In [1]:

```
import turtle
```

In [2]:

```
# The Screen
# Create a seperate screen to draw shapes
s = turtle.getscreen()
```

Create your first Turtle Object

A turtle object is like a pen on the paper. The object gives you access to properties of the pen and some control over the movement of the pen. We will see these in the next few examples.

In [3]:

```
# Creating a Turtle object (pen)
p = turtle.Turtle()
```

The screen will act as a canvas to draw on it and you can program the turtle to move around the screen a turtle has certain charactristics that you can change such as Size, color, and speed

Moving the Turtle

There are four directions that a turtle can move: Forward Backward

and rotate:

- Left
- Right

In [4]:

```
# move the turtle
p.right(90) # or p.rt(90)
p.forward(100)# or p.fd(100)
p.left(90) # or p.lt(90)
p.backward(100) # or p.bk(100)
```

Move the Turtle by coordinates

The turtle is initiated at point (0,0). You can give coordinates of the next move. The point (0,0) is called the home. You can also ask the turtle to go back to home. p.home or p.goto(0,0)

In [5]:

```
p.goto(100,100) # go to coordinate (100,100)
p.home() # go to home
```

Drawing a shape: Square

In [6]:

```
p.fd(100)
p.rt(90)
p.fd(100)
p.rt(90)
p.fd(100)
p.rt(90)
p.fd(100)
```

You can change the turtle shape

- Square
- Arrow
- Circle
- Turtle
- Triangle
- Classic

You can use the method shape("shape type") to change the shape of the pointer.

Try it yourself.

In [7]:

```
p.speed(1)
p.forward(100)
p.speed(10)
p.forward(100)
```

Customisations

- Pen Colour: pencolor()
- Fill Colour: fillcolor()
- Pen Size: pensize()
- Pen Speed: speed()

Drawing a circle

- Circle: circle(radius) Pay attention that the centre of the circle is not set by the method. The circle will start from the current position of the pointer.

You can also create a dot using the method dot(). Try the following example:

In [8]:

```
p.circle(90)
p.goto(200,200)
p.dot(20)
p.home
```

Out[8]:

```
<bound method TNavigator.home of <turtle.Turtle object at 0x000002DD40940670
>>
```

Changing the Screen Color

You can change the colour of the screen background using the bgcolor() method.

In [ ]:

```
#turtle.bgcolor("blue")
turtle.bgcolor("white")
```

Chaning the Screen Title

You can set the title of the screen using the title() method.

In [9]:

```
turtle.title("My Turtle Program")
```

Cleaning the Screen

In [10]:

```
p.clear()
```

Pen Up/Down

Sometimes when you are drawing a shape, you may wish to put up the pen and put it down at another point in the page.

- penup()
- pwndown()

In [11]:

```
p.fd(100)
p.rt(90)

p.penup()
p.fd(100)
p.rt(45)

p.pendown()
p.fd(100)
p.rt(45)

p.penup()
p.fd(100)

p.pendown()
```

Filling in a shape

To fill in a shape, you can start a filling environment, once a closed-shape is sketched, it will be filled with a given color. Let's see an example.

The method called fillcolor() will change the color of the pointer as well. This will not change the color of the paintings, only the filling parts.

In [12]:

```
p.goto(-10,-10)
p.fillcolor("orange")
p.begin_fill()
p.circle(30)
p.end_fill()
```

Cloning a Turtle

You can use the clone() method to replicate a turtle. r=p.clone()

In [13]:

```
# replicating the current turtle with all of its properties
r = p.clone()
p.color("magenta")
r.color("red")
p.circle(70)
r.circle(40)
```

Leaving a Stamp

You can leave a stamp along the lines. A stamp is a imprint of the turtle (pointer shape). You can use the stamp() method to achieve this.

In [17]:

```
p.rt(-90)
p.fd(100)
p.stamp()
p.fd(100)
```

Undo

You can undo a change using undo() method.

In [87]:

```
p.rt(-90)
p.fd(100)
p.stamp()
p.fd(100)
p.undo()
```

Reseting the Environment

You can use the reset() method to reset the environemnt. This will clear the screen and the pointer to its home position (0,0).

In [88]:

```
p.reset()
```

Loops: Iterate and create similar shapes

for Loops

let's assume that we want to create several circles with different radius. A method to achieve this is simply drawing the circles one by one (lines of code). p.circle(10) p.circle(20) p.circle(30) p.circle(40)

However, we can use a for loop to repeat the process and increase/decrease the radius.

In [91]:

```
for i in range(5):
    p.circle(10*i)
p.circle(500*i)
```

While loop

A for loop will iterate a selection of instructions for a give nummber of times. A while loop will iterate a selection of instructions until certain condition(s) is/are met.

Let's look at the example below.

In [92]:

```
radius=1
counter=1
p.goto(-100,100)
while radius<500:
    counter=counter+1
    radius=10*counter
    p.circle(radius)
    p.speed(radius)
```

Conditional Statements

Conditional statements are used for comparison and checking if certain condition is met.

The == operator is used for checking if two sides of the operator are equal.

You can run a set of instruction if certain condition is met. Let's see an example.

In [ ]:

```
# Asking for user input
in1 = input("Would you like me to draw a shape? y/n: ")
# note that after running the code, you will have to respond to the question with a yes

if in1 == "y":
    p.pencolor("purple")
    p.circle(55)
elif in1=="n":
    print("No shapes have been added")
else:
    print("invalid reply")
```