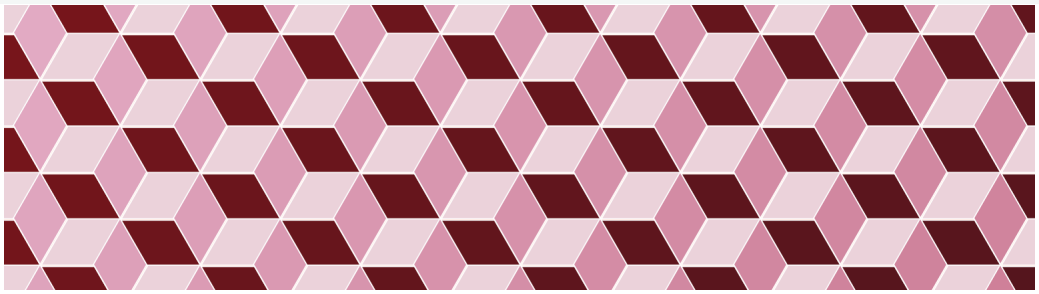


15-104 Introduction to Computing for Creative Practice

Fall 2022

31-Recursion

Instructor: Tom Cortina, tcortina@cs.cmu.edu, GHC 4117, 412-268-3514

1

Function Calls

- When a function calls another, that called function runs to completion and then returns back to where the call was made, and the program continues from there.
- This can happen in a nested fashion. A function can call another function, which calls yet another function, etc. When a function returns, it returns to where it was called.
- Example: A calls B. B calls C. C calls D.
  When D completes/returns, which function resumes? C
  Which function resumes when that function completes? B
- The computer keeps track of the functions on a system stack.

C
B
A

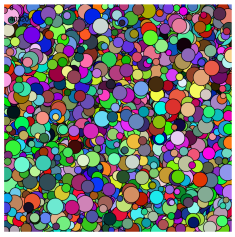
2

Recursion

- A function can call itself! Such a function is recursive.
- The computer still keeps track of where we return, even though the functions all have the same name.
- Example: Factorial
  5! = 5 * 4 * 3 * 2 * 1
  But we can also say that
  5! = 5 * 4!
- In general, n! = n * (n-1)!
- But this could go on forever, when do we stop?
  When n = 0, 0! = 1 by definition. (base case)

3

Factorial



```
var n = 0;
function setup() {
    createCanvas(400, 400); frameRate(1);
}
function draw() {
    background(220);
    var numCircles = factorial(n);
    drawCircles(numCircles);   // NOT SHOWN
    n += 1;
}
function factorial(n) {
    if (n == 0) return 1;           // base case
    return n * factorial(n-1);      // recursive case
}
```

4

Factorial: Trace

```
factorial(5) = 5 * factorial(4)
    factorial(4) = 4 * factorial(3)
        factorial(3) = 3 * factorial(2)
            factorial(2) = 2 * factorial(1)
                factorial(1) = 1 * factorial(0)
                    factorial(0) = 1
                factorial(1) = 1 * 1 = 1
            factorial(2) = 2 * 1 = 2
        factorial(3) = 3 * 2 = 6
    factorial(4) = 4 * 6 = 24
factorial(5) = 5 * 24 = 120
```

5

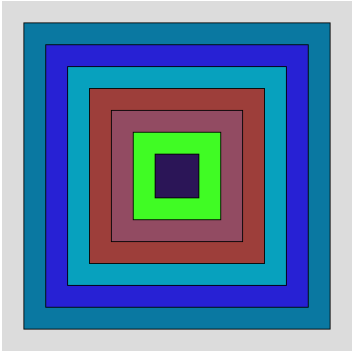
Recursive Squares

You know how to create this sketch iteratively (i.e. with a loop).
How would you create this sketch recursively?

Express the problem so its solution requires a simpler version of itself.

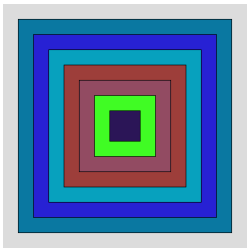
Drawing all squares starting with size s:

- If the starting size is 0, then you're done.
- Otherwise draw a square of size s and then draw all of the squares starting with size s-50. (recursive)



6

Recursive Squares



```
function setup() {
    createCanvas(400, 400); rectMode(CENTER);
}
function draw() {
    background(220);
    drawSquare(350);       // initial call
    noLoop();
}
function drawSquare(size) {
    if (size == 0) return;   // base case (non-recursive)
    fill(color(random(0,256), random(0,256), random(0, 256)));
    square(200, 200, size);
    drawSquare(size-50);  // recursive call – draw the remaining squares
                          //   starting with size-50 using itself
    return;
}
```

`return` by itself just exits the function without returning a calculated answer.

7

Recursive Squares: Trace

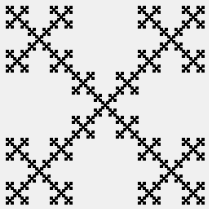
```
drawSquare(350)
    square(200, 200, 350)
    drawSquare(300)
        square(200, 200, 300)
        drawSquare(250)
            square(200, 200, 250)
            drawSquare(200)
                square(200, 200, 200)
                drawSquare(150)
                    square(200, 200, 150)
                    drawSquare(100)
                        square(200, 200, 100)
                        drawSquare(50)
                            square(200, 200, 50)
                            drawSquare(0)
```

Each recursive function (except the first call) returns back to the same function where it was called but the only thing left to do is also return.

8

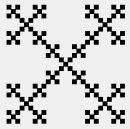
Fractal (self-similar image)

How do we create something like this?



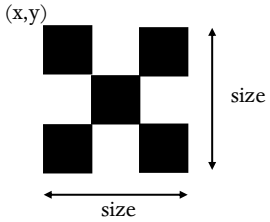
Break the problem down to drawing each of the five sections. For each of the five sections, break them down into five sections. Keep doing this until each section is of size 3 and then draw it.






9

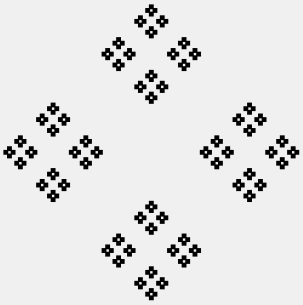
Fractal




```
function draw() {
    background(240); fill(0);
    recPattern(80, 80, 243);
}
function recPattern(x, y, size)
{
    if (size <= 3) rect(x, y, size, size);     // base case
    } else { // recursive case:
        var third = size / 3;
        recPattern(x, y, third);                              // upper left
        recPattern(x + 2 * third, y, third);                  // upper right
        recPattern(x + third, y + third, third);              // middle
        recPattern(x, y + 2 * third, third);                  // lower left
        recPattern(x + 2 * third, y + 2 * third, third); // lower right
    }
}
```

10

Try This



11