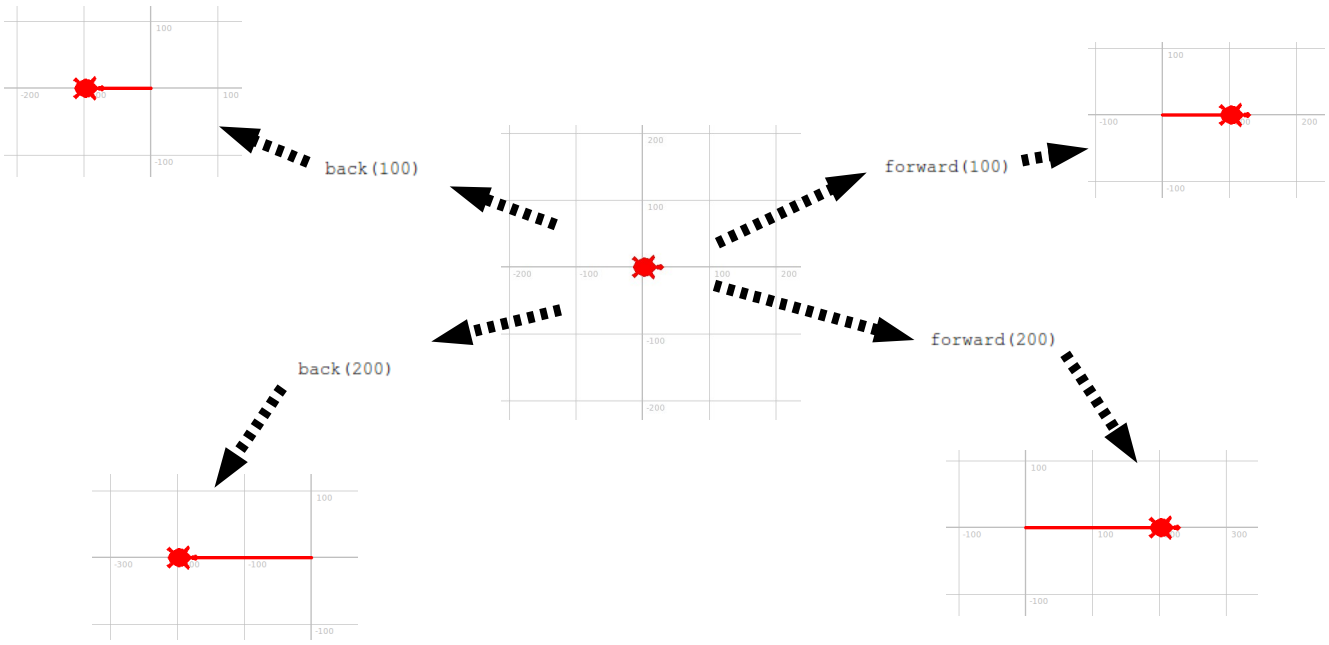
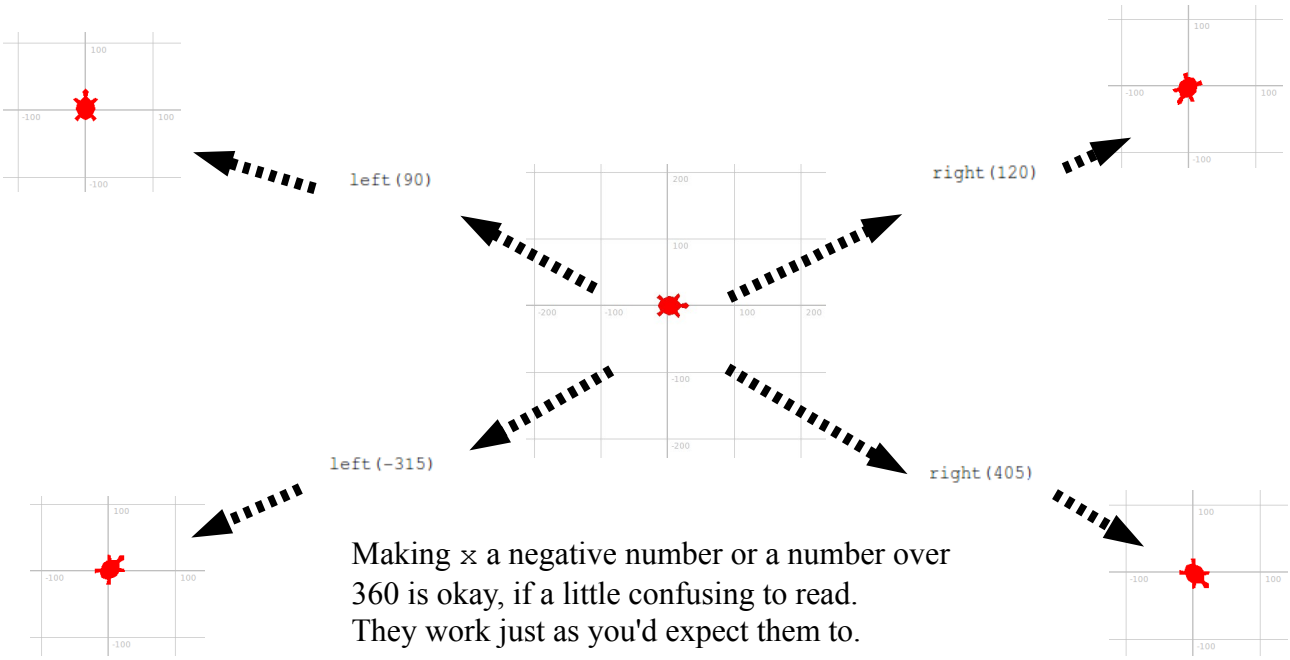
Movement

The `forward()`, `back()`, `left()`, and `right()` commands make the Turtle move around in the world relative to its current location and heading.

`forward(x)` and `back(x)` move the turtle `x` distance. With the pen down (as it starts out), the turtle will draw a line as it moves.

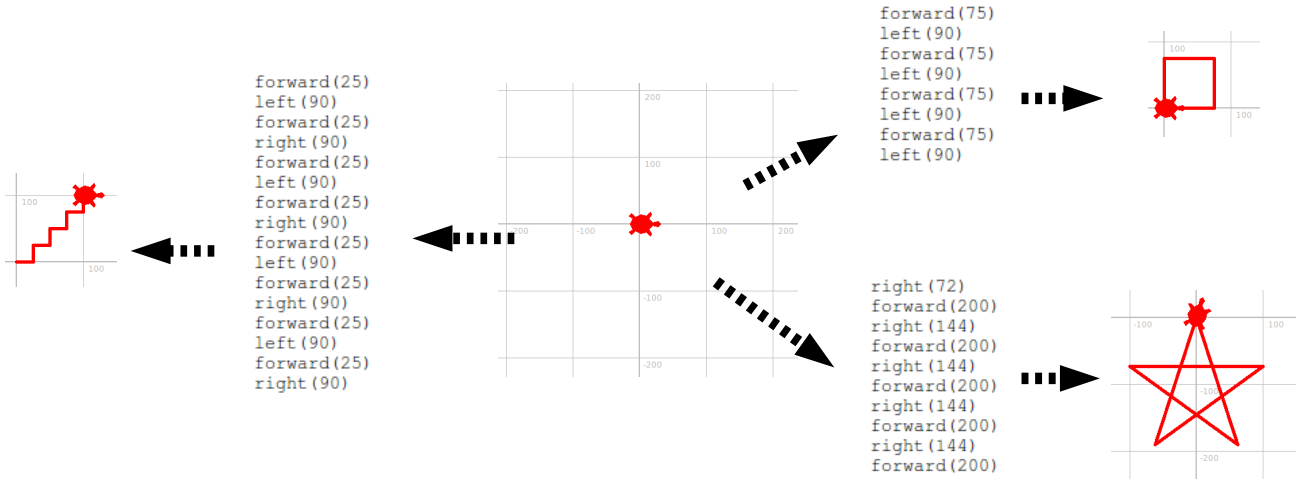


`left(x)` and `right(x)` turn the turtle `x` degrees in the corresponding direction so it faces a new heading. Since it's not moving, this does not leave a mark.



Making `x` a negative number or a number over 360 is okay, if a little confusing to read.
They work just as you'd expect them to.

By combining these commands, we can have the Turtle draw all kinds of shapes.



Repetition

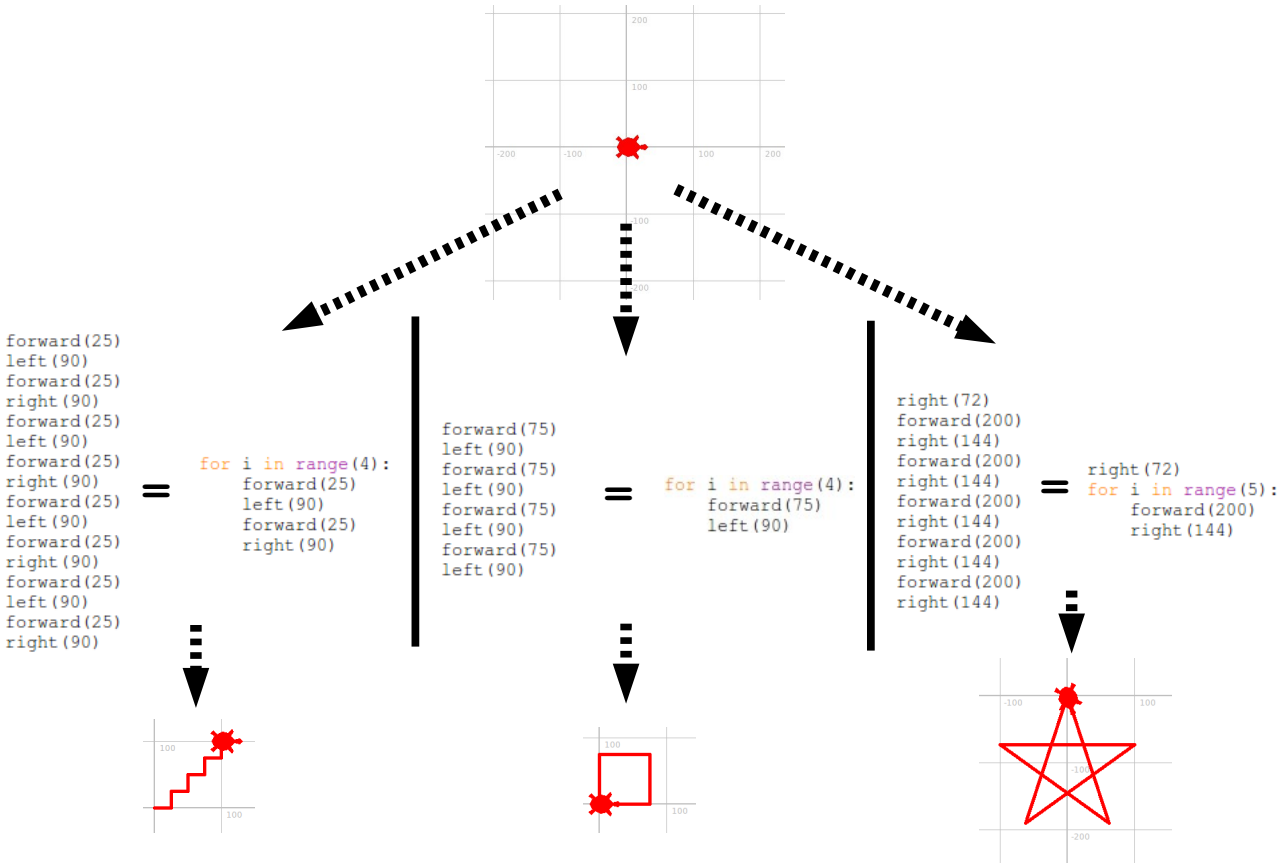
One of the things computers are really great at is doing things over and over again.

There's an easy way to tell the computer to do a small sequence of commands a certain number of times: a “for loop”.

This is accessible from Turtlecon.pyw's “Commands” menu as “repeat...”

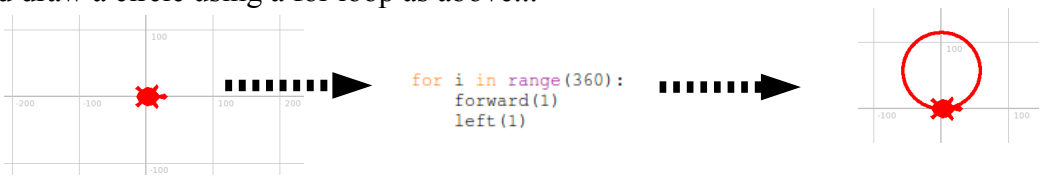
```
for i in range(<how many times?>):
    # commands go here
```

Replace the <how_many_times?> part with the number of times you want to repeat the commands. Then place all the commands to be repeated indented underneath.



Circles

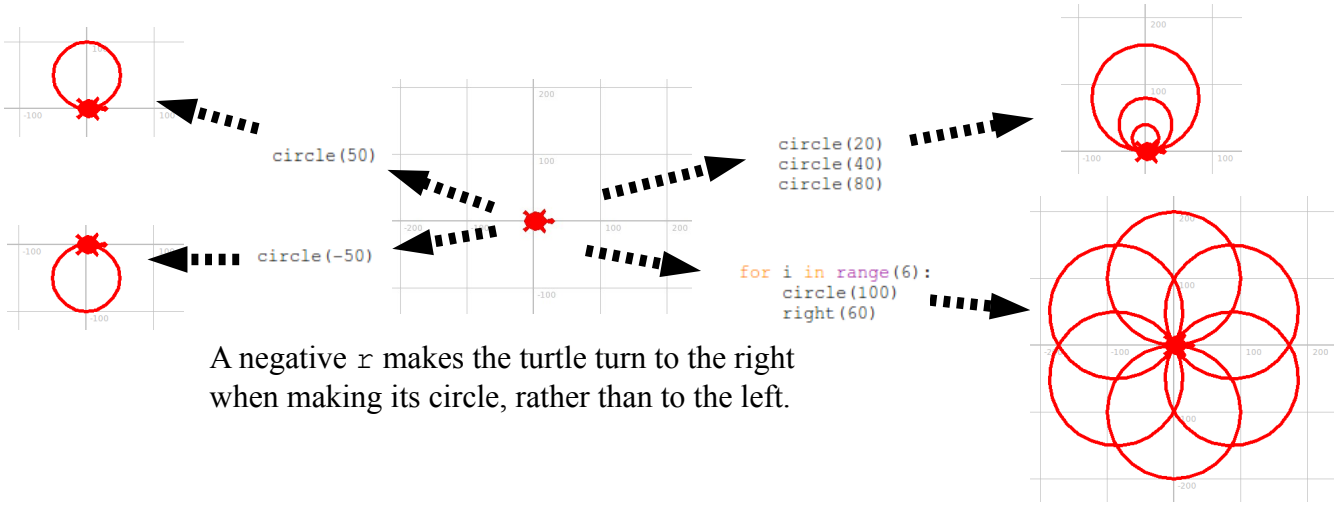
You could draw a circle using a for loop as above...



But that's slow, complex, and it's hard to figure out what size it will be. Anyhow, there's an easier way!

The `circle()` command makes the turtle move in a circular motion relative to its current location and heading. If the pen is down, it will draw a line as it moves.

`circle(r)` moves the turtle forward and to the left in a circle with a radius of `r`.

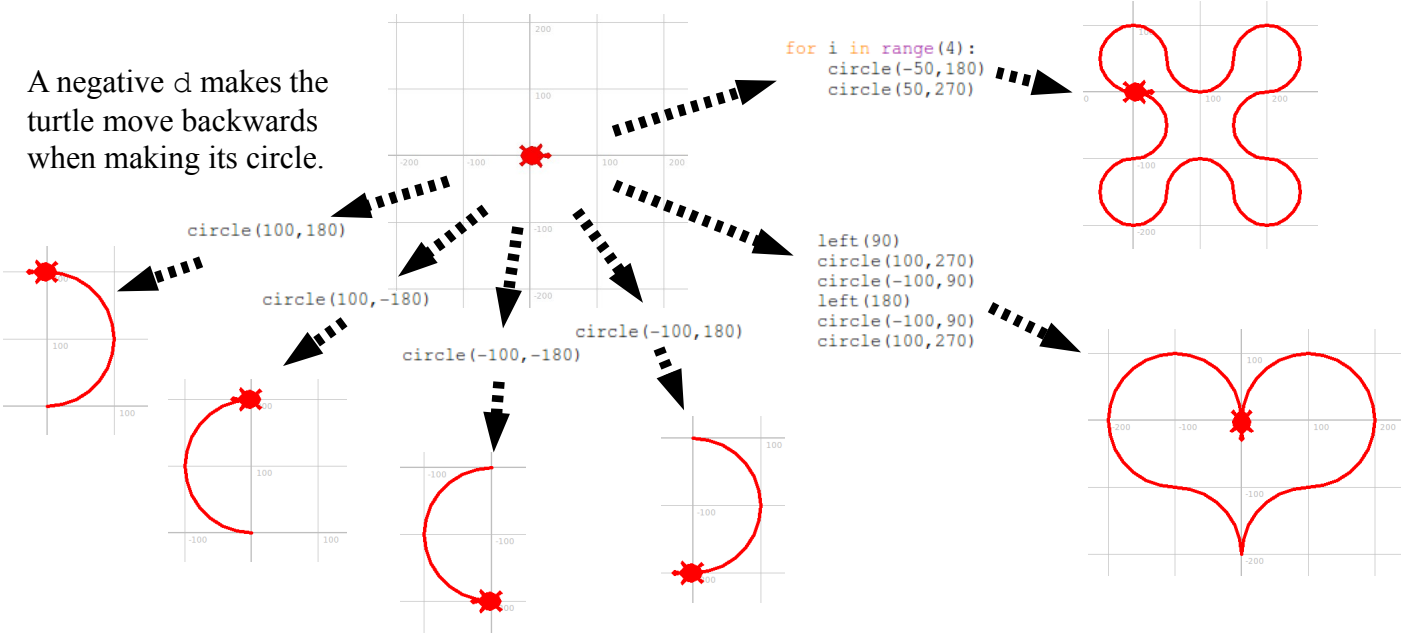


A negative `r` makes the turtle turn to the right when making its circle, rather than to the left.

Partial Circles

`circle(r,d)` moves the turtle forward and to the left `d` degrees of a circle with a radius of `r`.

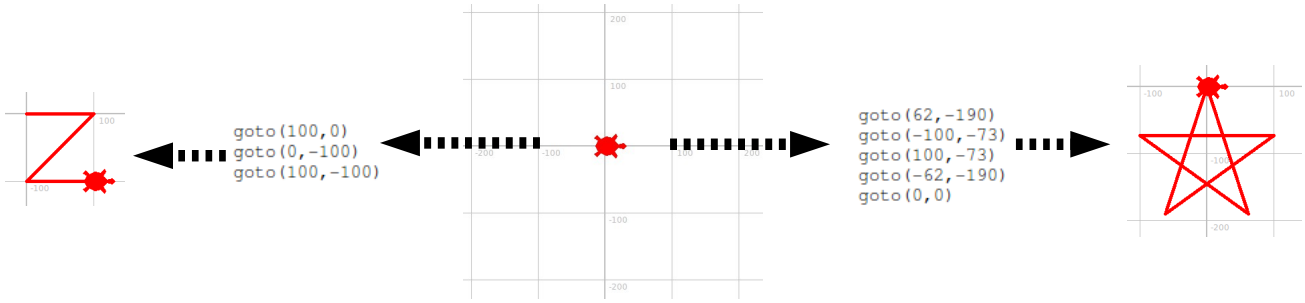
A negative `d` makes the turtle move backwards when making its circle.



Going to a specific position

The `goto()` command makes the turtle "go to" a specific position relative to the world and not its current location. The turtle does not turn as it moves so the heading remains the same, and if the pen is down it will draw a line.

`goto(x,y)` moves the turtle to coordinate position `(x,y)`.

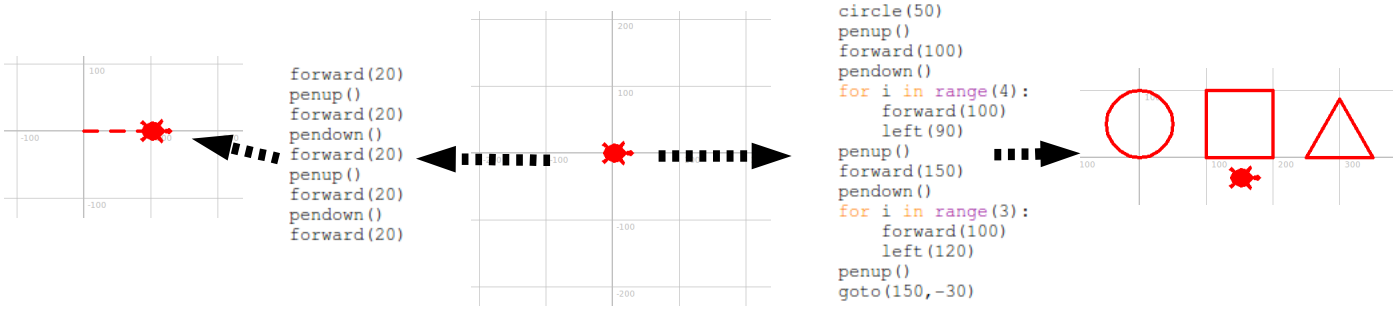


What if we want to make more than one long, continuous line?

The `penup()` and `pendown()` commands let us tell the turtle to stop and start drawing its line.

`penup()` raises the turtle's pen so it stops drawing.

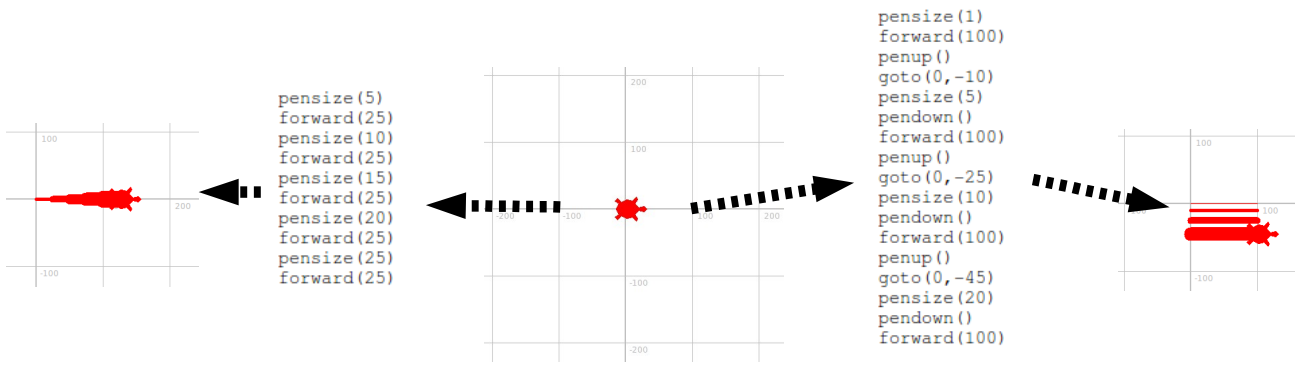
`pendown()` lowers the turtle's pen so it starts again.



Pen size – lines of different thicknesses

The `pensize()` command lets us change the thickness of the pen, and thus of the lines it draws.

`pensize(x)` sets the thickness of the pen to `x` diameter.

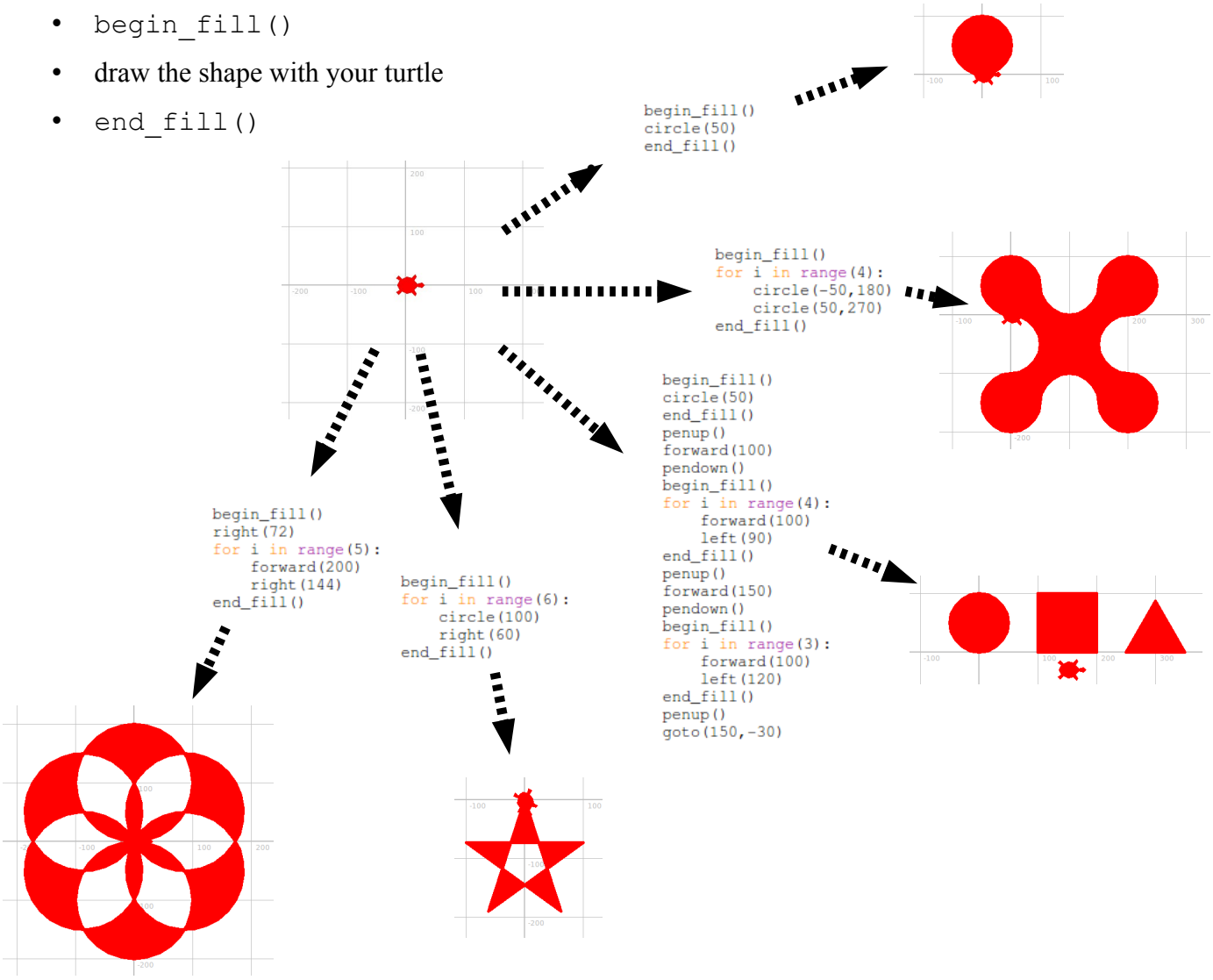


Filling in Shapes

The `begin_fill()` and `end_fill()` commands let us define shapes on the screen to be filled in with solid color (until we change it, that solid color is red).

They're used in combination with movement commands, like this:

- `begin_fill()`
- draw the shape with your turtle
- `end_fill()`



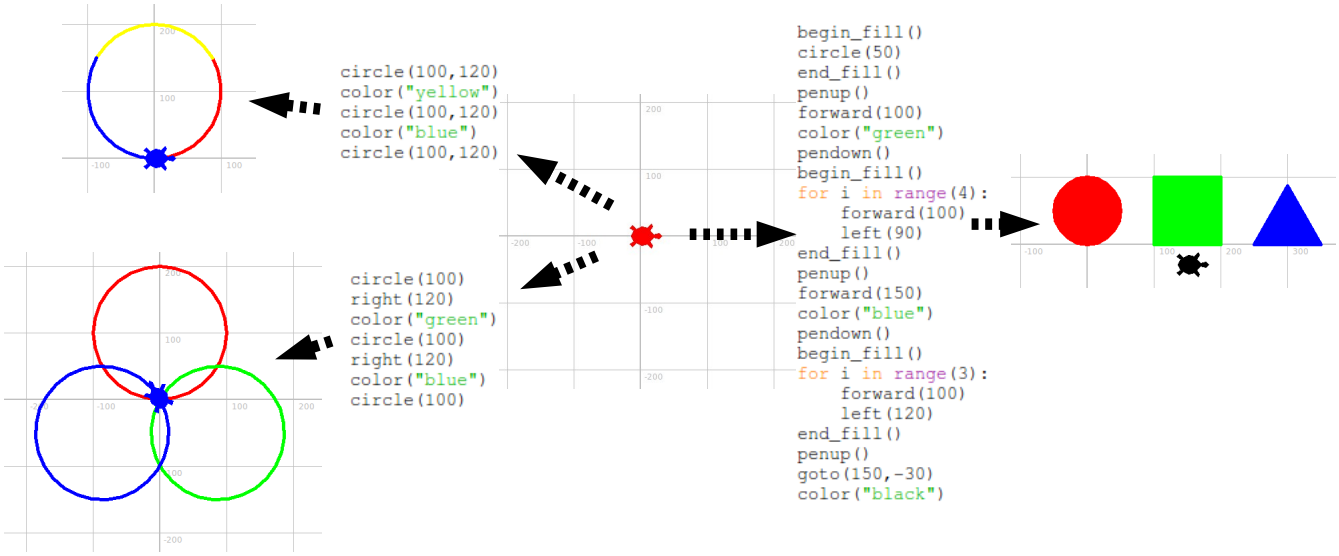
Overlapping lines while drawing your shape can have strange and interesting results.

Color

The `color()` command changes the color of the turtle; this affects the color of the line it draws with the pen and the color it fills in shapes with.

In our world, the turtles start out the color `"red"`. By adding a `color()` command before we move, however, we can make our lines be drawn in other colors.

`color(c)` changes the turtle's color to `c`.



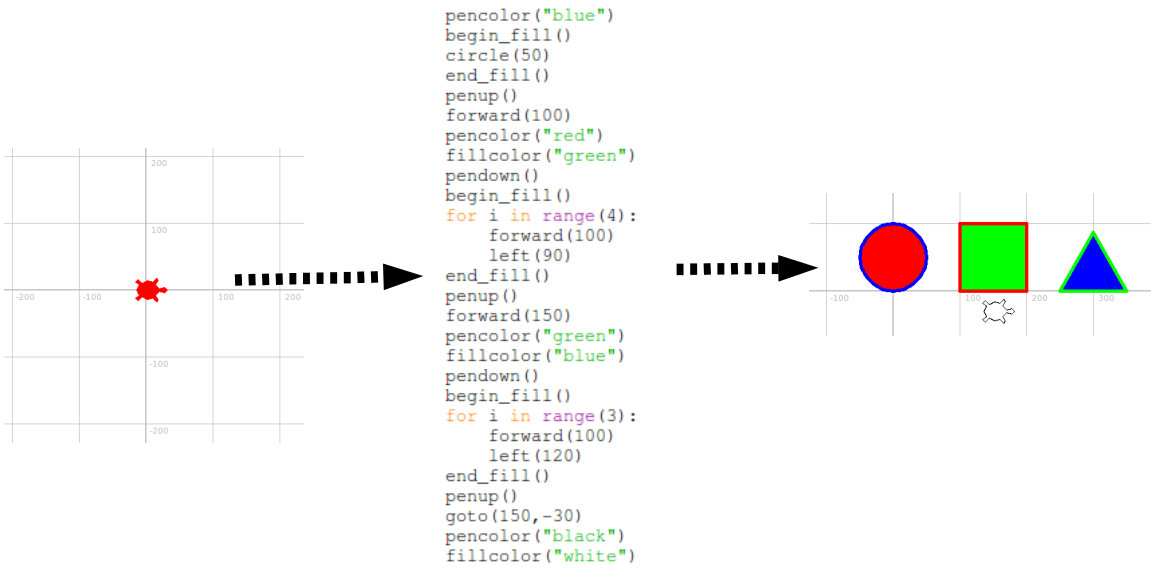
Pen color/Fill color

There are in actuality two colors our turtle keeps track of, and the `color()` command changes both.

We can use `pencolor()` and `fillcolor()` to change each independently, though.

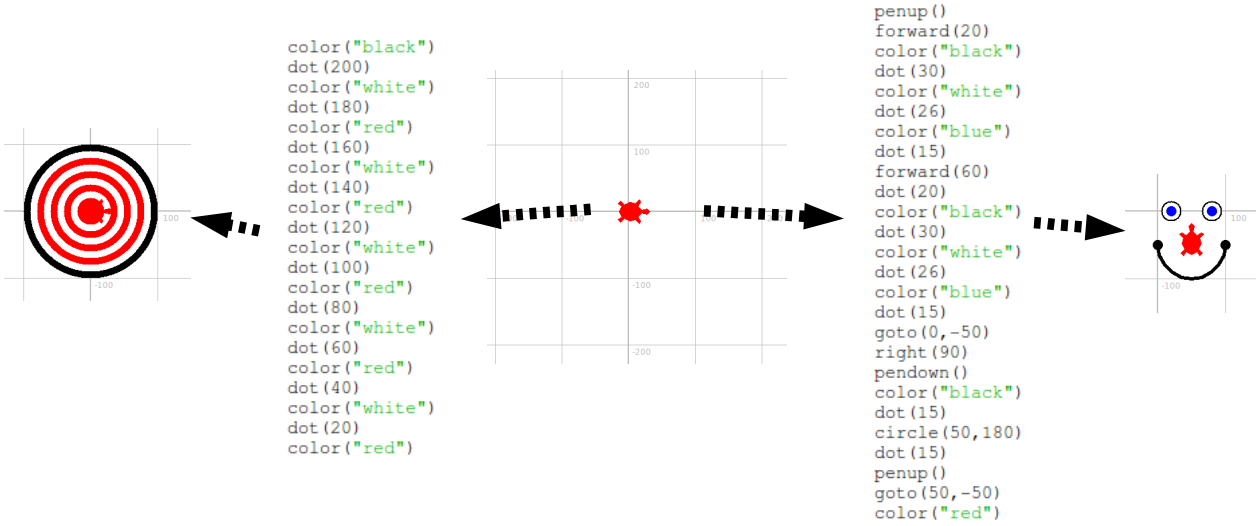
`pencolor(c)` changes the color drawn by the turtle's pen when moving to `c`.

`fillcolor(c)` changes the color filled in by the turtle when using `begin_fill()` and `end_fill()` to `c`.



Dots

`dot(x)` makes a dot of `x` radius centered on your turtle.



```
color("black")
dot(200)
color("white")
dot(180)
color("red")
dot(160)
color("white")
dot(140)
color("red")
dot(120)
color("white")
dot(100)
color("red")
dot(80)
color("white")
dot(60)
color("red")
dot(40)
color("white")
dot(20)
color("red")
```

```
penup()
forward(20)
color("black")
dot(30)
color("white")
dot(26)
color("blue")
dot(15)
forward(60)
dot(20)
color("black")
dot(30)
color("white")
dot(26)
color("blue")
dot(15)
goto(0,-50)
right(90)
pendown()
color("black")
dot(15)
circle(50,180)
dot(15)
penup()
goto(50,-50)
color("red")
```

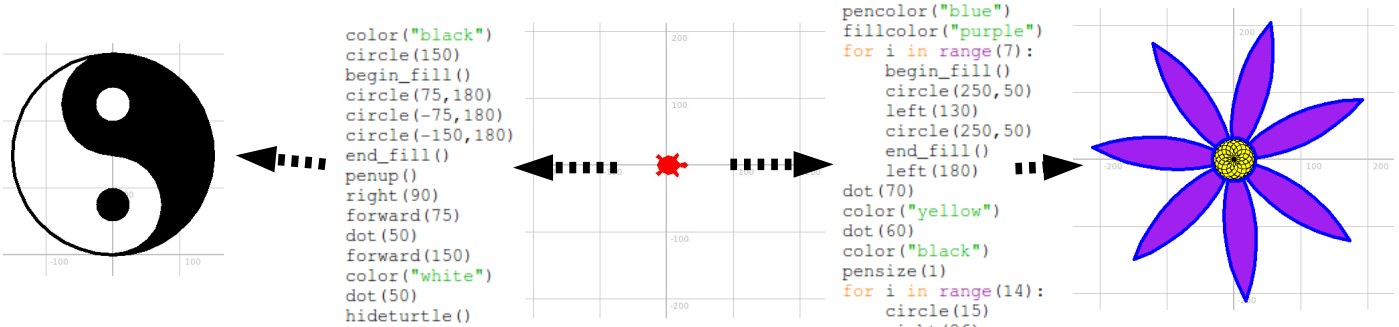
Various other commands...

`speed("fastest")` makes the turtle move as fast as it can (a time-saver when placed at the beginning of long programs).

`hideturtle()` makes the turtle invisible (e.g, if you don't want it around when you're done) and `showturtle()` makes it visible again.

Have fun!

The few commands above can be combined in endless ways to draw just about anything you can imagine, so experiment with them, have some fun, and see what you can come up with!



```
color("black")
circle(150)
begin_fill()
circle(75,180)
circle(-75,180)
circle(-150,180)
end_fill()
penup()
right(90)
forward(75)
dot(50)
forward(150)
color("white")
dot(50)
hideturtle()
```

```
pencolor("blue")
fillcolor("purple")
for i in range(7):
    begin_fill()
    circle(250,50)
    left(130)
    circle(250,50)
    end_fill()
    left(180)
dot(70)
color("yellow")
dot(60)
color("black")
pensize(1)
for i in range(14):
    circle(15)
    right(26)
hideturtle()
```