# Recursive Tracing

# Factorial

- $n!$
- n factorial= 1*2*3*4*5*….*n
- 4! = 4 factorial = 1*2*3*4=24
- 5! = 1*2*3*4*5 = 5 * 4!=120
- 1! = 1
- 0!=1
- -1 factorial doesn’t make sense

# Iterative Factorial

```
public static int iterativeFactorial(int n){
        int answer = 1;
        for(int i = 2; i <= n; i++){
            answer *= i;
        }
        return answer;
    }
```

```
iterativeFactorial(4)
answer=1*2*3*4
```

# Recursive Factorial

```
public static int recursiveFactorial(int 4 ){
```

```
public static int recursiveFactorial(int 3 ){
```

```
public static int recursiveFactorial(int 2 ){
```

```
public static int recursiveFactorial(int 1 ){
        if(n == 1){
            return 1;
        }
        else{
            return n * recursiveFactorial(n-1);
        }
    }
```

```
recursiveFactorial(4)
    return 4*recursiveFactorial(3)
                return 3*recursiveFactorial(2)
                            return 2*recursiveFactorial(1)
                                        return 1
```

# Recursive Factorial

```
public static int recursiveFactorial(int 4 ){
public static int recursiveFactorial(int 3 ){
public static int recursiveFactorial(int 2 ){
public static int recursiveFactorial(int 1 ){
        if(n == 1){
            return 1;
        }
        else{
            return n * recursiveFactorial(n-1);
        }
    }
```

Base Case!

```
recursiveFactorial(4)
    return 4*recursiveFactorial(3)
                return 3*recursiveFactorial(2)
                            return 2*recursiveFactorial(1)
                                        return 1
```

# Recursive Factorial

4 * 3 * 2 * 1

```
public static int recursiveFactorial(int 4 ){
```

```
public static int recursiveFactorial(int 3 ){
```

```
public static int recursiveFactorial(int 2 ){
```

```
public static int recursiveFactorial(int 1 ){
        if(n == 1){
            return 1;
        }
        else{
            return n * recursiveFactorial(n-1);
        }
    }
```

24

return 4* 6

return 3* 2

return 2* 1

return 1

# Recursive Mystery 1

- mystery1(0)
  - 0
  - Base case
- mystery1(1)
  - 1
  - Base case
- mystery1(2)
  - 1, 2
- mystery1(3)
  - 1, 3
- mystery1(4)
  - 1, 2, 4

```java
public static void mystery1(int n){
        if (n <= 1) {
            System.out.print(n);
        } else {
            mystery1(n / 2);
            System.out.print(", " + n);
        }
    }
```

```java
public static void mystery1(int n){
        if (n <= 1) {
            System.out.print(n);
        } else {
            mystery1(n / 2);
            System.out.print(", " + n);
        }
    }
```

# Recursive Mystery 2

```java
public static void mystery2(int n){
        if (n <= 0) {
            System.out.print("*");
        } else if (n % 2 == 0) {
            System.out.print("(");
            mystery2(n - 1);
            System.out.print(")");
        } else {
            System.out.print("[");
            mystery2(n - 1);
            System.out.print("]");
        }
    }
```

- mystery2(0)
  - *
- mystery2(1)
  - [*]
- mystery2(2)
  - ([*])
- mystery2(3)
  - [([*])]
- mystery2(4)
  - ([([*])])
- Alternating balance () and [] with a * in the middle

```java
public static void mystery2(int n){
        if (n <= 0) {
            System.out.print("*");
        } else if (n % 2 == 0) {
            System.out.print("(");
            mystery2(n - 1);
            System.out.print(")");
        } else {
            System.out.print("[");
            mystery2(n - 1);
            System.out.print("]");
        }
    }
```

# Recursive Mystery 3

- Mystery3(“taco”)

```
public static void mystery3(String str){
        if(!str.isEmpty()){
            System.out.print(str.charAt(0));
            mystery3(str.substring(1));
            System.out.print(str.charAt(0));
        }
    }
```