

# CIS 1068

## Little Bit of Recursion

```
2     public static boolean isEven(int x) {
3         return x%2==0;
4     }
5
6     public static int triple(int x) {
7         return x*3;
8     }
9
10    public static int someFunc(int x) {
11        if (isEven(x)) {
12            return x+1;
13        } else {
14            return triple(x);
15        }
16    }
17
18    public static void main(String args[]) {
19        int x=5;
20        System.out.println(someFunc(x));
21    }
```

```
2 public static boolean isEven(int x) {
3     return x%2==0;
4 }
5
6 public static int triple(int x) {
7     return x*3;
8 }
9
10 public static int someFunc(int x) {
11     if (isEven(x)) {
12         return x+1;
13     } else {
14         return triple(x);
15     }
16 }
17
18 public static void main(String args[]) {
19     int x=5;
20     System.out.println(someFunc(x));
21 }
```

so many x's



Why is this ok?

Why is this ok? different scope.  
different vars, same name

## What Happens Here?

```
2  public static int f(int x) {
3      if (x==1) {
4          return 1;
5      } else {
6          return 2*f(x-1);
7      }
8  }
9  public static void main(String args[]) {
10     int x = 3;
11     System.out.println(f(x));
12 }
```

# What Happens Here?

```
2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
9 public static void main(String args[]) {
10     int x = 3;
11     System.out.println(f(x));
12 }
```

Is this even legal?

- ▶ **Question:** Is it ok for a function to call itself?
- ▶ **Answer:** Yes. We call it **recursion**

# How Do We Handle This?

It's no different from any other function call you've ever made:

- ▶ copy the arguments to the function
- ▶ execute the function
- ▶ jump back



## It's almost as though we did this

(but don't actually recopy the function. This would be an error)

```
public static int f(int x) {
    if (x==1) {
        return 1;
    } else {
        return 2*f(x-1);
    }
}
```

```
public static int f(int x) {
    if (x==1) {
        return 1;
    } else {
        return 2*f(x-1);
    }
}
```

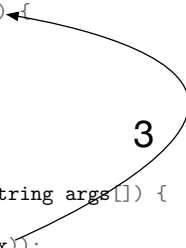
```
public static int f(int x) {
    if (x==1) {
        return 1;
    } else {
        return 2*f(x-1);
    }
}
```

```
public static void main(String args[]) {
    int x = 3;
    System.out.println(f(x));
}
```

```
2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
```

```
2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
```

```
2 public static int f(int x) ← {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
9 public static void main(String args[]) {
10     int x = 3;
11     System.out.println(f(x));
12 }
```



```
2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
```

```
2 public static int f(int x) ← {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
```

```
2 public static int f(int x) ← {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }
```

```
9 public static void main(String args[]) {
10     int x = 3;
11     System.out.println(f(x));
12 }
```

2

3

```
2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }

2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }

2 public static int f(int x) {
3     if (x==1) {
4         return 1;
5     } else {
6         return 2*f(x-1);
7     }
8 }

9 public static void main(String args[]) {
10     int x = 3;
11     System.out.println(f(x));
12 }
```

The diagram illustrates three recursive calls to the function `f(x)`. Each call is represented by a copy of the function definition. Arrows point from the function signature `f(int x)` in each copy to the corresponding label: 1 for the top call, 2 for the middle call, and 3 for the bottom call. The bottom call is part of a `main` method that sets `x = 3` and prints the result of `f(x)`.

prints 4

## Another One

```
1  public class SimpleRecur2 {
2      public static int f(int x) {
3          if (x==1) {
4              return 2;
5          } else {
6              return 2*f(x-1)+1;
7          }
8      }
9      public static void main(String args[]) {
10         int x = 5;
11         System.out.println(f(x));
12     }
13 }
```