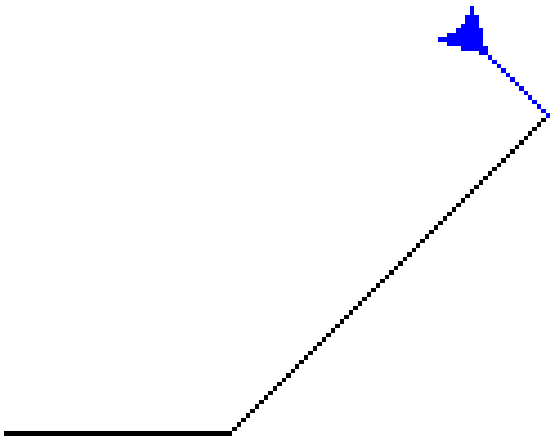
Turtles and Recursion

- Please read 5.8-5.10 about recursion
- Please read 4.2-4.6 about turtles

Turtles

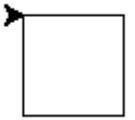
```
import turtle

turtle.forward(50)          # move turtle forward fifty units
turtle.left(45)             # turn turtle left 45 degrees
turtle.forward(100)         # move turtle forward 100 units
turtle.color("blue")        # make turtle blue
turtle.right(90)            # turn right 90 degrees
turtle.backward(20)         # go backward
```



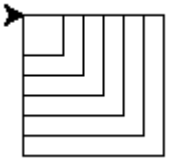
Draw a square of any size

```
>>> square(20)
```

```
>>> square(50)
```



```
>>> for i in [20, 30, 40, 50, 60, 70]:
        square(i)
```



How to write the `square` function?

Use a loop, `turtle.forward` and `turtle.right`.

Draw a square of any size

```
import turtle

def square(n):
   for i in range(4):
      turtle.forward(n)
      turtle.right(90)
```

```
import turtle

turtle.up()
turtle.goto(0, -100)
turtle.down()

turtle.begin_fill()
turtle.fillcolor("yellow")
turtle.circle(100)
turtle.end_fill()

turtle.up()
turtle.goto(-67, -40)
turtle.setheading(-60)
turtle.width(5)
turtle.down()
turtle.circle(80, 120)

turtle.fillcolor("black")
for i in [-35, 35]:
    turtle.up()
    turtle.goto(i, 35)
    turtle.setheading(0)
    turtle.down()
    turtle.begin_fill()
    turtle.circle(10)
    turtle.end_fill()

turtle.hideturtle()
```

Filled yellow circle

Smile

Two eyes

Recursion

Execution analysis 1

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
Entered func_b, x=10
Exiting func_b, x=10
Exiting func_a, s=lenochod
Exiting main()
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

Stack

main()

Output:

Entered main()

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

Stack

main()

Output:

Entered main()

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

s : "lenochod"

Stack

func_a("lenochod")
main()

Output:

Entered main()
Entered func_a, s=lenochod

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

s : "lenochod"

Stack

func_a("lenochod")
main()

Output:

Entered main()
Entered func_a, s=lenochod

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

x : 5

Stack

func_b(5)
func_a("lenochod")
main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

x : 5

Stack

func_b(5)
func_a("lenochod")
main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

s : "lenochod"

Stack

func_a("lenochod")
main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

x : 10

Stack

func_b(10)
func_a("lenochod")
main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
Entered func_b, x=10
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

x : 10

Stack

func_b(10)
func_a("lenochod")
main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
Entered func_b, x=10
Exiting func_b, x=10
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

s : "lenochod"

Stack

func_a("lenochod")
main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
Entered func_b, x=10
Exiting func_b, x=10
Exiting func_a, s=lenochod
```

```
def func_b(x):
    print("Entered func_b, x=" + str(x))
    print("Exiting func_b, x=" + str(x))

def func_a(s):
    print("Entered func_a, s=" + s)
    func_b(5)
    func_b(10)
    print("Exiting func_a, s=" + s)

def main():
    print("Entered main()")
    func_a("lenochod")
    print("Exiting main()")
```

Local variables

Stack

main()

Output:

```
Entered main()
Entered func_a, s=lenochod
Entered func_b, x=5
Exiting func_b, x=5
Entered func_b, x=10
Exiting func_b, x=10
Exiting func_a, s=lenochod
Exiting main()
```

Execution analysis 2

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 0

Stack

main(0)

Output:

Entered main, x=0

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 0

Stack

main(0)

Output:

Entered main, x=0

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 1

Stack

main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 1

Stack

main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 2

Stack

main(2)
main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1
Entered main, x=2

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 2

Stack

main(2)
main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1
Entered main, x=2

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 3

Stack

main(3)
main(2)
main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1
Entered main, x=2
Entered main, x=3

```
def main(x):
    print("Entered main, x="+str(x))
    main(x+1)
    print("Exiting main, x="+str(x))
```

Local variables

x : 996

Stack

main(996)
...
main(3)
main(2)
main(1)
main(0)

Output:

```
...
Entered main, x=996
RecursionError: maximum recursion depth exceeded while
calling a Python object
Entered main, x=996
```

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
        main(x+1)
    else:
        print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 0

Stack

main(0)

Output:

Entered main, x=0

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
        main(x+1)
    else:
        print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 0

Stack

main(0)

Output:

Entered main, x=0

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
       main(x+1)
    else:
       print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 1

Stack

main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
        main(x+1)
    else:
        print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 1

Stack

main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
       main(x+1)
    else:
       print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 2

Stack

main(2)
main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1
Entered main, x=2

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
        main(x+1)
    else:
        print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 2

Stack

main(2)
main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1
Entered main, x=2
It's quittin' time

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
       main(x+1)
    else:
       print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 2

Stack

main(2)
main(1)
main(0)

Output:

Entered main, x=0
Entered main, x=1
Entered main, x=2
It's quittin' time
Exiting main, x=2

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
        main(x+1)
    else:
        print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 1

Stack

main(1)
main(0)

Output:

```
Entered main, x=0
Entered main, x=1
Entered main, x=2
It's quittin' time
Exiting main, x=2
Exiting main, x=1
```

```
def main(x):
    print("Entered main, x="+str(x))
    if x < 2:
        main(x+1)
    else:
        print("It's quittin' time")
    print("Exiting main, x="+str(x))
```

Local variables

x : 0

Stack

main(0)

Output:

```
Entered main, x=0
Entered main, x=1
Entered main, x=2
It's quittin' time
Exiting main, x=2
Exiting main, x=1
Exiting main, x=0
```

Factorial

- fac(4) == 4 * 3 * 2 * 1 == 24
- fac(5) == 5 * 4 * 3 * 2 * 1 == 120
- Therefore, fac(5) == 5 * fac(4)
- More generally, factorial of x:

  fac(x) == 1 when x == 1

  fac(x) == x * fac(x-1) when x != 1

Factorial

```
def fac(x):
    """
    signature: int -> int
    calculates factorial of x
    """
    if x == 1:
       return 1
    else:
      old_fac = fac(x-1)
       result = x * old_fac
       return result
```

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables
x : 4

Stack
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 4

Stack

fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 3

Stack

fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 3

Stack

fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 2

Stack

fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 2

Stack

fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 1

Stack

fac(1)
fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 1

Stack

fac(1)
fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 2
old_fac : 1

Stack

fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 2
old_fac : 1
result : 2

Stack

fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 2
old_fac : 1
result : 2

Stack

fac(2)
fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 3
old_fac : 2

Stack

fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 3
old_fac : 2
result : 6

Stack

fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 3
old_fac : 2
result : 6

Stack

fac(3)
fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 4
old_fac : 6

Stack

fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 4
old_fac : 6
result : 24

Stack

fac(4)

```
def fac(x):
    if x == 1:
        return 1
    else:
        old_fac = fac(x-1)
        result = x * old_fac
        return result
```

Local variables

x : 4
old_fac : 6
result : 24

Stack

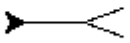
fac(4)

Turtles and recursion

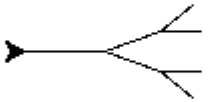
```
import turtle

def tree(branchLen):
    if branchLen > 10:
        turtle.forward(branchLen)
        turtle.right(20)
        tree(branchLen - 10)
        turtle.left(40)
        tree(branchLen - 10)
        turtle.right(20)
        turtle.backward(branchLen)
```

tree(20)

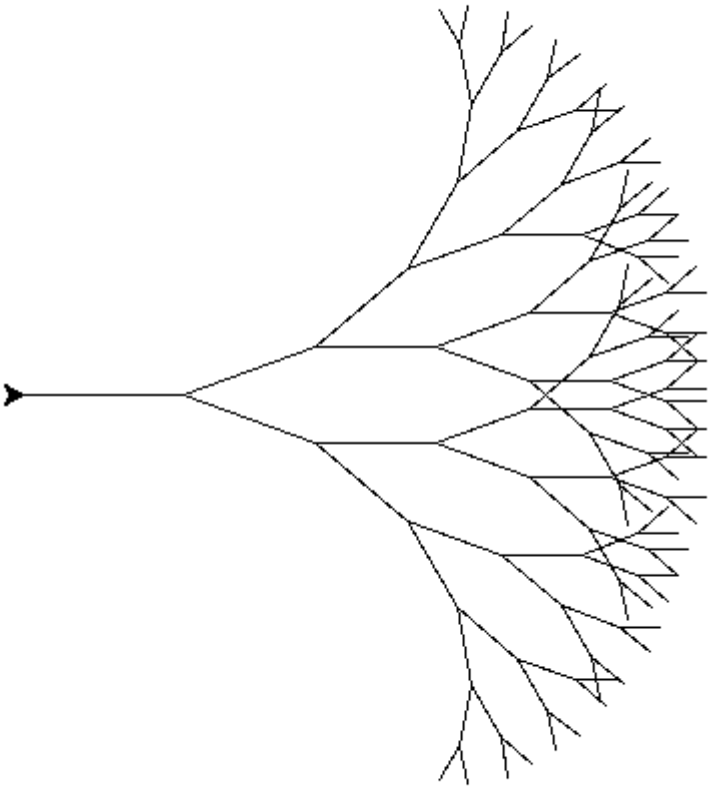
tree(30)



tree(40)

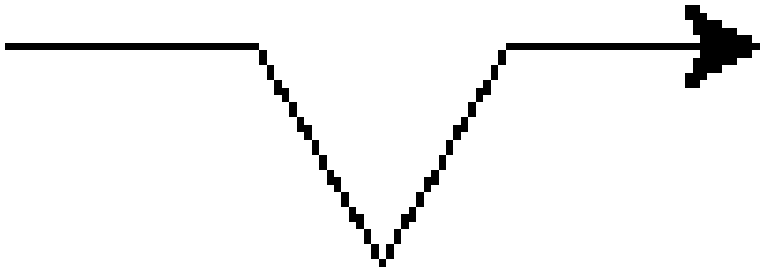


tree(80)

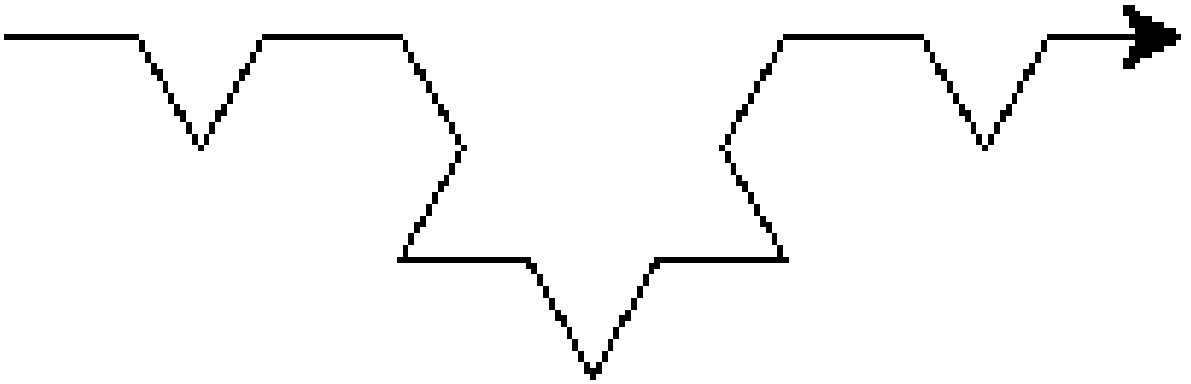


```
import turtle

def snowflake(length, depth):
    if depth == 0:
      turtle.forward(length)
    else:
      snowflake(length/3, depth-1)
      turtle.right(60)
      snowflake(length/3, depth-1)
      turtle.left(120)
      snowflake(length/3, depth-1)
      turtle.right(60)
      snowflake(length/3, depth-1)
```

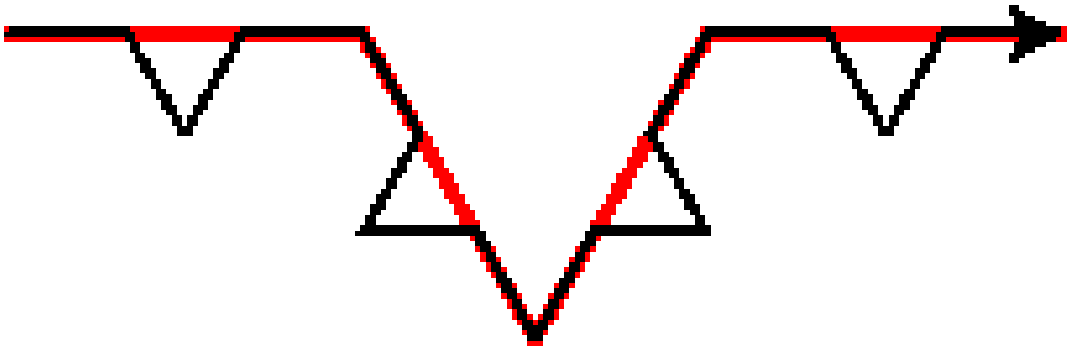
snowflake(200, 1)



snowflake(200, 2)

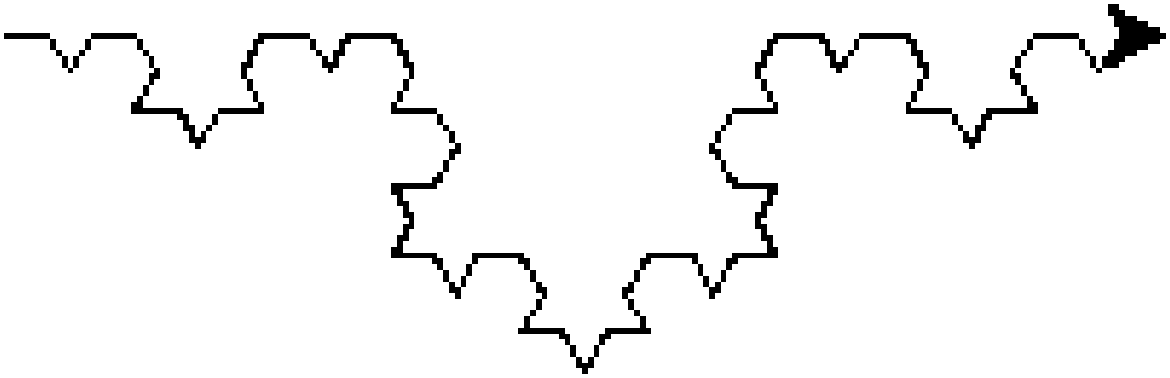


snowflake(200, 2)
snowflake(200, 1)



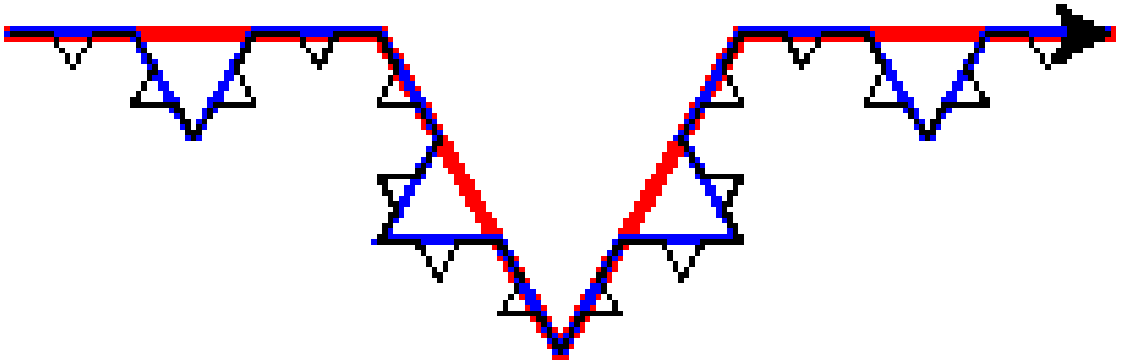
snowflake(200, 3)

- `

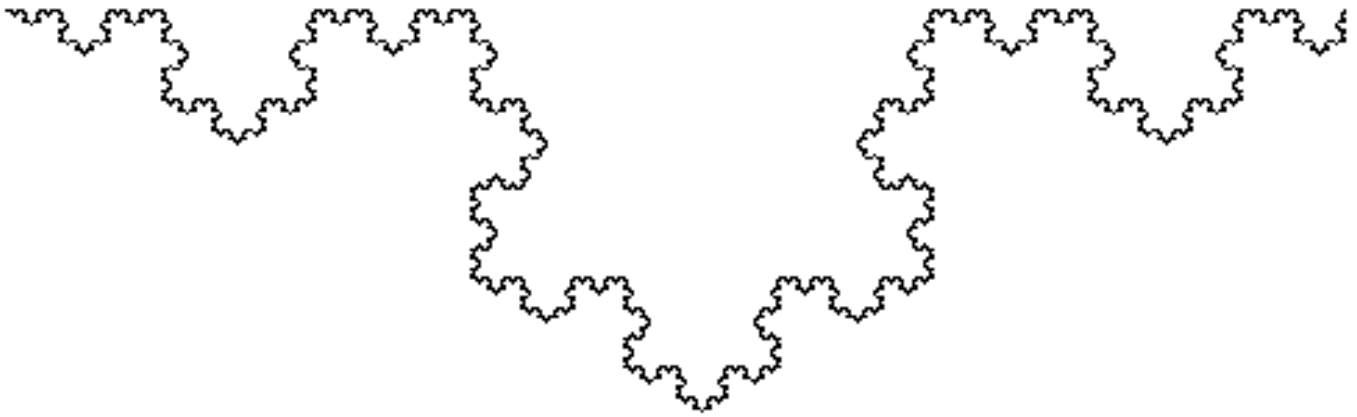


snowflake(200, 1)
snowflake(200, 2)
snowflake(200, 3)

- `



snowflake(500, 5)



Interactivity with turtles

```
import turtle

def drawcircle(x, y):
    """
    Draw a circle at the given coordinates
    """
    turtle.up()
    turtle.goto(x, y)
    turtle.down()
    turtle.circle(5)

def sayhello():
    turtle.write("hello!", False, 'left', ('Times New Roman', 36, 'bold'))

# For the rest of the program,
# the turtle will call drawcircle
# when the user clicks the mouse
turtle.onscreenclick(drawcircle)

# For the rest of the program,
# the turtle iwll call sayhello
# when the user presses the space key
turtle.onkey(sayhello, "space")

# Make the turtle listen for keyss
turtle.listen()

# Wait forever
turtle.mainloop()
```