In [1]:
```
import turtle
```

In [2]:
```
# Speed of the turtle (global parameter)
TURTLE_SPEED = 0  # 0 = no animation (set values 1 - 10 for slow/fast ani
```

In [3]:
```
def draw_nested_squares(tt, origin, dist, nsquares):
    """Draws nested squares.

    Args:
        tt: Turtle instance to use for the drawing.
        origin: x,y coordinates of the starting point.
        dist: Distance between consecutive squares.
        nsquares: Nr. of squares to draw.
    """
    tt.speed(TURTLE_SPEED)
    tt.up()
    tt.goto(origin[0], origin[1])  # for experts: tt.goto(*origin)
    tt.setheading(0)
    tt.down()
    for isquare in range(nsquares):
        for _ in range(4):
            tt.forward(isquare * dist)
            tt.left(90)
    tt.up()
```

In [4]:
```
def draw_circle_spiral(tt, origin, radinc, anginc, ncircles):
    """Draws a spiral of circles.

    Args:
        tt: Turtle instance to use for the drawing.
        origin: x, y coordinates of the starting point.
        radinc: Increment of the circle radii between consecutive circles
        anginc: Increment of the starting angle between consecutive circl
        ncircles: Nr. of circles to draw.
    """
    tt.speed(TURTLE_SPEED)
    tt.up()
    tt.goto(origin[0], origin[1])
    tt.setheading(0)
    tt.down()
    for icircle in range(ncircles):
        tt.circle((icircle + 1) * radinc)
        tt.left(anginc)
    tt.up()
```

In [5]:
```python
# Function definition with additional type annotations.

def draw_vasarely_star(
    tt: turtle.Turtle,
    origin: tuple[int, int],
    linesep: int,
    nlines: int
):
    """Draws a Vasarely star.

    Args:
        tt: Turtle instance to use for the drawing.
        origin: x,y coordinates of the starting point.
        linesep: Separation between consecutive lines (on the axes).
        nlines: Nr. of lines to draw on each half-axis.
    """
    tt.speed(TURTLE_SPEED)
    size = nlines * linesep
    origx, origy = origin
    for xsign in [-1, 1]:
        # Make sure vertical axes are plotted (but only once, not twice)
        niters = nlines if xsign == -1 else nlines + 1
        for ysign in [-1, 1]:
            for ii in range(niters):
                tt.up()
                tt.goto(origx, origy + ysign * ii * linesep)
                tt.down()
                tt.goto(origx + xsign * (size - ii * linesep), origy)
    tt.up()
```

In [6]:
```python
tt = turtle.Turtle()
```

In [7]:
```python
tt.reset()
```

In [8]:
```python
draw_nested_squares(tt, (-200, -300), 20, 10)
```

In [9]:
```python
draw_circle_spiral(tt, (-150, 250), 3, 10, 50)
```

In [10]:
```python
draw_vasarely_star(tt, (200, 0), 20, 15)
```

In [11]:
```python
tt.hideturtle()
```

Uncomment next line, if you want to be able to resize / move the canvas with the plot.
The next command will then wait, until the canvas window is closed.

In [12]:
```python
# turtle.done()
```

In [ ]:
```python

```