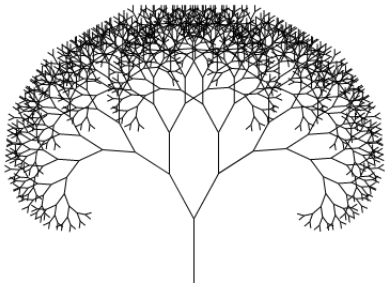
Impact Lab 2023: Programming Fundamentals
Lecture 8: Fractal Trees

Summer 2023

School of Computing
and Data Science

Wentworth Institute of
Technology



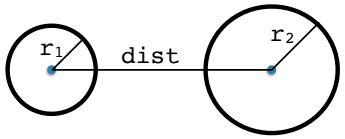
Topics for Today

- **Object Communication**
- Recursion
- Fractal Trees

Object Communication

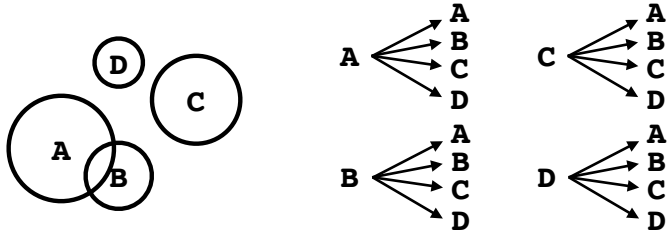
We can already tell if an object and the mouse interact, but what about multiple objects?

For circles, it's nice and easy, but for other objects it can be very complicated.



More Nested Loops

If I have many bubbles, I need to check every pairwise combination of bubbles.



So, we pick a bubble, loop though every other bubble, and check if they touch.

Topics for Today

- Object Communication
- **Recursion**
- Fractal Trees

Exercise

Write a factorial method that takes as input an integer (assumed to be >= 0) and returns as an integer the result

$$n! = \prod_{k=1}^{n} k$$

What is Recursion?

A way of programming in which a method refers to itself in order to solve a problem

Never necessary

In some situations, results in simpler and/or easier-to-write code

Can often be more expensive in terms of memory and/or time

Consider:

$$0! = 1$$ `Base Case`

$$n! = n(n-1)!$$ `Recursive Step`

When attempting to write a recursive algorithm, you must have a base case and a "step". In this case, n depends on n-1

Example

```
function factorial_r(n) {
  if (n == 0) {
    return 1;
  } else {
    return (n * factorial_r(n - 1));
  }
}
```

Base Case

Recursive Step

Notice how the factorial method calls itself

How the Code Executes

Call Stack

All the calls stack up until the base case is reached

setup
`let x = factorial_r(4);`

How the Code Executes

Call Stack

All the calls stack up until the base case is reached

factorial_r(4)
`return 4 * factorial_r(3);`

setup
`let x = factorial_r(4);`

How the Code Executes

Call Stack

All the calls stack up until the base case is reached

factorial_r(3)
`return 3 * factorial_r(2);`

factorial_r(4)
`return 4 * factorial_r(3);`

setup
`let x = factorial_r(4);`

How the Code Executes

```
factorial_r(2)
return 2 * factorial_r(1);

factorial_r(3)
return 3 * factorial_r(2);

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

How the Code Executes

```
factorial_r(1)
return 1 * factorial_r(0);

factorial_r(2)
return 2 * factorial_r(1);

factorial_r(3)
return 3 * factorial_r(2);

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

How the Code Executes

```
factorial_r(0)
return 1

factorial_r(1)
return 1 * factorial_r(0);

factorial_r(2)
return 2 * factorial_r(1);

factorial_r(3)
return 3 * factorial_r(2);

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

How the Code Executes

```
factorial_r(0)
return 1

factorial_r(1)
return 1 * factorial_r(0);

factorial_r(2)
return 2 * factorial_r(1);

factorial_r(3)
return 3 * factorial_r(2);

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

Now, we return all the calls back to main

How the Code Executes

```
factorial_r(1)
return 1 * 1;

factorial_r(2)
return 2 * factorial_r(1);

factorial_r(3)
return 3 * factorial_r(2);

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

Now, we return all the calls back to main

Wentworth
Computing & Data Science

How the Code Executes

```
factorial_r(2)
return 2 * 1;

factorial_r(3)
return 3 * factorial_r(2);

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

Now, we return all the calls back to main

Wentworth
Computing & Data Science

How the Code Executes

```
factorial_r(3)
return 3 * 2;

factorial_r(4)
return 4 * factorial_r(3);

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

Now, we return all the calls back to main

Wentworth
Computing & Data Science

How the Code Executes

```
factorial_r(4)
return 4 * 6;

setup
let x = factorial_r(4);
```

Call Stack

All the calls stack up until the base case is reached

Now, we return all the calls back to main

Wentworth
Computing & Data Science

How the Code Executes

Call Stack

All the calls stack up until the base case is reached

Now, we return all the calls back to main

```
setup
let x = 24;
```


Solving via Recursion

In general, to solve a problem using recursion break it into sub-problems

If a sub-problem is similar to the original problem, just smaller in size, you can apply the same approach to solve the sub-problem recursively

Always make sure to have a base case, which is when the sub-problem has become "too small"

Topics for Today

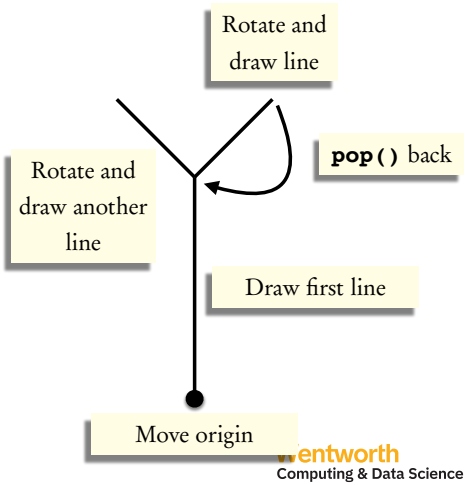
- Object Communication
- Recursion
- **Fractal Trees**

Fractal Trees: The First Branches

We're going to make a tree using recursion.

This will take a few new tools that p5.js has available:

```
translate();
rotate();
push();
pop();
```



Fractal Trees: OOP

Can we use OOP to create/ animate this process?

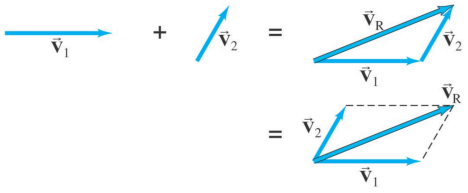
Each Branch object could store a start and end position.

To do this, we'll need to use vectors and, probably, some trig...

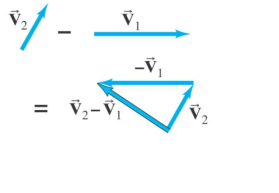
Vector Operations

- There are two main ways to think of **`Vector`** objects.
  - As points in space.
  - As "arrows" with magnitude and direction.

Addition Picture



Subtraction Picture



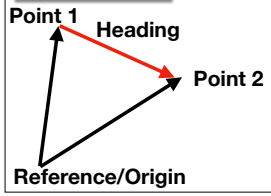
Vector Operations

Direction and Distance

Common problem:

How far is one point from another point and in what direction?

Heading



Distance

Direction