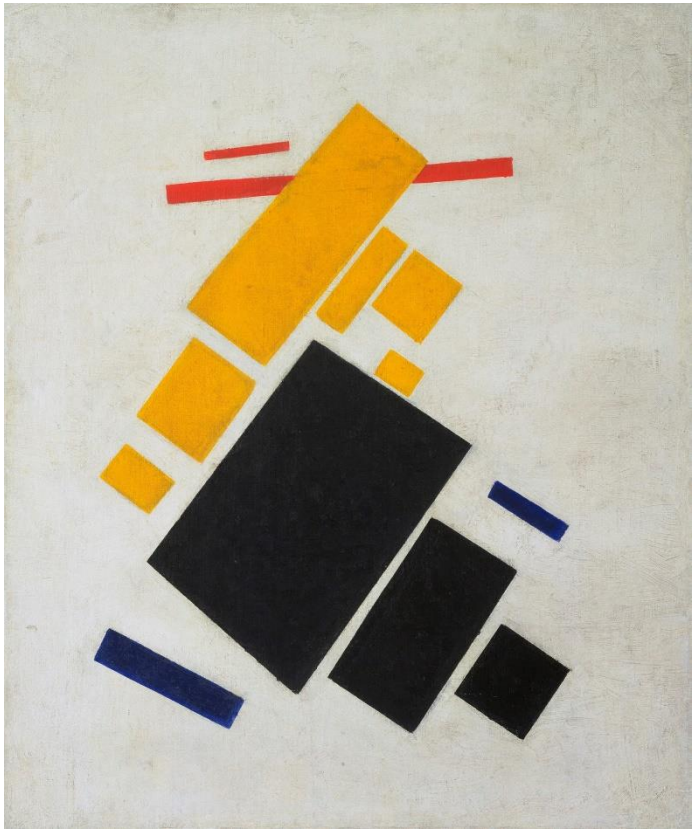


Red Square:
Painterly Realism of a Peasant Woman in Two Dimensions

Suprematism: shapes and colors



Two Dimensional Self Portrait



Airplane Flying

Artistic turtle

```
import turtle

screen = turtle.Screen()
screen.setup(500,500)
screen.colormode(255)
screen.bgcolor(221,226,222)
```

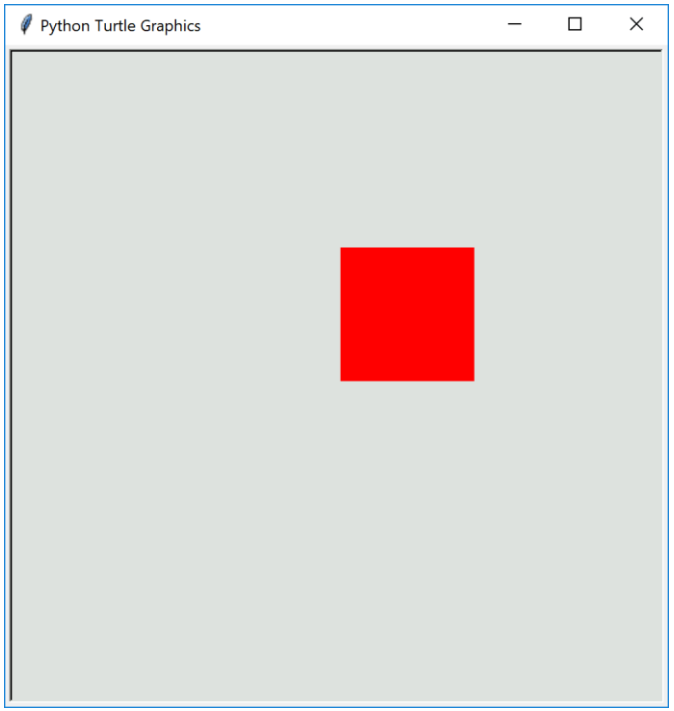
Color can be specified as a color string, or as a mix of (**Red**, **Green**, **Blue**)

```
pen = turtle.Turtle()
pen.hideturtle()
```

No more animals –
just a pen (and a brush)

Red square: version 1

```
# drawing red square
pen.color("red")
pen.fillcolor("red")

pen.begin_fill()
# start drawing
pen.forward(100)
pen.left(90)

pen.forward(100)
pen.left(90)

pen.forward(100)
pen.left(90)

pen.forward(100)
pen.left(90)
# end drawing
pen.end_fill()
```



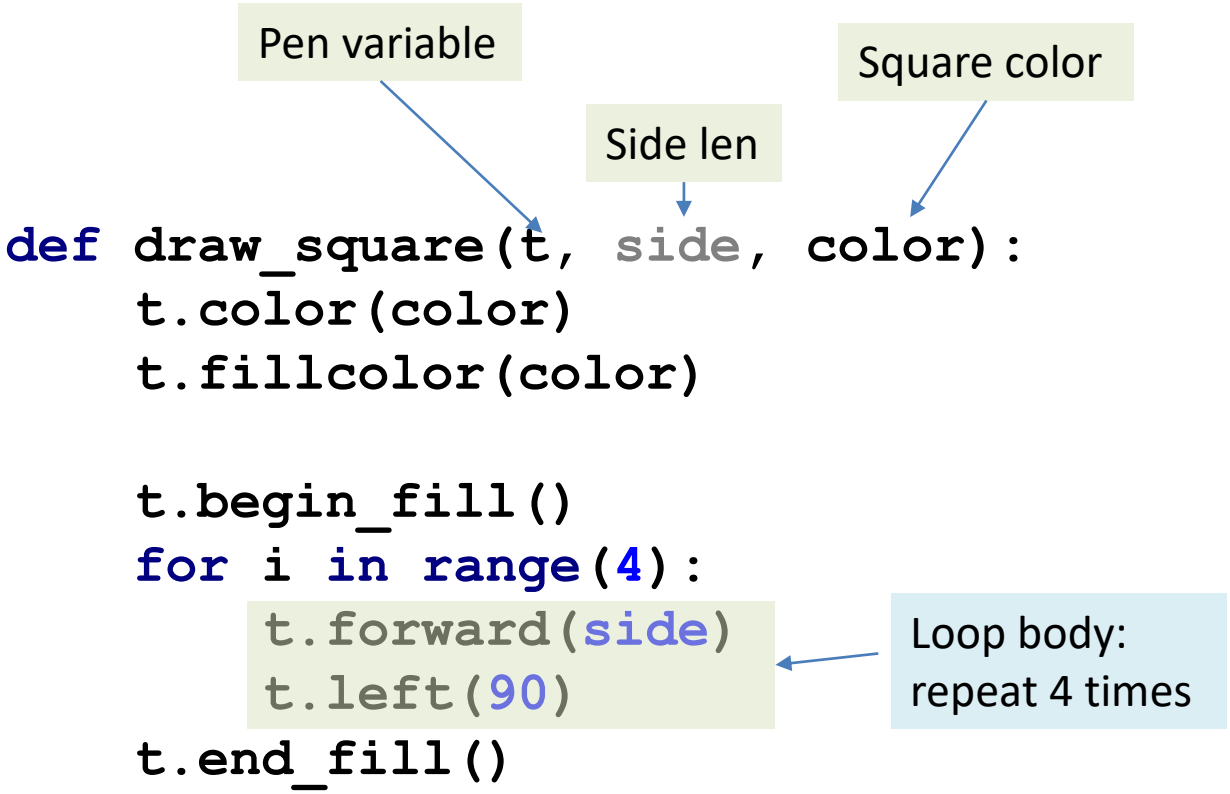
Identifying repeating patterns

```
# drawing red square
pen.color("red")
pen.fillcolor("red")

pen.begin_fill()
# start drawing
pen.forward(100)
pen.left(90)

pen.forward(100)
pen.left(90)

pen.forward(100)
pen.left(90)

pen.forward(100)
pen.left(90)
# end drawing
pen.end_fill()
```

What commands do we want to repeat?

How many times do we want to repeat?

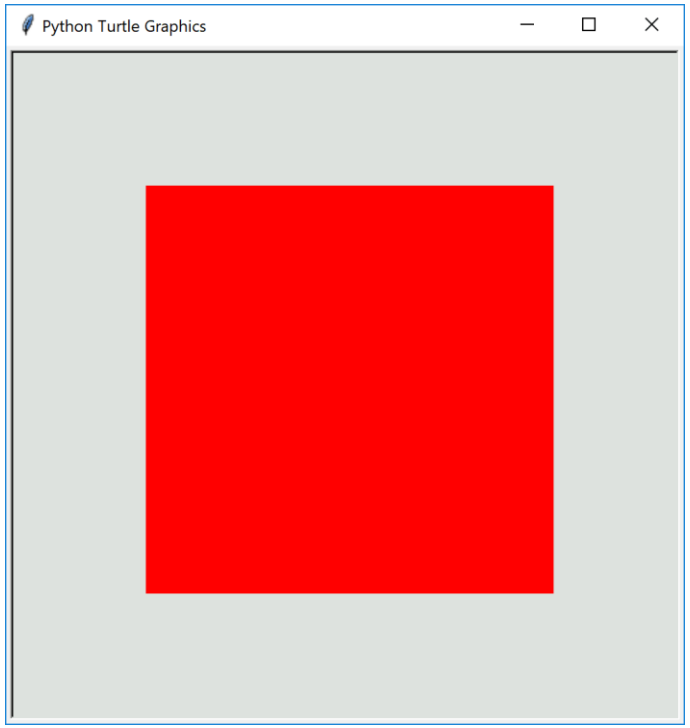
Red square: version 2:
with function and loop



Red square: version 2:

center the square

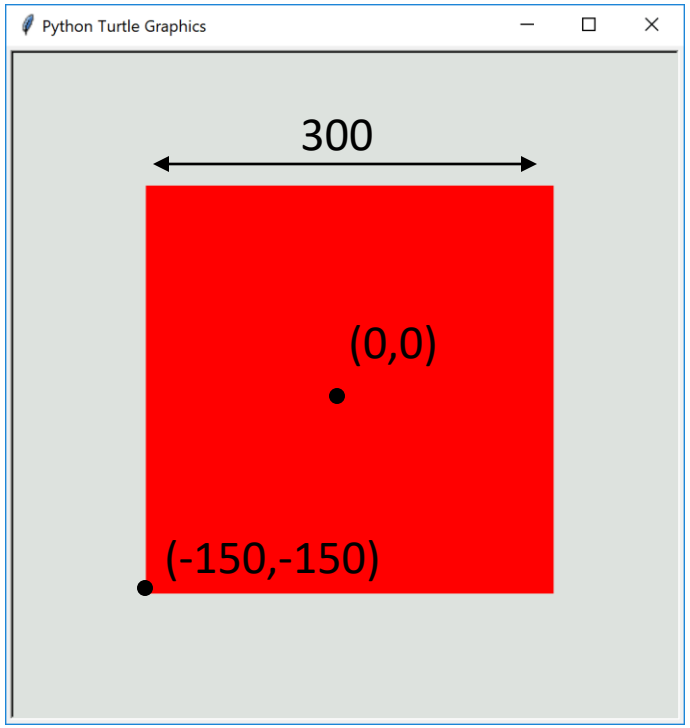
```
def draw_square(t, side, color):
    t.goto(-side/2, - side/2)
    t.color(color)
    t.fillcolor(color)

    t.begin_fill()
    for i in range(4):
        t.forward(side)
        t.left(90)
    t.end_fill()
```



Red square: version 2:

center the square

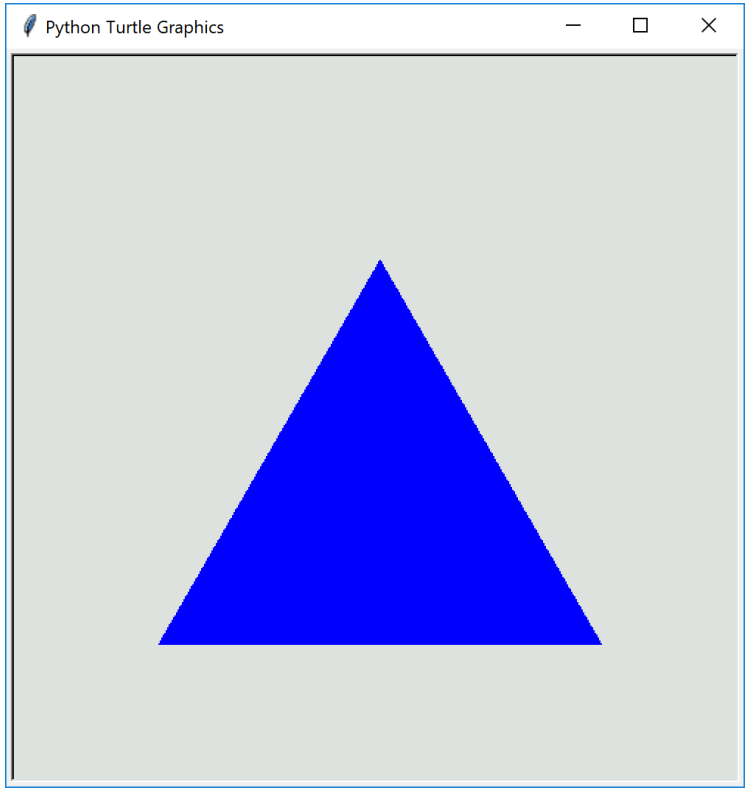
```
def draw_square(t, side, color):
    t.goto(-side/2, - side/2)
    t.color(color)
    t.fillcolor(color)

    t.begin_fill()
    for i in range(4):
        t.forward(side)
        t.left(90)
    t.end_fill()
```



Completed version in red_square.py

Draw triangle

```
def draw_triangle(t, side, color):
    t.goto(-side/2, - side/2)
    t.color(color)
    t.fillcolor(color)

    t.begin_fill()
    for i in range(3):
        t.forward(side)
        t.left(120)
    t.end_fill()



draw_triangle(pen, 300, "blue")
```



What is the difference between rectangle and triangle?

Could we create any regular n-gon?

```
def draw_triangle(t, side, color):
    t.goto(-side/2, - side/2)
    t.color(color)
    t.fillcolor(color)

    t.begin_fill()
    for i in range(3):
        t.forward(side)
        t.left(120)
    t.end_fill()
```

What should we change to make it generic?

Generic n-gon

```
def draw_ngon(t, n, side, color):
    """
    Draws an arbitrary n-sided polygon
    Parameters:
        t: turtle pen
        n: number of sides of the polygon
        side: length of each side
        color: fill color
    """
    t.color(color)
    t.fillcolor(color)

    t.begin_fill()
    for i in range(n):
        t.forward(side)
        t.left(360/n)
    t.end_fill()
```

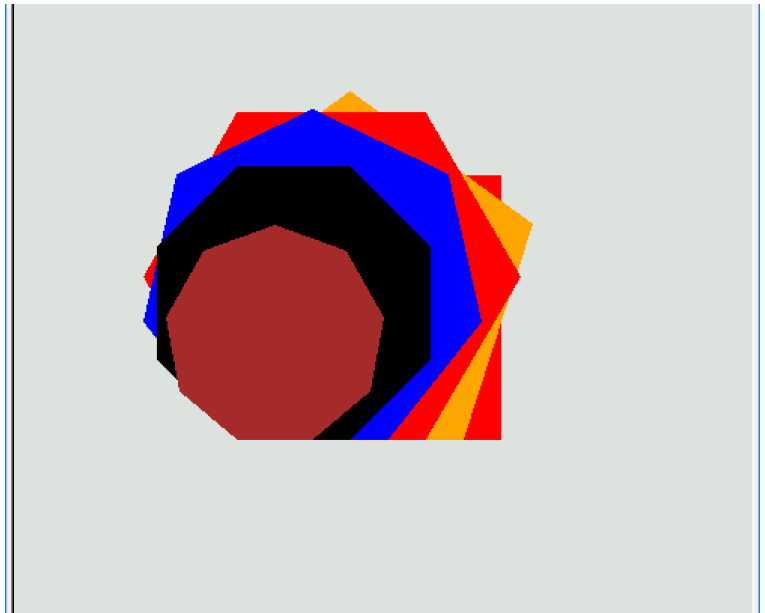
How many times to repeat?

How many degrees should we turn?

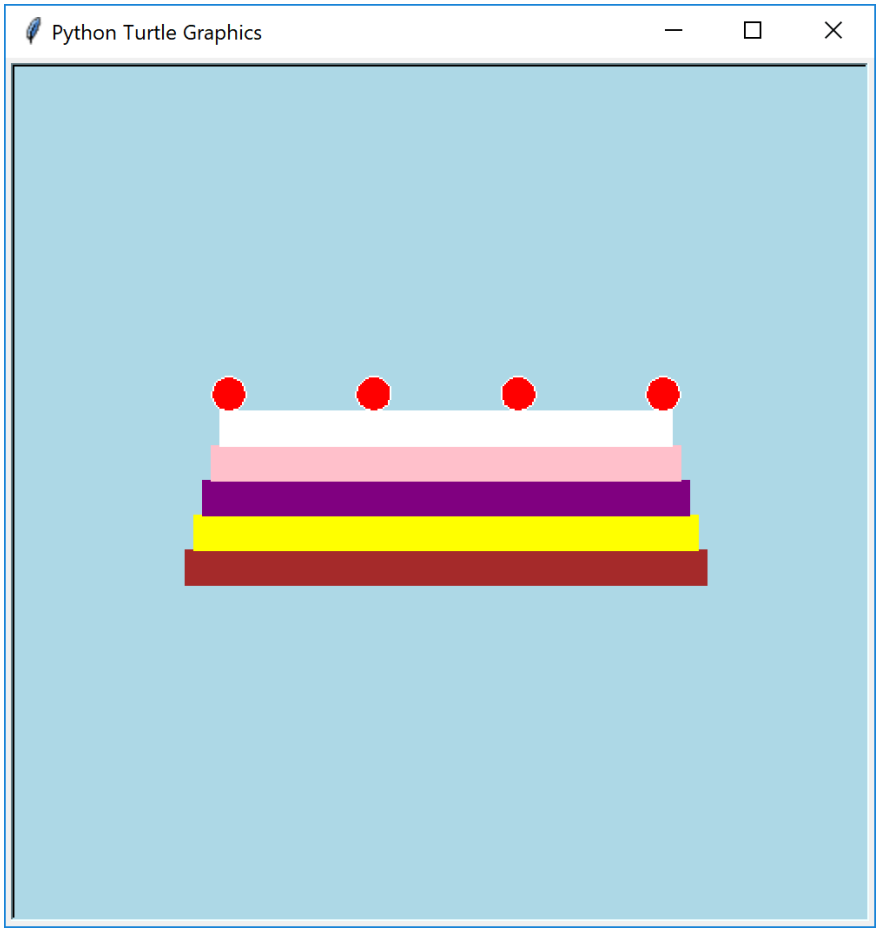
Completed version in [polygon.py]

N-gon art

```
# stacking polygons
colors = ["red", "blue", "black", "orange",
          "purple", "yellow", "green", "brown"]
pen.goto(-100, - 100)
curr_side_len = 200
for n in range (4, 10):
    curr_side_len -= 25
    draw_ngon(pen, n, curr_side_len,
              random.choice(colors))
```

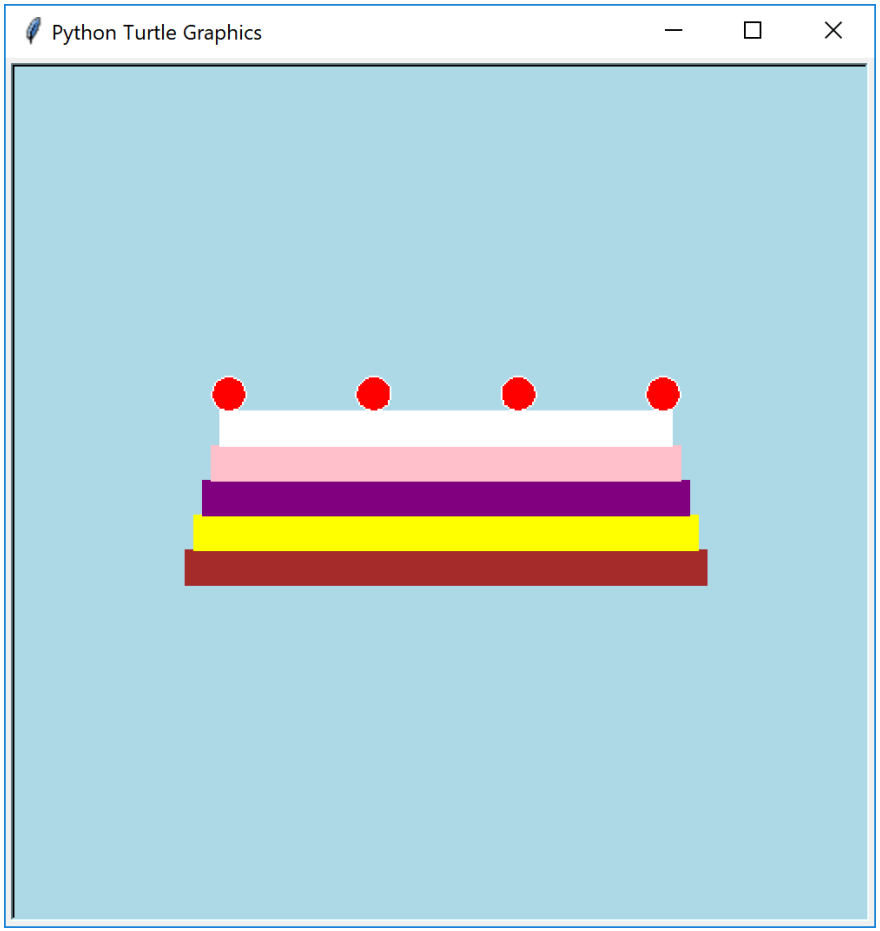


Birthday cake



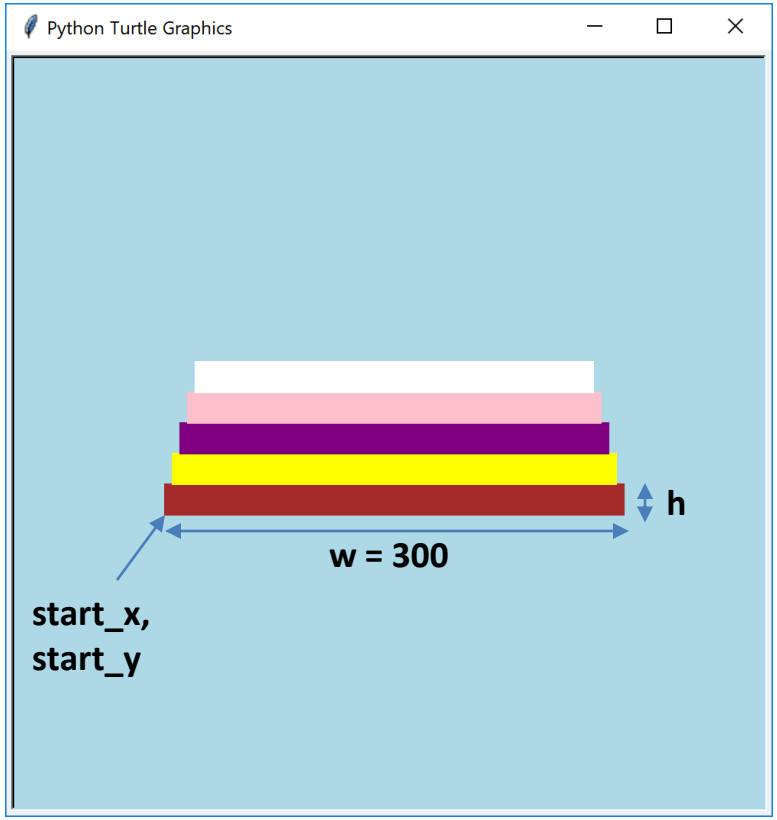
Completed version in [birthday_cake.py]

Loop pattern design recipe



1. Draw the desired shape on paper
2. Mark distances and dimensions
3. Identify repeating patterns: this goes into the body of the loop
4. Identify what changes from one repetition to another: make this change using accumulator variable (defined and initialized outside the loop)
5. Identify number of repetitions: loop header

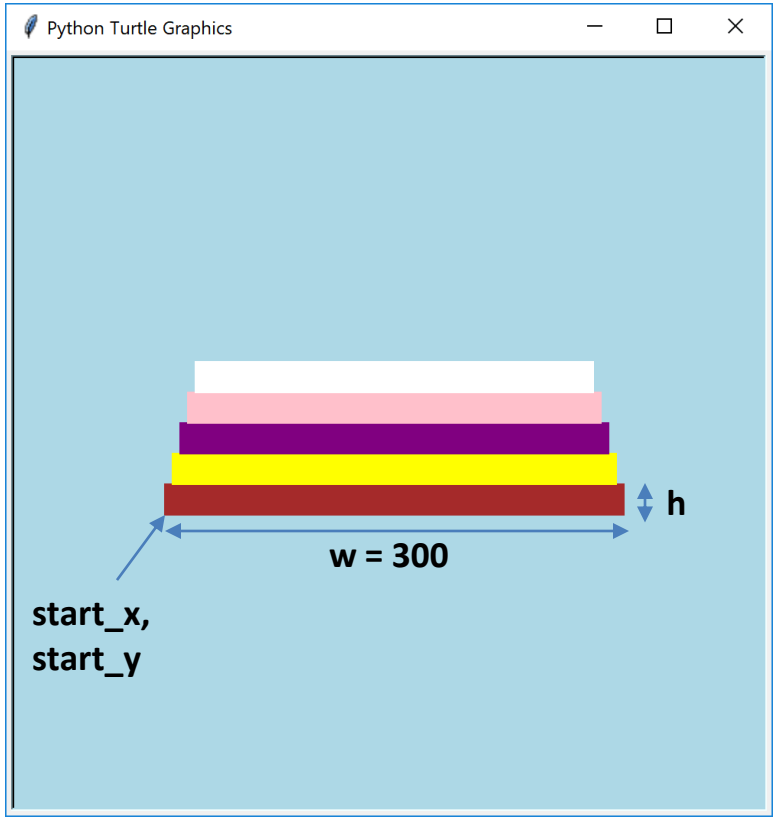
As it applies to birthday cakes …



Draw picture and mark coordinates and dimensions

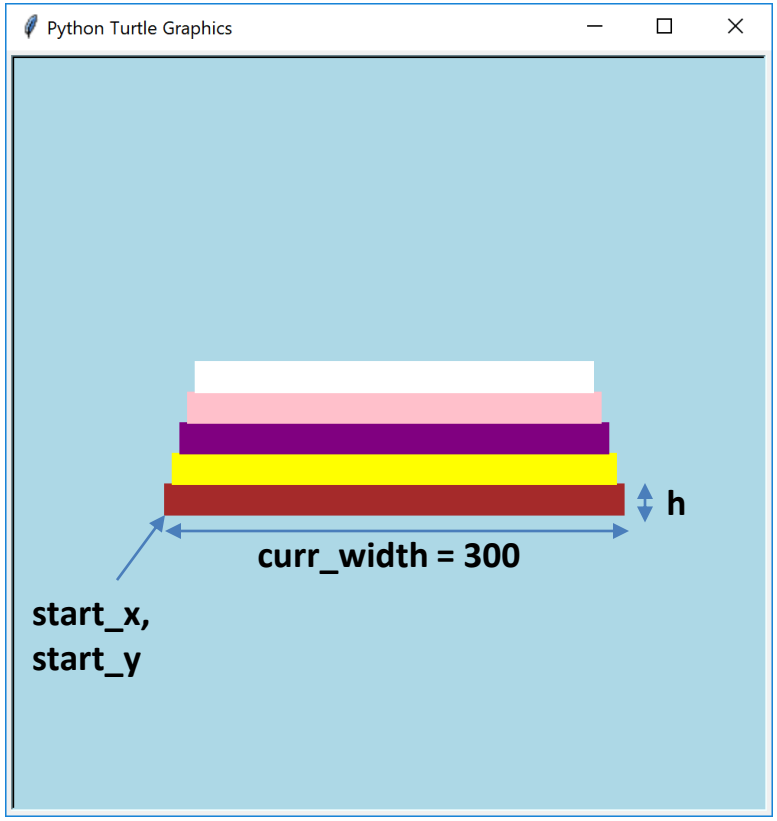
```
colors = ["brown", "yellow",
          "purple", "pink", "white"]
w = 300
h = 20
start_x = -w/2
start_y = -len(colors)*h/2
```

Code rectangle with current w,h



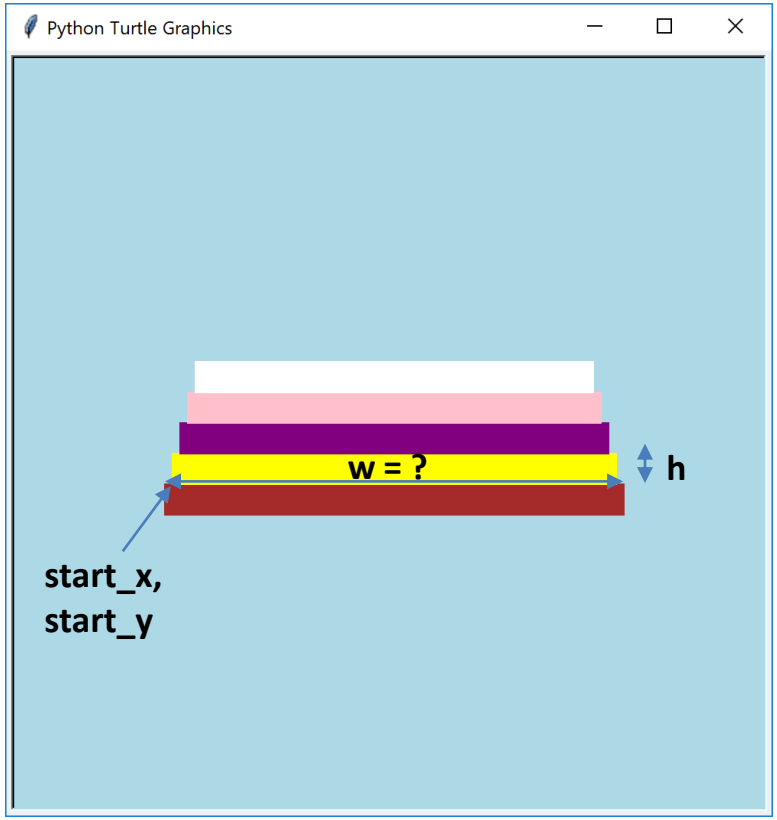

```
pen.goto(start_x, start_y
pen.color(colors[i])
pen.fillcolor(colors[i])

pen.begin_fill()
pen.forward(w)
pen.left(90)
pen.forward(h)
pen.left(90)
pen.forward(w)
pen.left(90)
pen.forward(h)
pen.left(90)
pen.end_fill()
```

There are total 5 rectangles to draw




```
for i in range(len(colors)):
    pen.goto(start_x, start_y)
    pen.color(colors[i])
    pen.fillcolor(colors[i])

    pen.begin_fill()
    pen.forward(w)
    pen.left(90)
    pen.forward(h)
    pen.left(90)
    pen.forward(w)
    pen.left(90)
    pen.forward(h)
    pen.left(90)
    pen.end_fill()
```

What changes at each iteration?



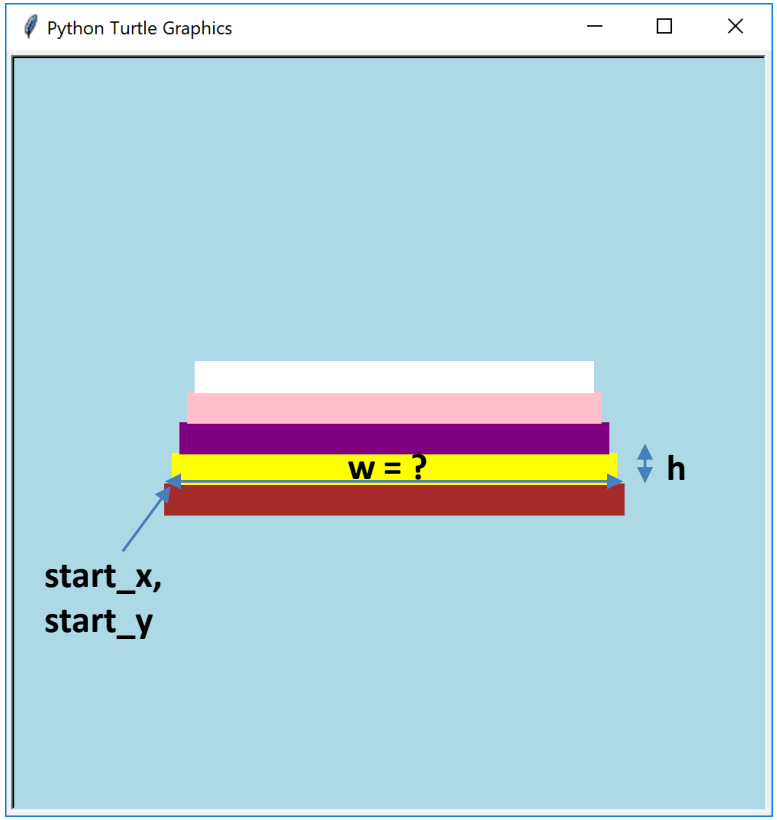

```
for i in range(len(colors)):
    pen.goto(start_x, start_y)
    pen.color(colors[i])
    pen.fillcolor(colors[i])

    pen.begin_fill()
    pen.forward(w)
    pen.left(90)
    pen.forward(h)
    pen.left(90)
    pen.forward(w)
    pen.left(90)
    pen.forward(h)
    pen.left(90)
    pen.end_fill()
```

How *start_x* changes?

How *start_y* changes?

How w changes?

What changes at each iteration?

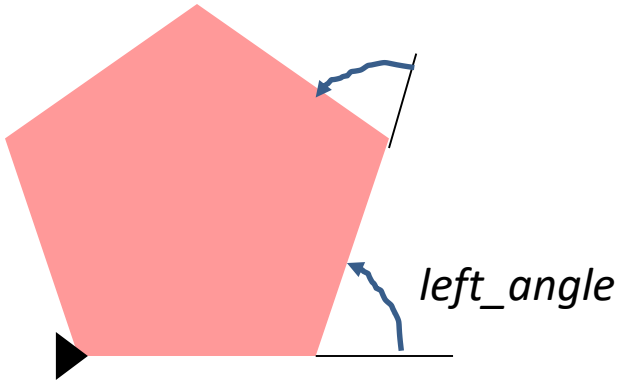



```
for i in range(len(colors)):
    pen.goto(start_x, start_y)
    pen.color(colors[i])
    pen.fillcolor(colors[i])

    pen.begin_fill()
    pen.forward(w)
    pen.left(90)
    pen.forward(h)
    pen.left(90)
    pen.forward(w)
    pen.left(90)
    pen.forward(h)
    pen.left(90)
    pen.end_fill()

    start_x += 5
    start_y += h

    w -= 10
```

Fast turtle!

- You can adjust the speed of the turtle with **`speed()`** or with **`tracer()`**
- **`tracer(n)`**
  - Sets drawing to update every "regular" n^{th} screen update
    - Use larger values for faster updates
- **`tracer(1)`**
  - Default – Slowest update
  - To speed up drawing, set to a higher value
- **`tracer(0)`**
  - Disables screen updates.
  - After you draw, call the **`update()`** function to force drawing to appear on screen

Designing n-star algorithm



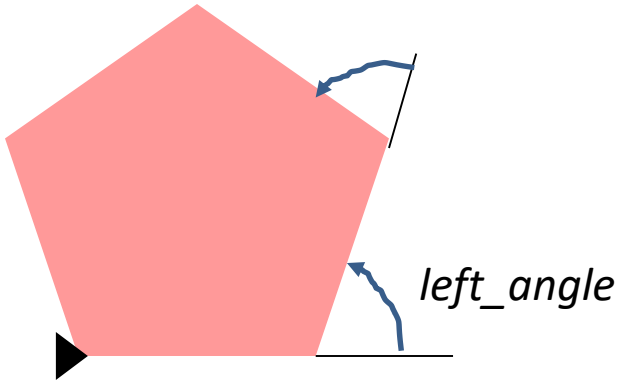
We want to design an algorithm for drawing a generic n-star (n>=5)

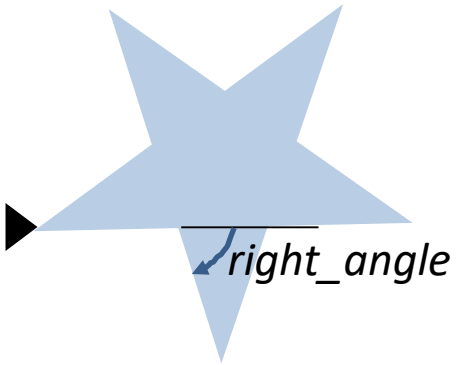
The logic starts with a **polygon**

- How many times did turtle turn to draw a pentagon (5-gon)?
- It ends facing the same direction – so how many degrees did it turn left in total?

- What is the angle at each turn?

Designing n-star algorithm

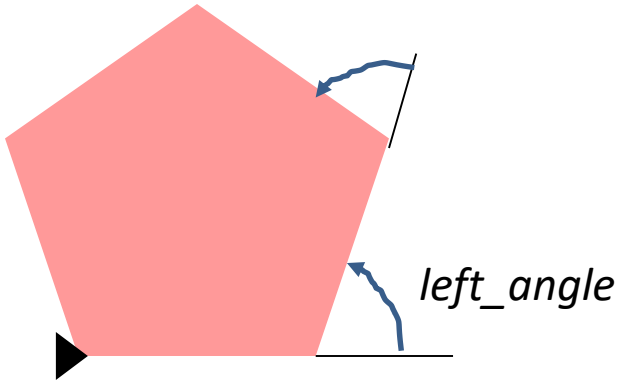




To draw a full star with n rays:

- How many times did turtle turn to the **right**?

Designing n-star algorithm



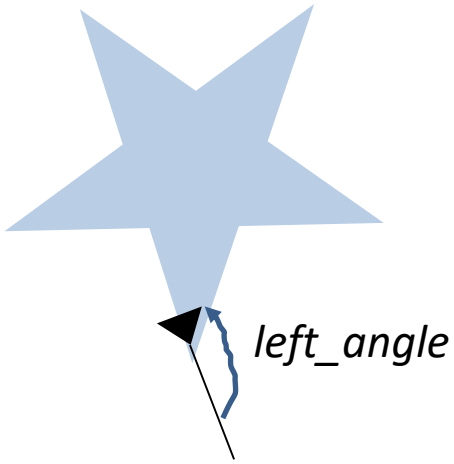
To draw a full star with n rays:

- How many times did turtle turn to the **right**?

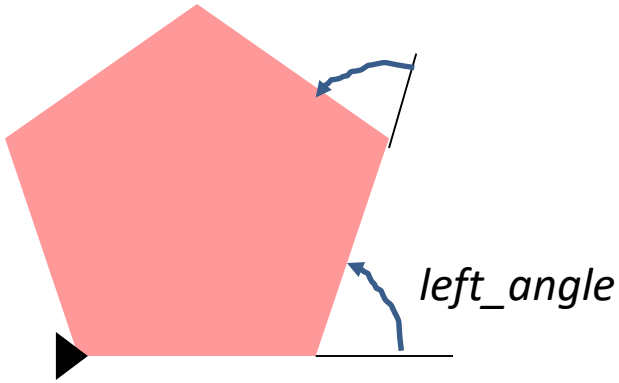
n times

- How many times did it turn to the **left**?

n times



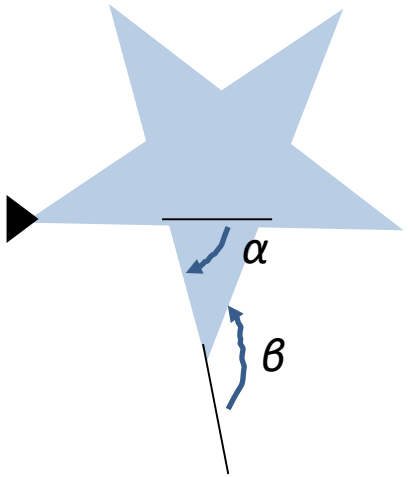
Designing n-star algorithm



To draw a full star with ***n*** rays:

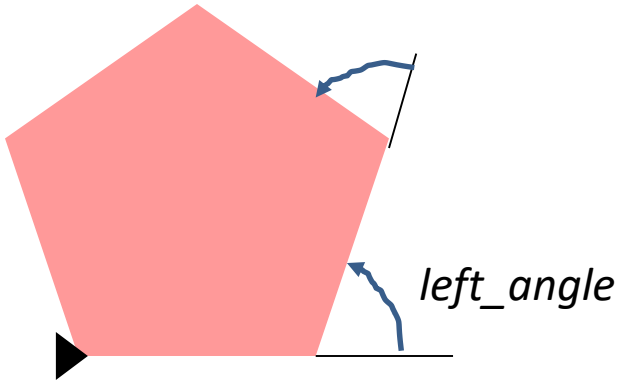
- Turn turtle ***n*** times to the **right**
- Turn turtle ***n*** times to the **left**

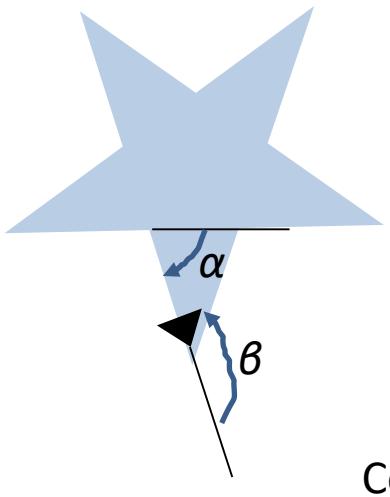
- The turtle ends facing the original direction -- making a complete left turn by 360 degrees



$\beta = 2\,\alpha$

Designing n-star algorithm





To draw a full star with n rays:

- The turtle made a complete left turn by 360 degrees
- But each *left_angle = 2*right_angle*!
- Therefore:

*n * β* - n * *α* = *360*

- *Thus:*

right_angle = 360/n
*left_angle = 2*right_angle*

Completed version in [star_turtle_algorithm.py]

Inspirations

- Interesting drawings can be obtained by running the same drawing routine multiple times
  - Circles forming a flower: flower.py
  - Circles forming a circular ornament: circles.py
  - Lines forming a square ornament: squares.py

- Nested loops are also useful here:
  - Multiple stars forming a circular ornament: nested_star_loop.py