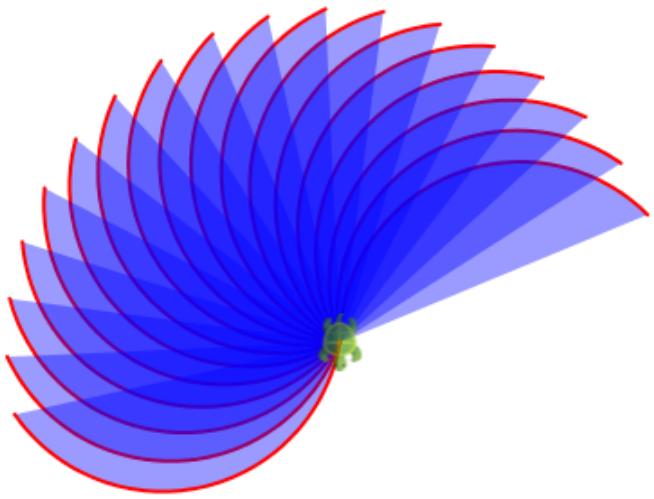
Programming Fundamentals

with Kojo



by

Lalit Pant

Version: March 11, 2015

A Note for Facilitators and Teachers

This book contains a series of activities for kids to play with.

Most activities contain a fully defined program and a picture of the output of the program. For such activities, ask a kid to type in the program inside the script editor, run it, and then check that the actual output of the program matches the output shown in the book. Then, ask the the kid to do some reflection, i.e., think about and discuss what was just learned.

Some activities contain an incomplete program, with the incomplete areas marked with `???`, and a picture of the output of the (complete) program. For such activities, ask a kid to type in the program inside the script editor, fill out the incomplete portions of the program, run it, and then (as before) check that the actual output of the program matches the output shown in the book.

At every step, encourage the following:

- exploration, discovery, and a sense of play.
- perseverance in the face of unexpected results, and joy in the process of figuring out what went wrong.
- reflection and discussion about what was learned.
- digressions and diversions from the provided sequence of activities.

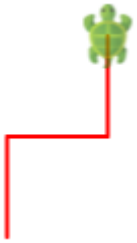
It is not important to finish all the activities. But it is vitally important to spend time with, go deep into, enjoy, and learn from each activity!

```
clear()
forward(50)
```

```
clear()
forward(50)
right(90)
forward(50)
right(90)
```

```
clear()
repeat(4) {
    forward(50)
    right(90)
}
```

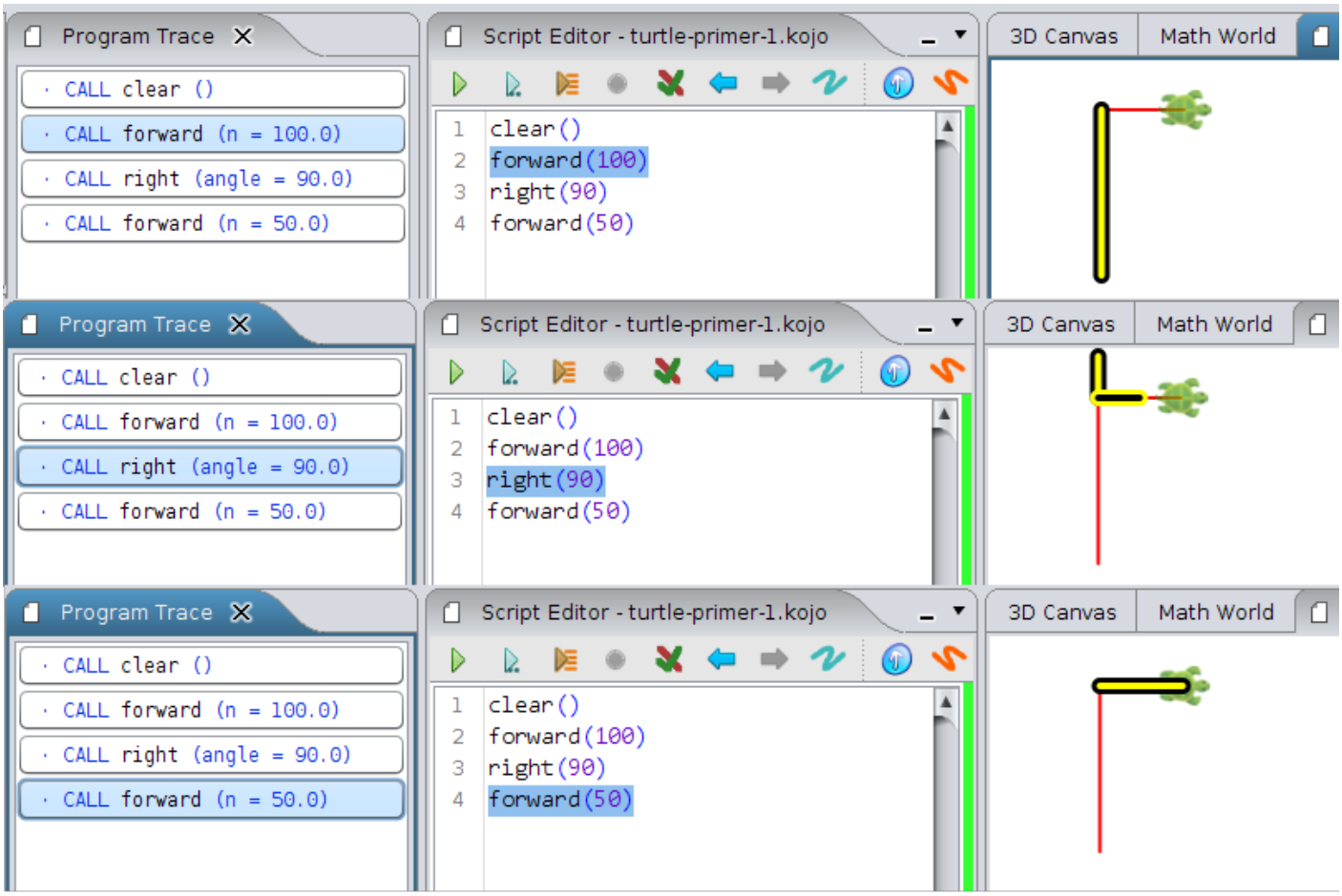
```
clear()
forward(50)
right(90)
forward(50)
left(90)
forward(50)
```



```
clear()
setAnimationDelay(100)
repeat(4) {
    forward(50)
    right(90)
    ???
}
```

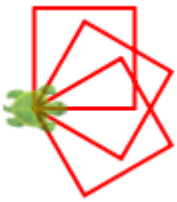
```
clear()
setAnimationDelay(100)
repeat(10) {
    forward(10)
    ???
}
```

```
// run this program with the tracing button shown above
clear()
forward(100)
right(90)
forward(100)
// trace shown below
```



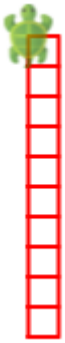
```
def square() {
    repeat(4) {
        forward(50)
        right(90)
    }
}
clear()
square()
```

```
def square() {
    // same as before
}
clear()
setAnimationDelay(100)
repeat(3) {
    square()
    right(30)
}
```



```
def square() { /* same as before */ }
clear()
setAnimationDelay(100)
repeat(2) {
    square()
    hop(25)
    right(90)
    hop(25)
    left(90)
}
```

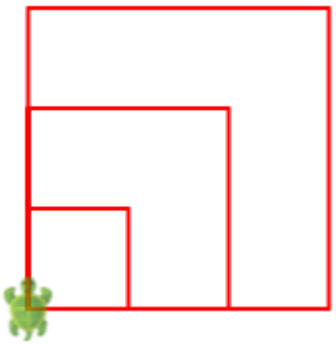
```
def square() { /* similar to before; size 15 */ }
clear()
setAnimationDelay(10)
repeat(10) {
    ???
}
```



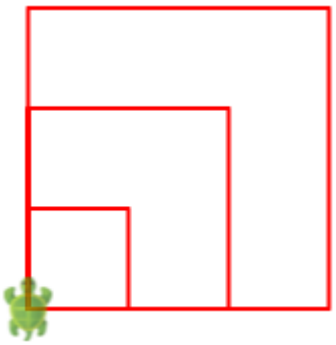
```
def square() { /* same as before */ }
def ladder() {
    setPenColor(randomColor)
    ???
}
clear()
setAnimationDelay(10)
ladder()
```

```
def square() { /* same as before */ }
def ladder() {/* same as before */ }
clear()
setAnimationDelay(10)
setPenThickness(4)
repeat(10) {
    ladder()
    ???
}
```

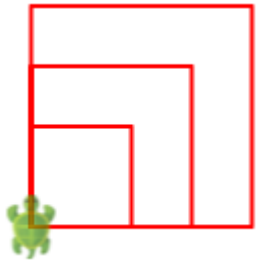
```
def square(n: Int) {
    repeat(4) {
        forward(n)
        right(90)
    }
}
clear()
square(50)
square(100)
square(150)
```



```
def square(n: Int) {
    // same as before
}
clear()
setAnimationDelay(100)
repeatFor(1 to 3) { n =>
    square(n * 50)
}
```

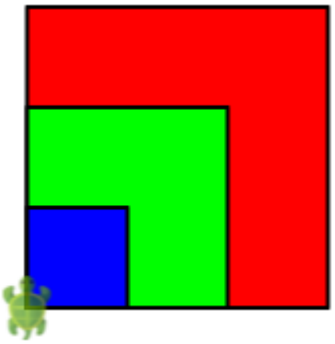


```
def square(n: Int) { /* same as before */ }
clear()
setAnimationDelay(100)
// make squares of sizes 50, 80, and 110
repeatFor(1 to 3) { n =>
    ???
}
```

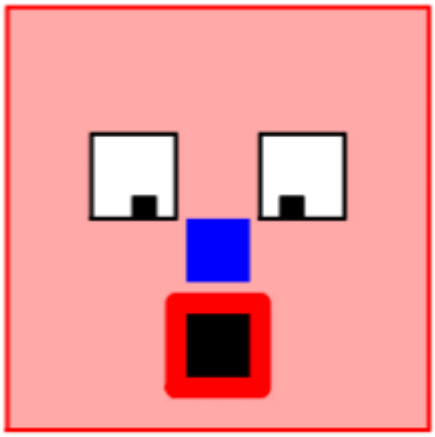


```
def square(n: Int) { /* same as before */ }
clear()
setAnimationDelay(100)
setPenThickness(20)
setBackground(yellow)
setPenColor(blue)
repeatFor(1 to 3) { n =>
    square(10 + n * 40)
}
```

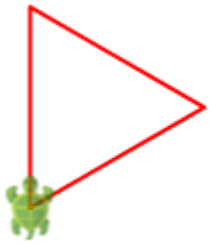
```
def square(n: Int) { /* same as before */ }
val sizes = Seq(150, 100, 50)
val colors = Seq(red, green, blue)
clear()
setAnimationDelay(100)
setPenColor(black)
repeatFor(0 to 2) { n =>
    setFillColor(colors(n))
    square(sizes(n))
}
```



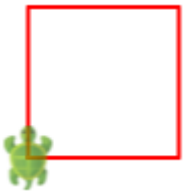
```
def square(n: Int) { /* same as before */ }
clear()
setAnimationDelay(100)
???
```



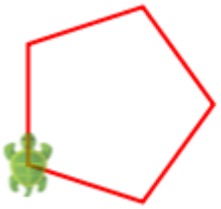
```
clear()
repeat(3) {
    forward(100)
    right(120)
}
```



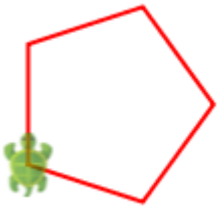
```
clear()
repeat(4) {
    forward(75)
    right(90)
}
```



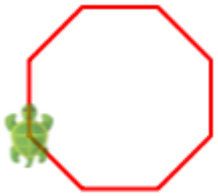
```
clear()
repeat(5) {
    forward(60)
    ???
}
```



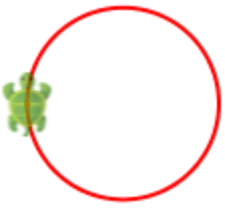
```
def polygon(sides: Int) {
    repeat(sides) {
        ???
    }
}
clear()
polygon(5)
```

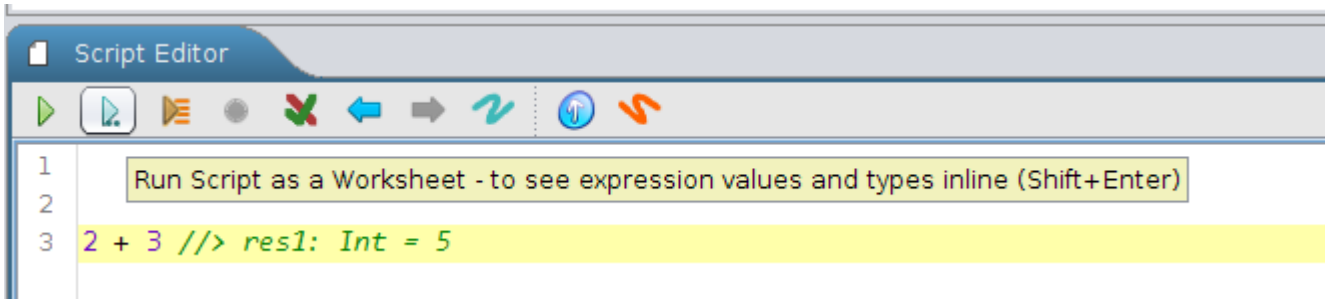


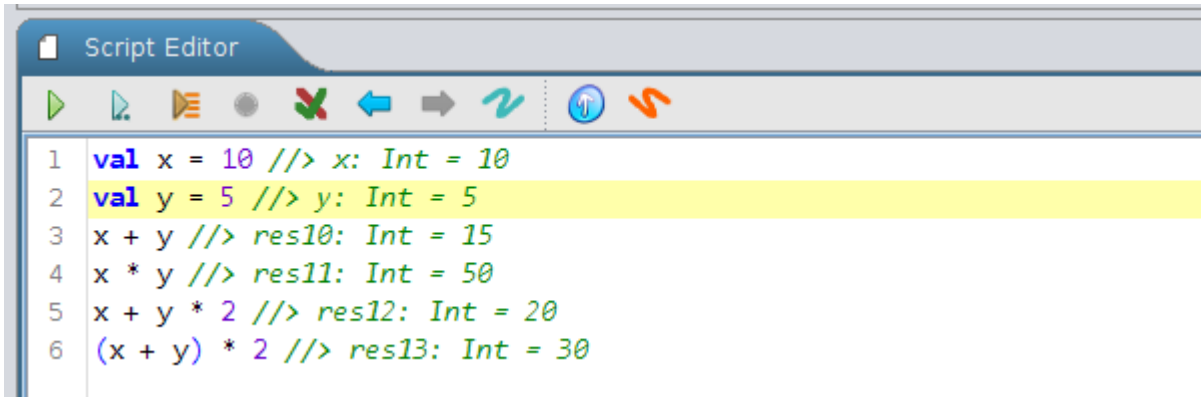
```
def polygon(sides: Int) {
    // same as before
}
clear()
polygon(8)
```

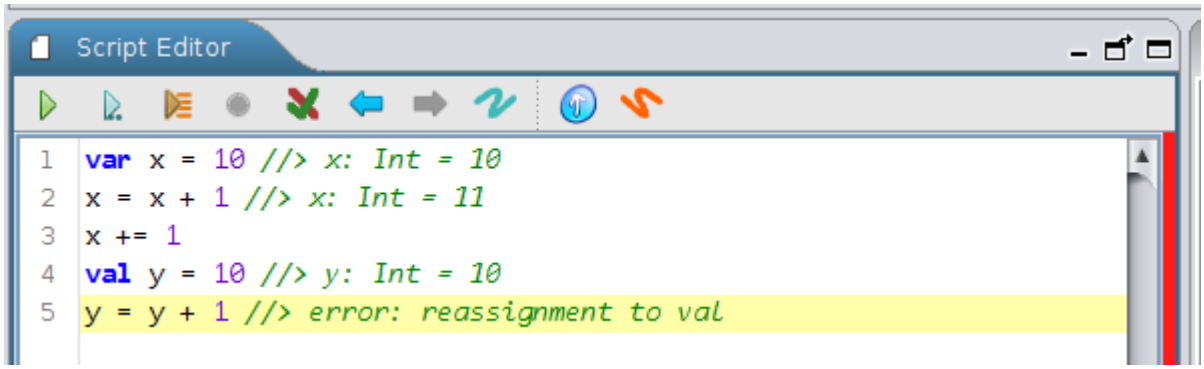


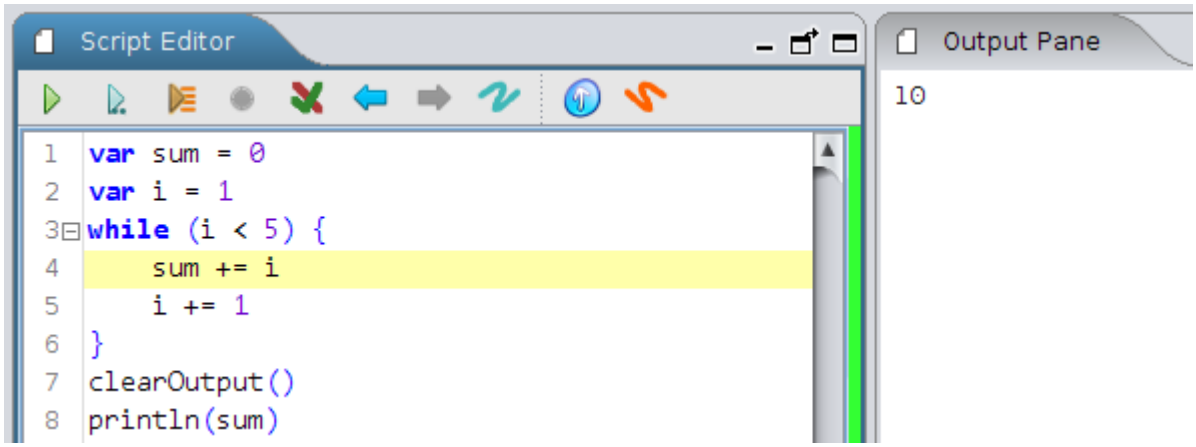
```
def polygon(sides: Int) {
    // same as before
}
clear()
setAnimationDelay(100)
polygon(???)
```




Script Editor
Run Script as a Worksheet - to see expression values and types inline (Shift+Enter)
2 + 3 //> res1: Int = 5


Script Editor
val x = 10 //> x: Int = 10
val y = 5 //> y: Int = 5
x + y //> res10: Int = 15
x * y //> res11: Int = 50
x + y * 2 //> res12: Int = 20
(x + y) * 2 //> res13: Int = 30

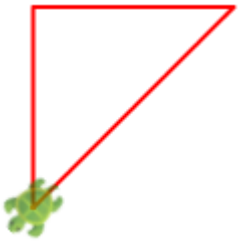

Script Editor
var x = 10 //> x: Int = 10
x = x + 1 //> x: Int = 11
x += 1
val y = 10 //> y: Int = 10
y = y + 1 //> error: reassignment to val


Script Editor
var sum = 0
var i = 1
while (i < 5) {
sum += i
i += 1
}
clearOutput()
println(sum)
Output Pane
10

Script Editor

```
def twice(n: Int) = n * 2 //> twice: (n: Int)Int
twice(5) //> res17: Int = 10
def sum(n1: Int, n2: Int) = n1 + n2 //> sum: (n1: Int, n2: Int)Int
sum(3, 4) //> res18: Int = 7
def diagonal(side1: Double, side2: Double) = {
    val dsquare = math.pow(side1, 2) + math.pow(side2, 2)
    math.sqrt(dsquare)
} //> diagonal: (side1: Double, side2: Double)Double
diagonal(3, 4) //> res19: Double = 5.0
diagonal(4, 5) //> res20: Double = 6.4031242374328485
```

```
def diagonal(side1: Double, side2: Double) = {
    // same as before
}
clear()
forward(100)
right(90)
forward(100)
right(135)
forward(diagonal(100, 100))
```

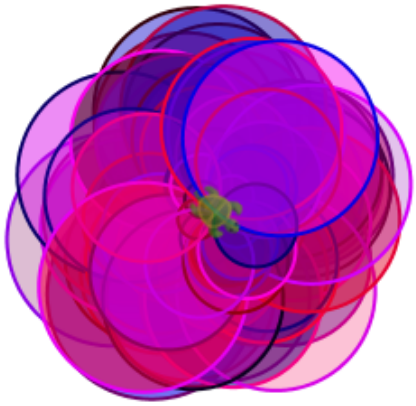


```
def diagonal(side1: Double, side2: Double) = {
    // same as before
}
var more = "yes"
while (more == "yes") {
    clear(); clearOutput()
    val s1 = readInt("First side of triangle")
    val s2 = readInt("Second side of triangle")
    val s3 = diagonal(s1, s2)
    val angle = math.atan2(s1, s2).toDegrees
    println(s"""
First side is: $s1, second side is: $s2.
The length of the diagonal is: $s3.
The angle between the second side
and the diagonal is: $angle degrees
""")
    forward(s1)
    right(90)
    forward(s2)
    right(180 - angle)
    forward(s3)
    more = readln("More triangles?")
}
```

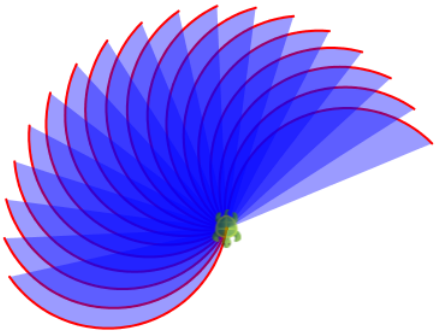
Output Pane

```
First side is: 120, second side is: 60.
The length of the diagonal is: 134.16407864998737.
The angle between the second side
and the diagonal is: 63.43494882292201 degrees
```

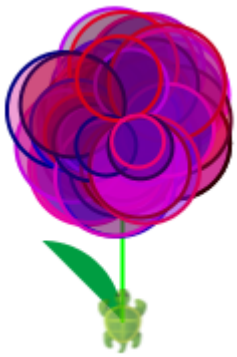
```
clear()
setAnimationDelay(10)
repeat(100) {
    setPenColor(Color(random(256), 0, random(256)))
    setFillColor(Color(random(256), 0, random(256),
    left(random(360))
    circle(random(50) + 10)
}
```



```
clear()
setAnimationDelay(10)
setFillColor(Color(0, 0, 255, 100))
repeat(18) {
    savePosHe()
    right(135, 100)
    restorePosHe()
    left(10)
}
```

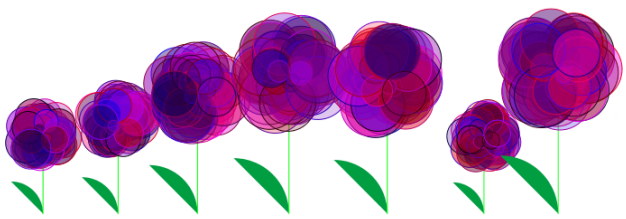


```
def flower(size: Int) {
    savePosHe()
    ???
    repeat(100) {
        ???
    }
    restorePosHe()
}
clear()
setAnimationDelay(10)
flower(20)
```



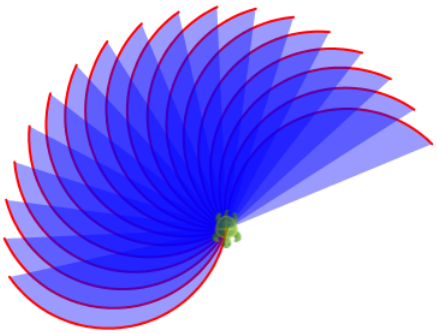
```
def flower(size: Int) {
    // same as before
}

def garden(flowers: Int) {
    repeat(flowers) {
        ???
    }
}
clear()
setAnimationDelay(10)
garden(7)
```



```
def figure(n: Int) {
    if (n < 10) {
        forward(n)
    }
    else {
        forward(n)
        right(90)
        figure(n - 5)
    }
}
clear()
figure(100)
// use tracing to understand this program
```

```
def pattern(n: Int) {
    if (n > 0) {
        savePosHe()
        right(135, 100)
        restorePosHe()
        left(10)
        pattern(n - 1)
    }
}
clear()
setAnimationDelay(10)
setFillColor(Color(0, 0, 255, 100))
pattern(18)
```



```
def tree(n: Int) {
    savePosHe()
    if (n < 10) {
        ???
    }
    else {
        forward(n)
        right(30)
        tree(n - 10)
        left(70)
        tree(n - 10)
    }
    restorePosHe()
}
clear()
setAnimationDelay(10)
setPenColor(Color(150, 95, 8))
tree(70)
```