Project 1: Pretty Pictures

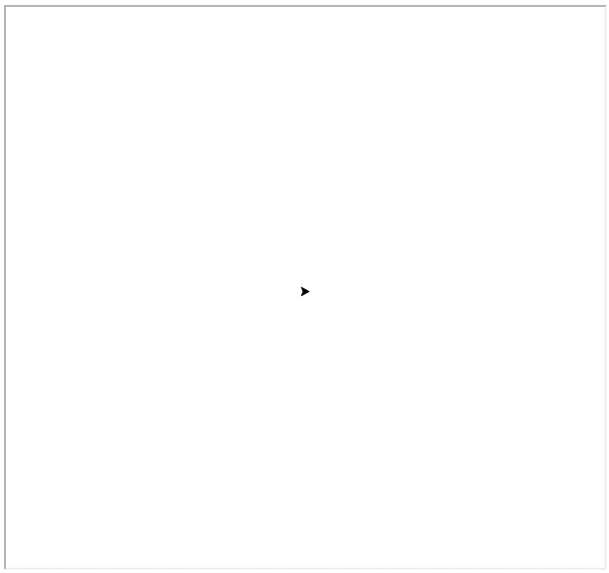
So far we have been concerned only with programs which read and write text. But we have been sitting in front of a computer with graphical elements on the screen as well as textual ones.

There are many ways to produce pictures, both line drawing and photographic, using programming languages like Python. For this project, we will use the `turtle` module which uses a model of drawing invented for children but fun for adults too. In this model, there is a little 'turtle' on screen, and we direct it where to go, and it leaves a trail behind it as it goes.

To begin, we import the `turtle` module, and create a new turtle, which we call `t`:

```
Python
>>> import turtle
>>> t = turtle.Turtle()
```

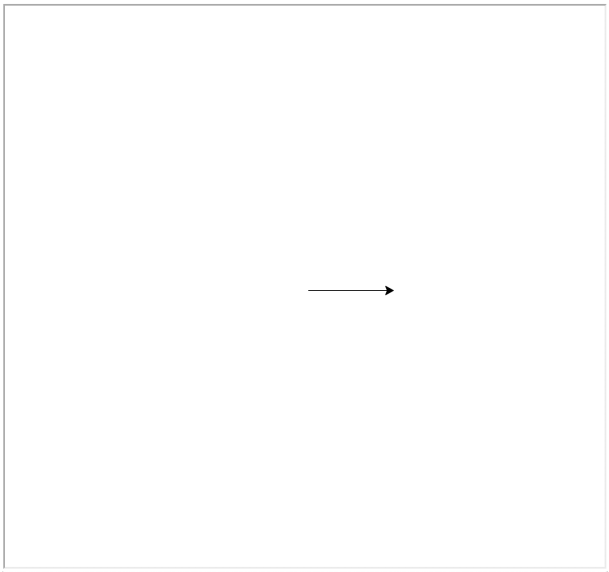
Upon typing the second line, a blank window appears, with the turtle represented by an arrow, pointing to the right:



We can now issue a command for the turtle to follow:

```
Python
>>> t.forward(100)
```

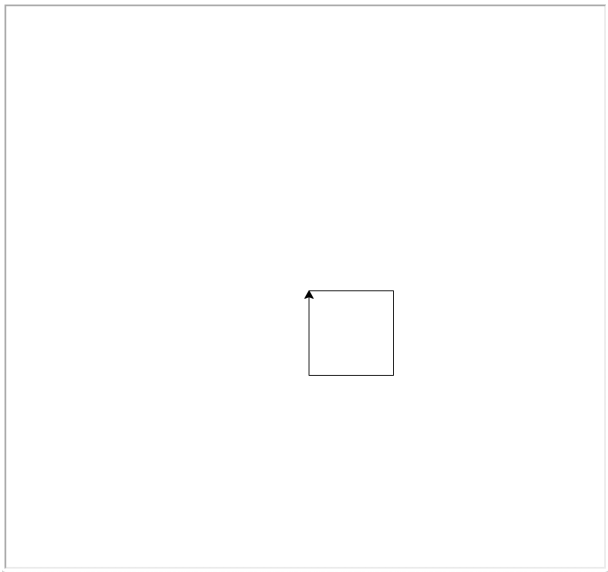
Here is the result:



We can complete the square by turning repeatedly by ninety degrees and moving forward.

```
Python
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
>>> t.right(90)
>>> t.forward(100)
```

The final result is a square of side 100, with the turtle in its original position, but pointing upwards:



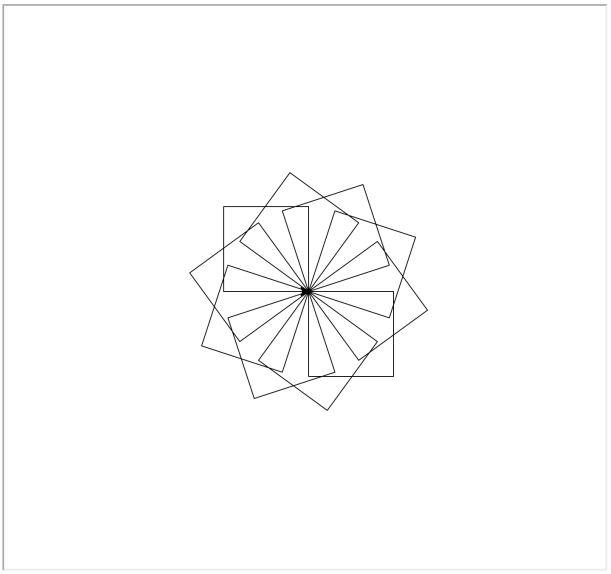
We can write a function to make a square of any size:

```
Python
>>> def square(x):
...     for _ in range(4):
...         t.fd(x)
...         t.rt(90)
```

The functions `fd` and `rt` are abbreviations for `forward` and `right`. The underscore _ is used to indicate that we are not using the counter from the **`for`** loop. We can make a primitive star by using `square` multiple times:

```
Python
>>> for _ in range(10):
...     square(100)
...     t.rt(360/10)
```

Here is the result:



When experimenting, the methods `home` and `clear` are useful:

```
Python
>>> t.home()
>>> t.clear()
```

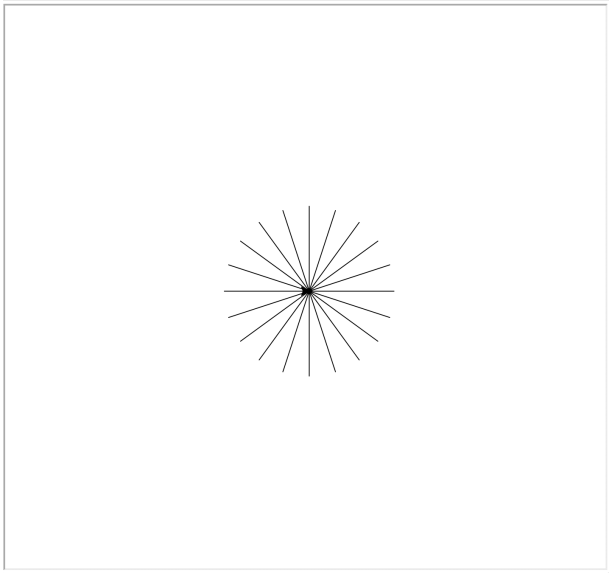
The `home` method moves the turtle to the origin and restores its direction to the default. The `clear` function clears the turtle screen.

QUESTION 1 Write a function `many_squares` which, given a number of squares to use and a size for the 'star', draws it.

To make another kind of star, we can use the `backward/bk` method:

```
def star(l, n):
    for _ in range(n):
        t.fd(l)
        t.bk(l)
        t.rt(360/n)
```

Here is `star(100, 20)`:



There is a `left/lt` equivalent to `right/rt` as well.

QUESTION 2 Write a function to draw a polygon with a given number of sides of a given length. Use this function, together with right turns, to repeat the given polygon multiple times to make a symmetrical pattern.

QUESTION 3 Write a function `circle`, which draws a circle for a given centre position and radius, choosing the number of sides dependent on size to give a smooth result. If you need π, it can be found as `math.pi` after using **`import`** `math`.

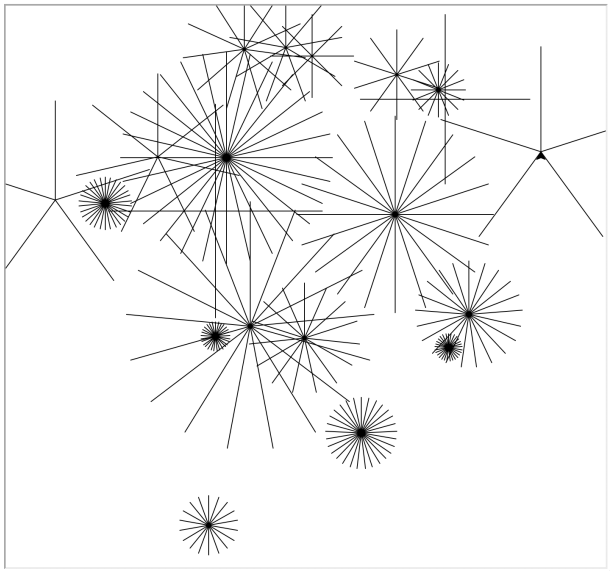
So far we have no way to prevent the turtle leaving a trail behind. What if we want to draw multiple stars? We can use the methods `penup` (stop drawing a trail) and `pendown` (resume drawing a trail):

```
def star(x, y, l, n):
    t.penup()
    t.home()
    t.fd(x)
    t.lt(90)
    t.fd(y)
    t.pendown()
    for _ in range(n):
        t.fd(l)
        t.bk(l)
        t.rt(360/n)
```

Now we can use the `random` module to draw lots of stars:

```
Python
>>>import random
>>>for _ in range(20):
...    star(random.randint(-300, 300),
...         random.randint(-300, 300),
...         random.randint(10, 150),
...         random.randint(3, 30))
```

Here is the result of one run:



We can simplify by using the method `goto` which moves to a given coordinate directly. We also use `setheading` to start each star at a random angle:

```
def star(x, y, l, n):
    t.penup()
    t.goto(x, y)
    t.setheading(random.randint(0,359))
    t.pendown()
    for _ in range(n):
        t.fd(l)
        t.bk(l)
        t.rt(360/n)
```

QUESTION 4 Using the `goto` method, write a function to draw a square grid of circles of diameter fifty which touch one another.

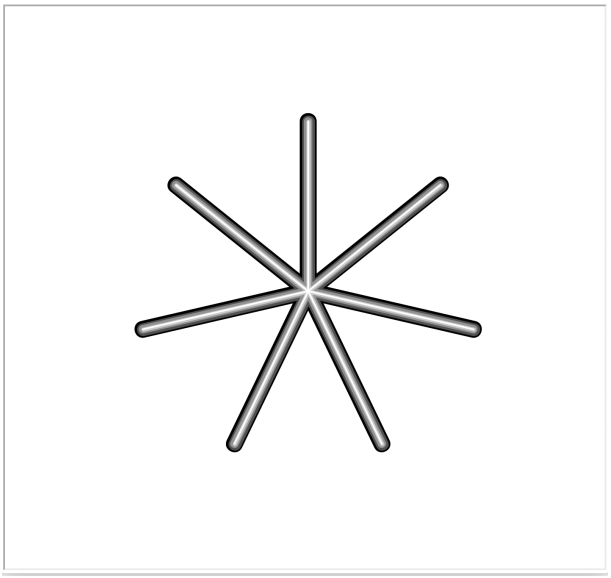
There are two problems with our pictures: they take a long time to draw, and the turtle gets in the way of the final result. To improve the speed, we use the `speed` method, which takes a number from 1 (slowest) to 10 (fastest). In addition, the number 0 means that no animation takes place, and the picture is drawn as quickly as possible. We can stop the turtle getting in the way of our final picture by using the `hideturtle` method (it has an opposite in `showturtle`). Try this:

```
Python
>>> t.hideturtle()
>>> t.speed(0)
>>> star(0, 0, 200, 7)
>>> t.showturtle()
```

The method `pensize` can be used to change the thickness of the trail. The `pencolor` method may be used to change the colour. The default pen width is 1 and, as we know, the default pen colour is black, which is the same as the red-green-blue triple (0, 0, 0). Consider this sequence of commands, where we use various shades of grey from black (0, 0, 0) to white (1, 1, 1):

```
Python
>>> t.pensize(20)
>>> star(0, 0, 200, 7)
>>> t.pensize(15)
>>> t.color(0.25, 0.25, 0.25)
>>> star(0, 0, 200, 7)
>>> t.pensize(10)
>>> t.color(0.5, 0.5, 0.5)
>>> star(0, 0, 200, 7)
>>> t.pensize(5)
>>> t.color(0.75, 0.75, 0.75)
>>> star(0, 0, 200, 7)
>>> t.pensize(2)
>>> t.color(1, 1, 1)
>>> star(0, 0, 200, 7)
>>> t.hideturtle()
```

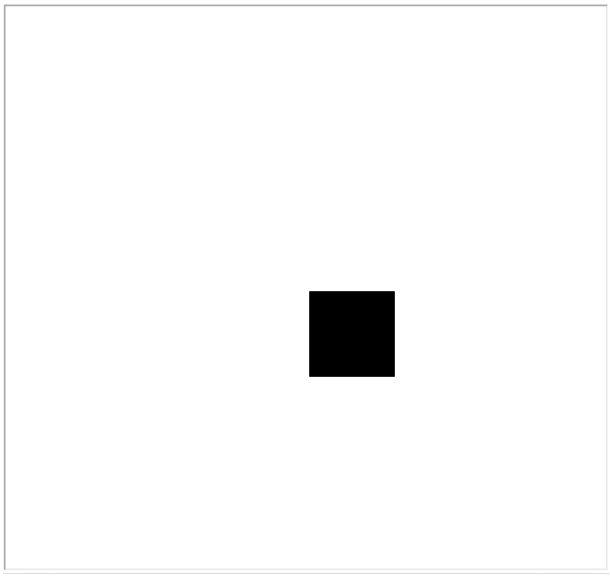
Here is the result:



QUESTION 5 Write a program to display the whole gamut of colours available in the RGB space. That is to say, all combinations of red, green and blue, with a reasonable granularity – perhaps steps of 0.1.

The `turtle` module provides its own functions for drawing filled shapes:

```
Python
>>> t.begin_fill()
>>> t.fd(100)
>>> t.rt(90)
>>> t.fd(100)
>>> t.rt(90)
>>> t.fd(100)
>>> t.rt(90)
>>> t.fd(100)
>>> t.end_fill()
>>> t.hideturtle()
```



The fill colour can be set with `fillcolor`. To make a filled shape with no border, make sure the pen is up.

QUESTION 6 Modify your circle program to draw a filled circle.

Some of our arrangements of stars were prettier than others. Let us write a program to allow the user to see one after another, and when a nice one appears, to save it to file. Each image will be drawn in turn, requiring the Space key to be pressed to go to the next one. Pressing 's' instead will save the file. Pressing 'x' will quit the program. We shall be using our usual `star` function.

To begin with, we shall define three functions, to make the display, to leave the program, and to save the picture, to be triggered by the Space key, 'x' key, and 's' key respectively:

```
import sys

def many_stars():
    t.clear()
    for _ in range(20):
        star(random.randint(-300, 300),
             random.randint(-300, 300),
             random.randint(10, 150),
             random.randint(3, 30))

def stars_exit():
    sys.exit(0)

def save_stars():
    turtle.Screen().getcanvas().postscript(file='stars.ps')
    stars_exit()
```

The function `exit` from the `sys` module exits the current program (the 0 indicates an ordinary exit, as opposed to one caused by an error). It is preferable to the plain `exit()` we have been using thus far, which is intended only for use in interactive Python. The long line in `save_stars` is an incantation (which you need not understand) to save the contents of the turtle screen to the file `stars.ps`. This a so-called PostScript file, which you may open on your computer to show it. PostScript is quite an old-fashioned format, so you might need to download a program to view it.

Now, we set up the screen, hiding the turtle, setting the speed to maximum and turning animation off. We then tell the `turtle` module we wish to link certain keys to certain of our functions: they will be run each time the given key is pressed.

```
t.hideturtle()
turtle.Screen().tracer(0, 0)
turtle.Screen().onkey(save_stars, 's')
turtle.Screen().onkey(many_stars, ' ')
turtle.Screen().onkey(stars_exit, 'x')
```

Finally, we ask the turtle module to listen to the window for these keys, run our `many_stars` function once to draw the first pattern, and call `turtle.mainloop()` to begin listening for the keys.

```
turtle.listen()
many_stars()
turtle.mainloop()
```

The `turtle.mainloop()` line must be the last statement in the program. Here is the whole program:

```
import turtle
import random
import sys

def star(x, y, l, n):
    t.penup()
    t.goto(x, y)
    t.setheading(random.randint(0, 359))
    t.pendown()
    for _ in range(n):
        t.fd(l)
        t.bk(l)
        t.rt(360.0 / n)

def many_stars():
    t.clear()
    for _ in range(20):
        star(random.randint(-300, 300),
             random.randint(-300, 300),
             random.randint(10, 150),
             random.randint(3, 30))

def stars_exit():
```

```
    sys.exit(0)

def save_stars():
    turtle.Screen().getcanvas().postscript(file='stars.ps')
    stars_exit()

t = turtle.Turtle()

t.hideturtle()
turtle.Screen().tracer(0, 0)
turtle.Screen().onkey(save_stars, 's')
turtle.Screen().onkey(many_stars, ' ')
turtle.Screen().onkey(stars_exit, 'x')
turtle.listen()

manystars()
turtle.mainloop()
```

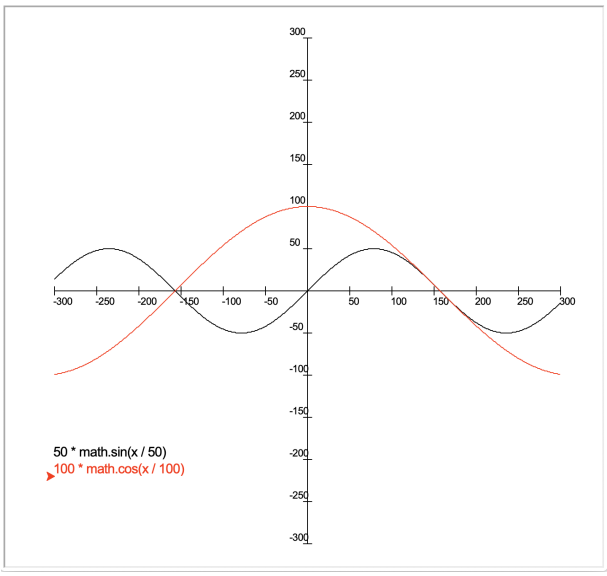
As well as key presses, we can detect mouse clicks, by using the function `Screen().onscreenclick` providing a function of which takes the x and y coordinates of the click. Here is a program to draw a star at any location clicked by the user:

```
import turtle
import random

def star(x, y, l, n):
    t.penup()
    t.goto(x, y)
    t.setheading(random.randint(0, 359))
    t.pendown()
    for _ in range(n):
        t.fd(l)
        t.bk(l)
        t.rt(360.0 / n)

def draw_star(x, y):
    star(x, y, random.randint(10, 150), random.randint(3, 30))
    turtle.Screen().update()

t = turtle.Turtle()
turtle.Screen().tracer(0, 0)
t.hideturtle()
turtle.listen()
turtle.Screen().onscreenclick(draw_star)
turtle.mainloop()
```

Now let us use what we have learned to write two more substantial programs.

PROJECT 1A: A GRAPH PLOTTER

Write a program which takes one or more formulae on the command line and plots them. For example, we might see:



For the text on the axes, and for the labels, you will need to use the turtle function `write`. For example, the following will write the text 'Hello' at the current position in 16pt Arial:

```
Python
>>>t.write('Hello', font = ('Arial', 16, 'normal'))
```

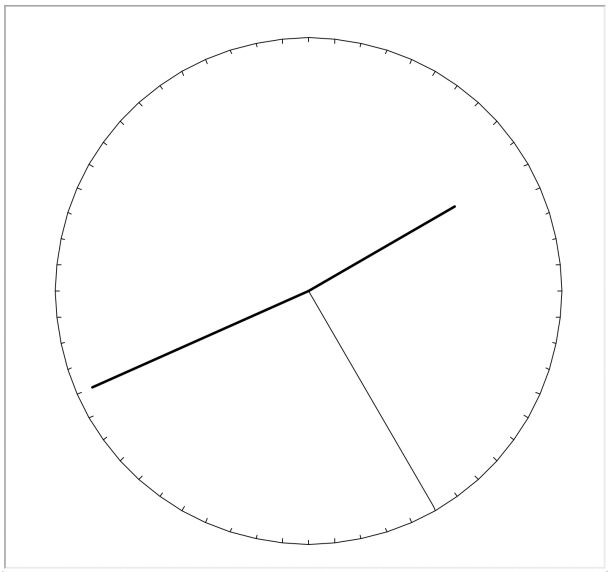
Remember that the built-in Python function `eval` can evaluate a given piece of Python program. For example, if the variable `x` has value `10` the result of `eval('x * 2')` is `20`.

The answer to this first part can be found at the back of the book; answers to the following extensions are not given.

EXTENSIONS:

- Allow the axes to be set on the command line, including scaling in x and y directions.
- Use `turtle.Screen().input` to ask for the formulae, if none are given on the command line – the program is then interactive.
- Allow for the plotting of graphs using polar coordinates, graphs parameterised in terms of x and y, and so on.

PROJECT 1B: A CLOCK Write a clock program, which displays the current time on an analog clock, updating once a second. For example:



You will need the time module for this. If we have a function `clockface`, we can pass it the current time like this:

```
import time

tm = time.localtime()

clockface(tm.tm_hour, tm.tm_min, tm.tm_sec)
```

The answer to this first part can be found at the back of the book; answers to the following extensions are not given.

EXTENSIONS:

- Add the hour labels 1..12 to the clock.
- Make a prettier clock face and prettier hands, perhaps based on a clock in your house.
- Add a digital clock and an alarm function.