Physics Simulation

An Interesting Website

www.boxcar2d.com

Scope

- Each variable has a **scope** where it can be accessed and how long it lives.

```
for (int i = 0; i < 5; i++) {
    int y = i * 4;
}
i = 3; // Error!
y = 2; // Error!
```

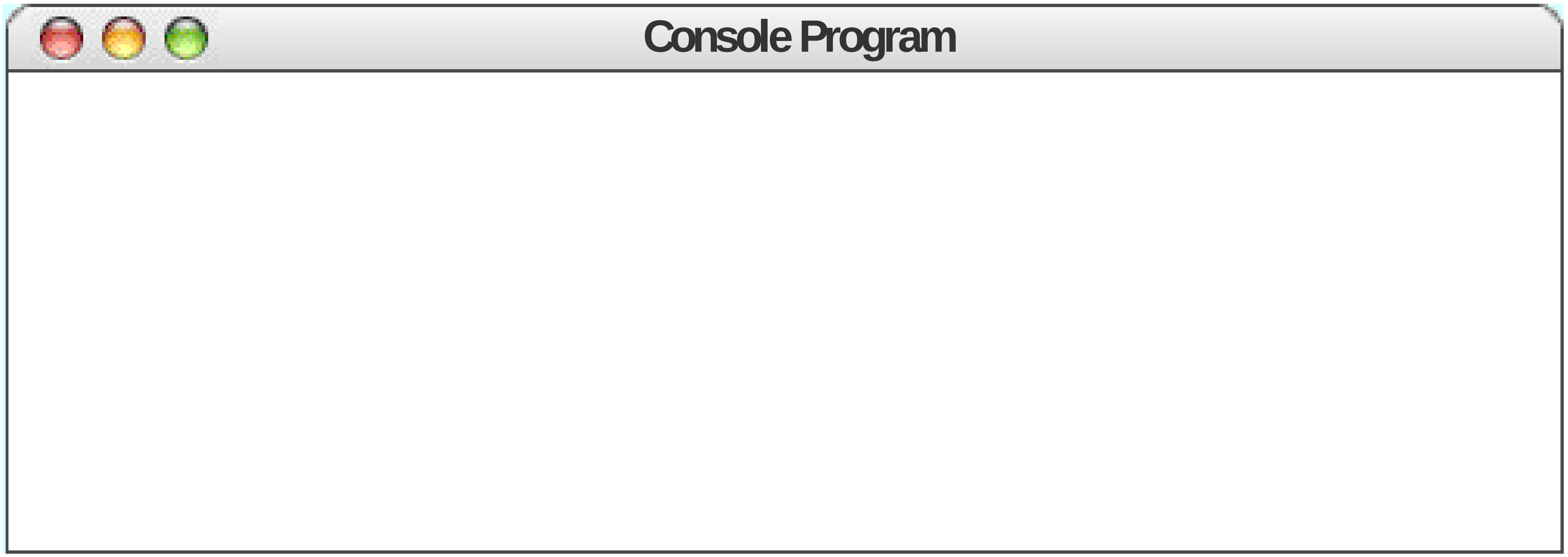
Scope of Method Calls

- A variable declared inside a method is called a **local variable**.
- Local variables can only be accessed inside of the method that declares them.

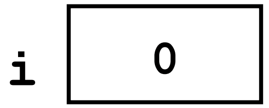
```
public void run() {
    int x = 5;
    someOtherMethod();
}
private void someOtherMethod() {
    x = 4; // Error!
}
```

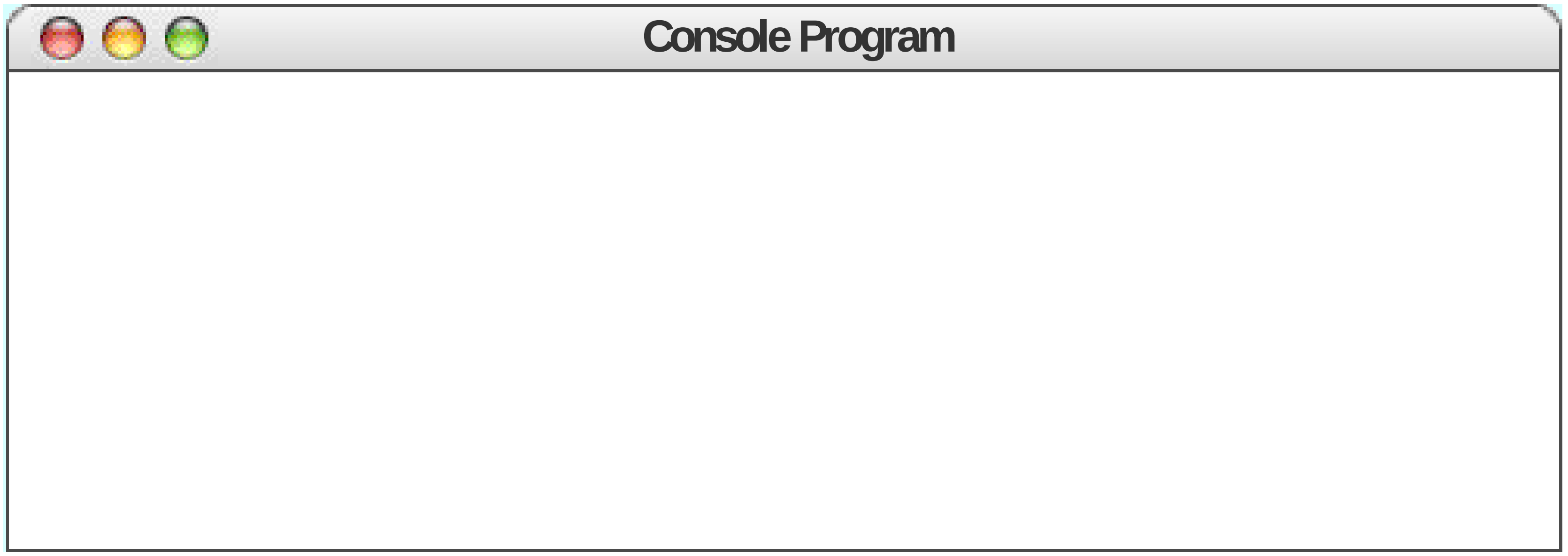
```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

i



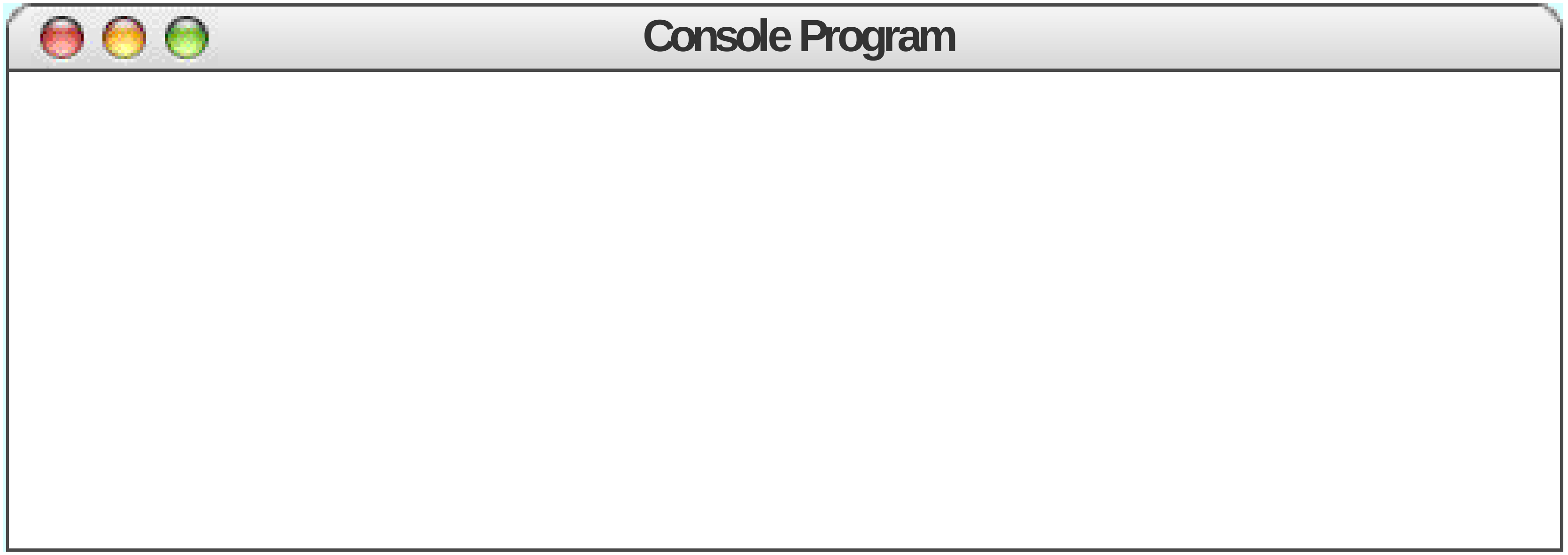
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```





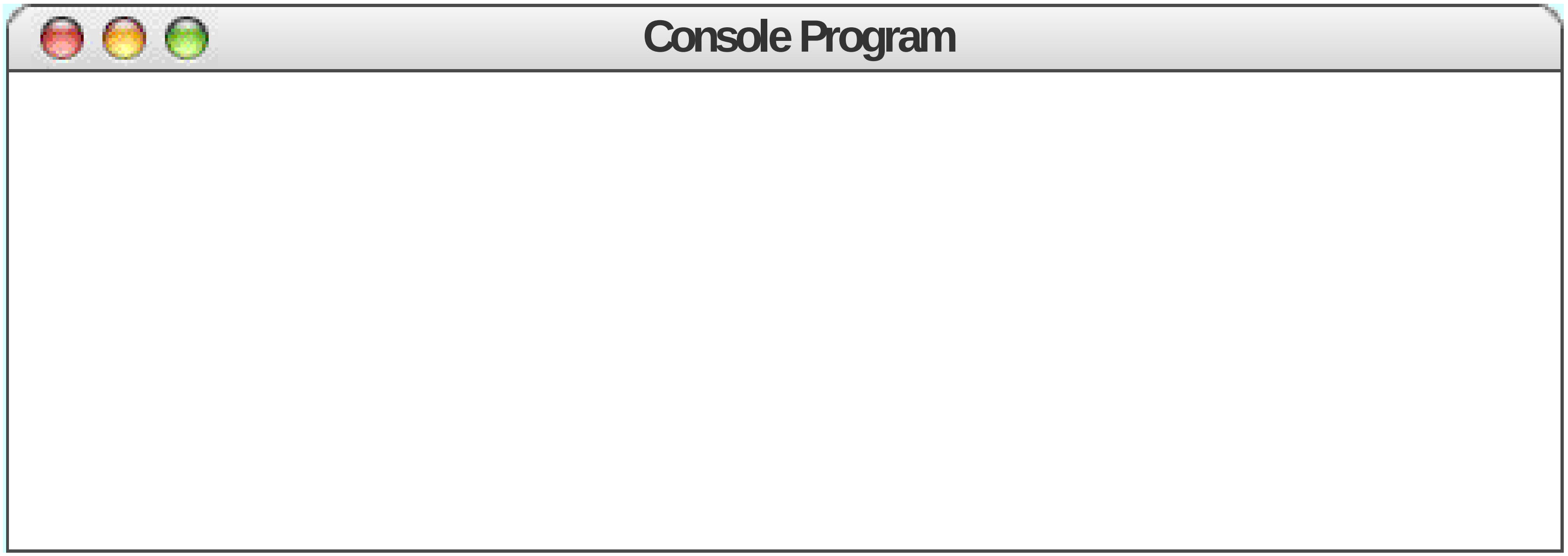
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i $\boxed{0}$



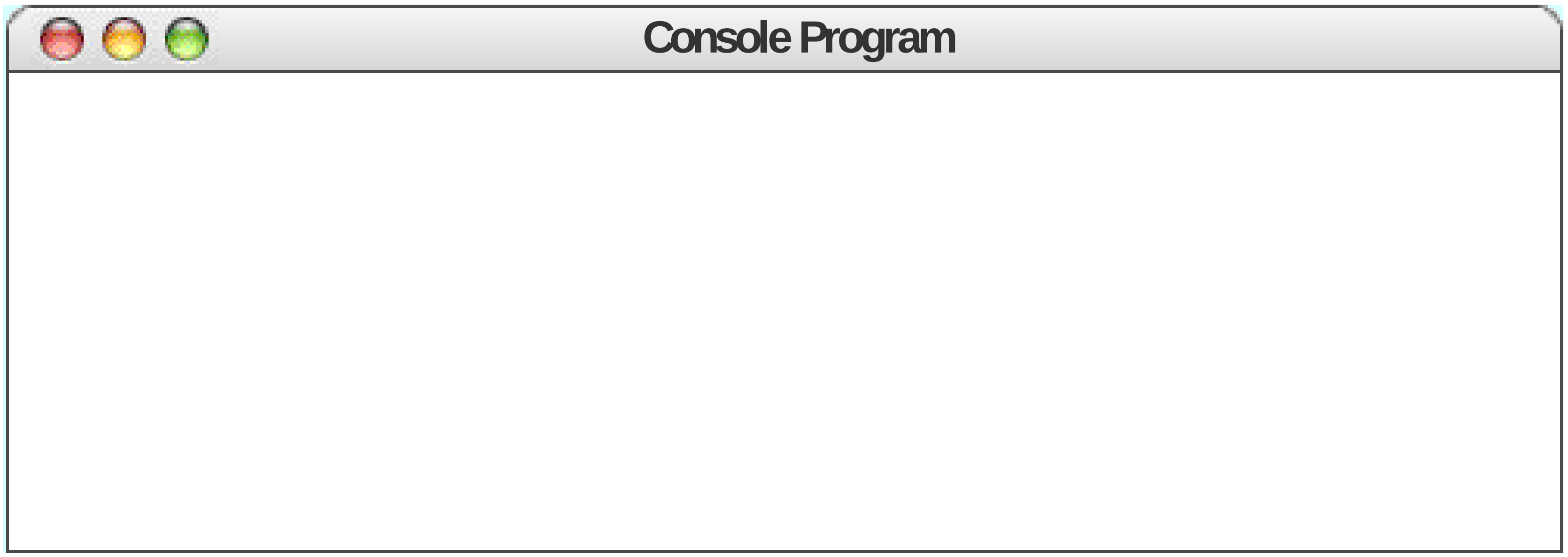
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

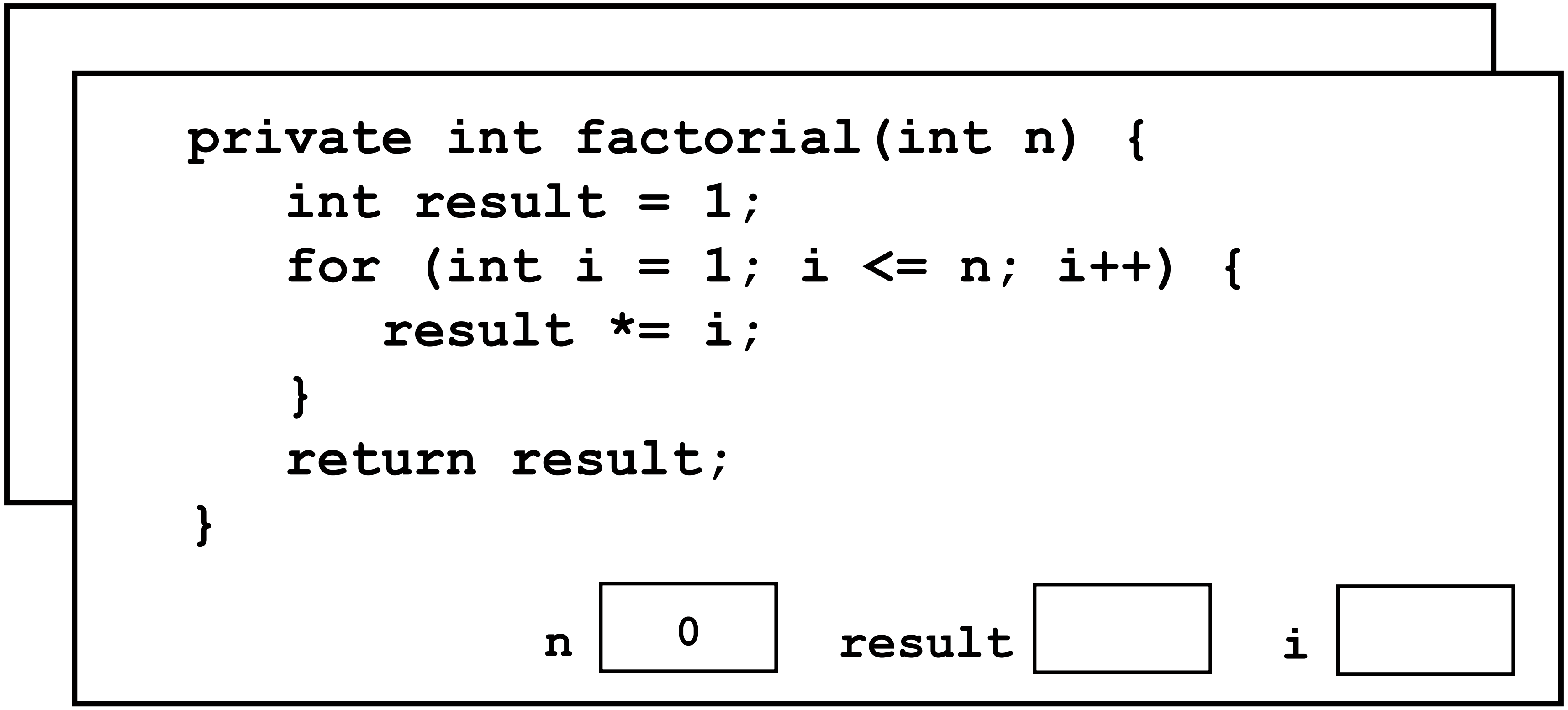
i $\boxed{0}$



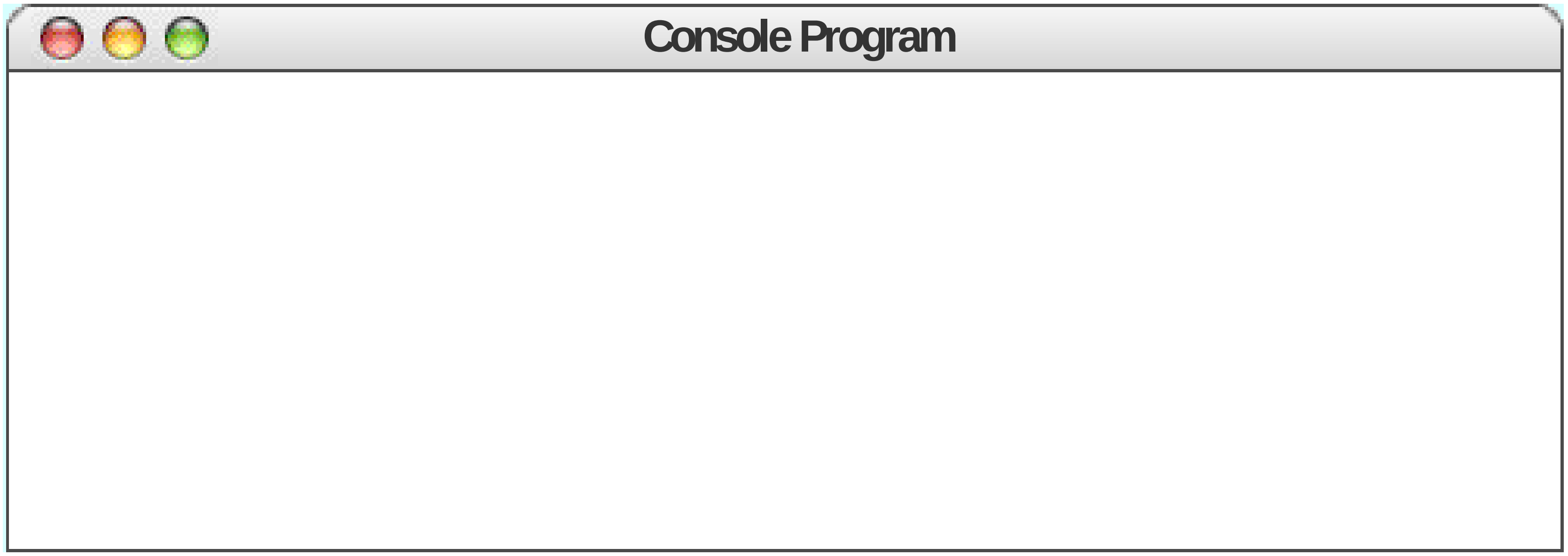
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

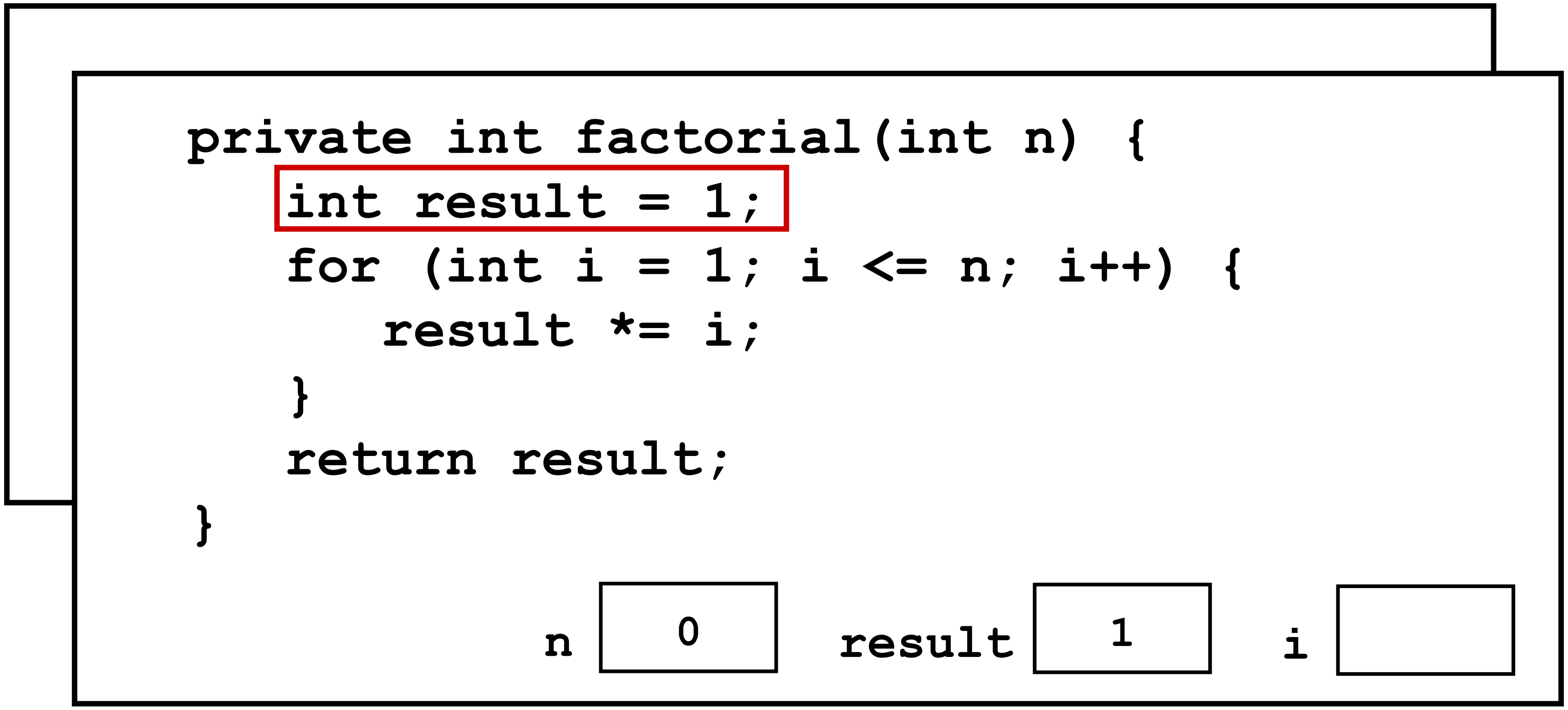
```
i  0
```



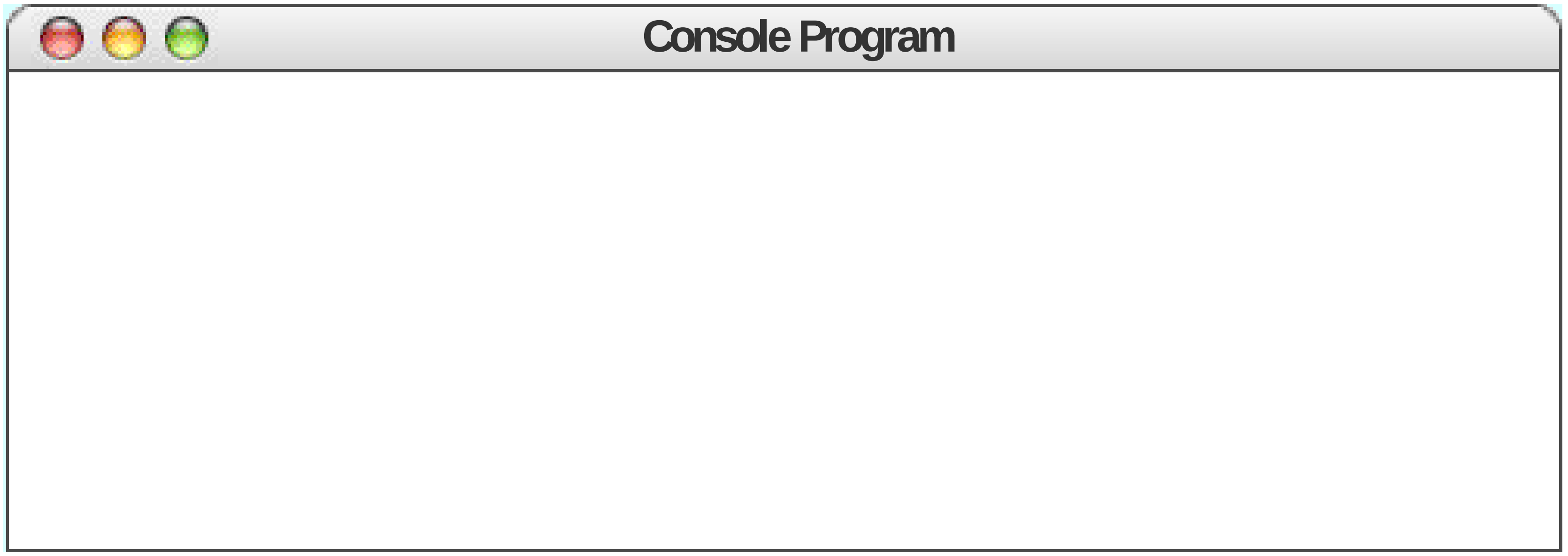


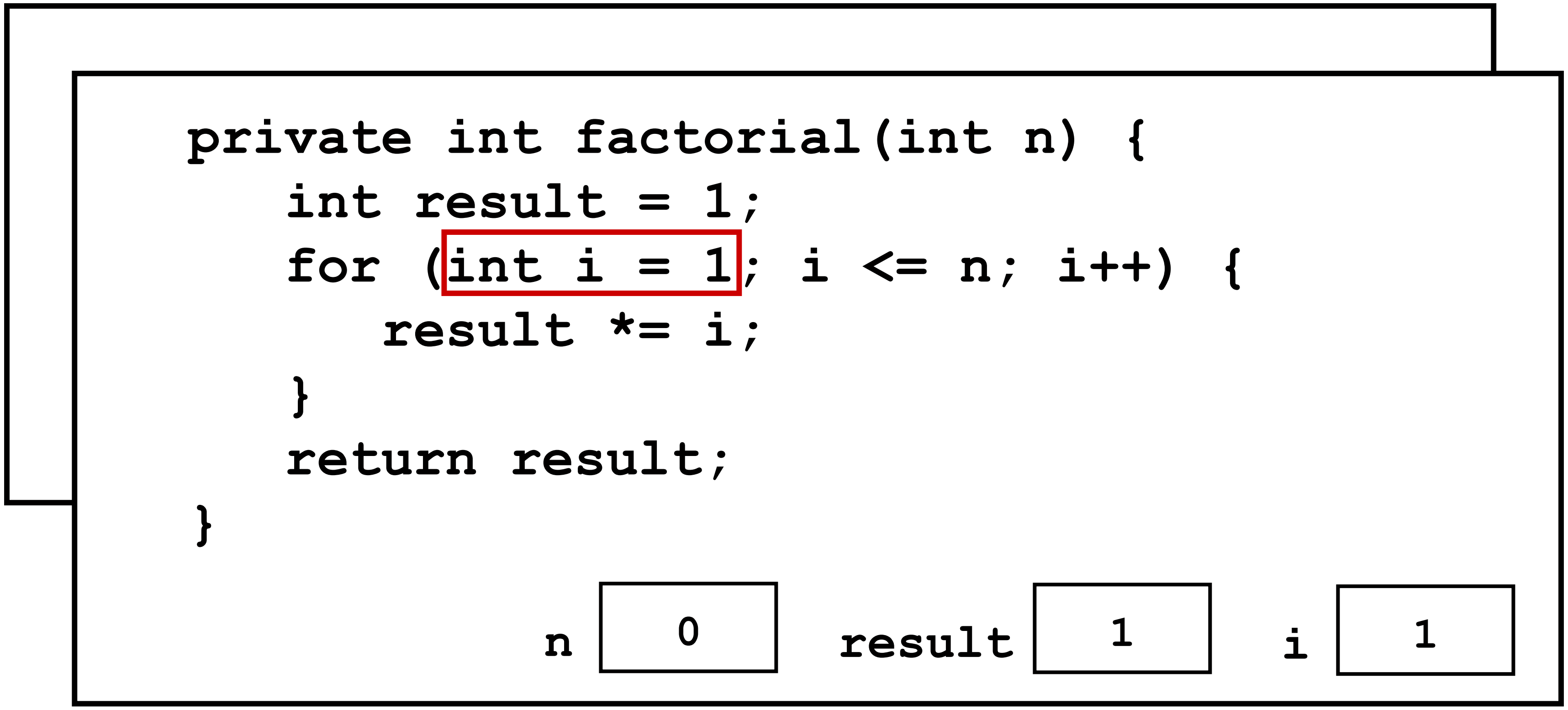
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



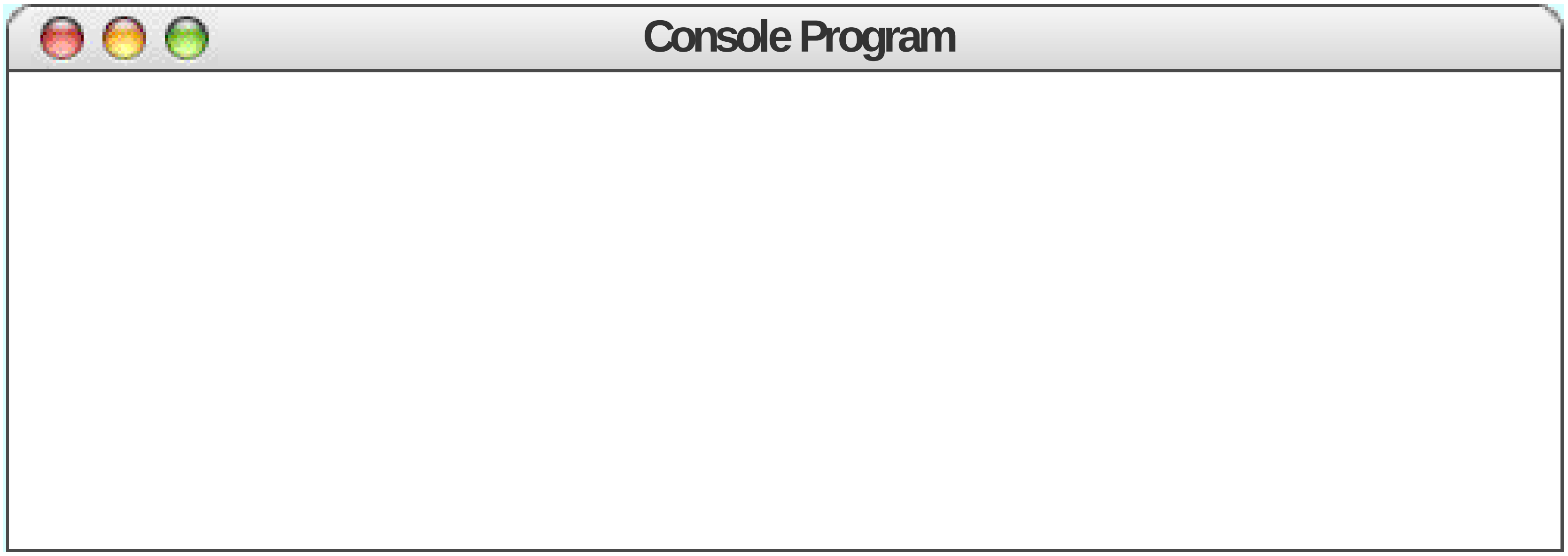


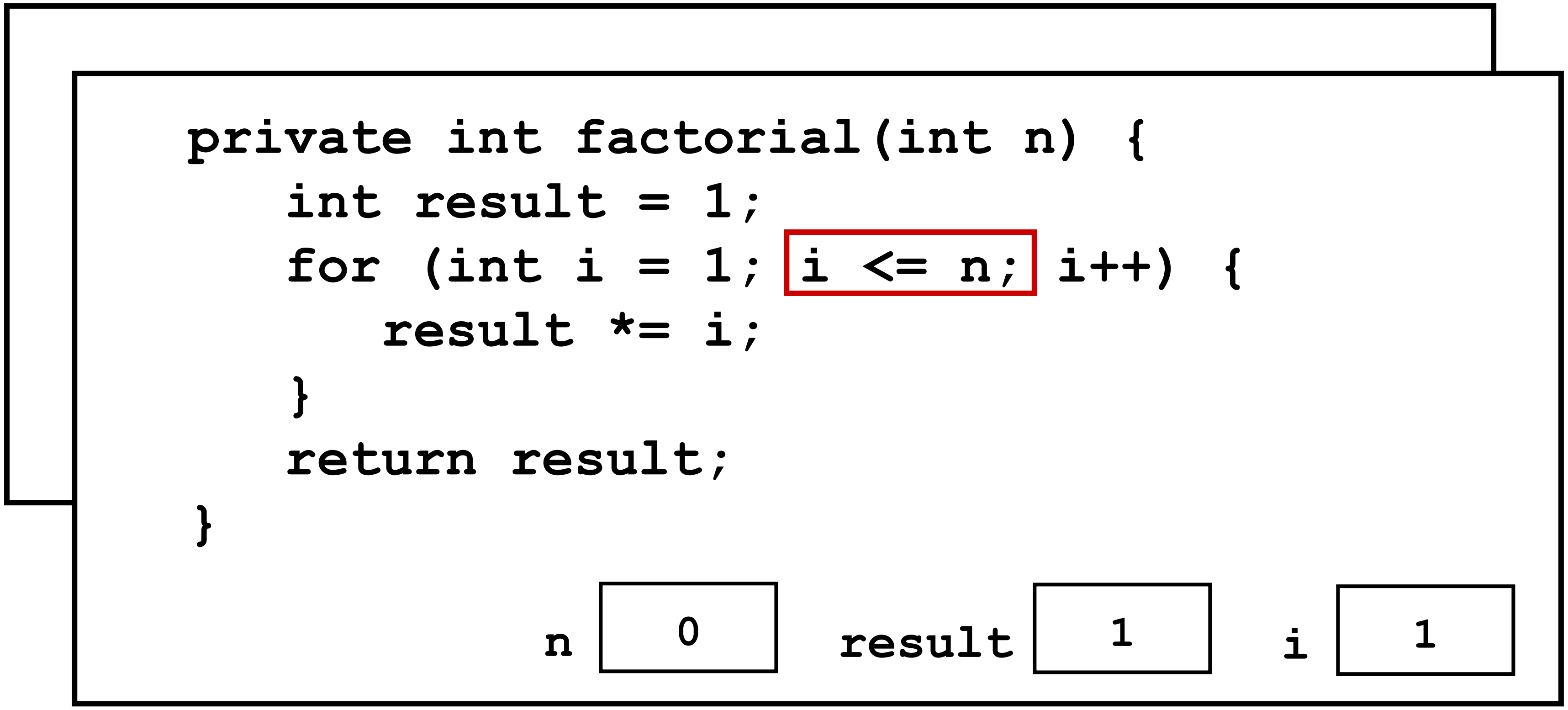
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



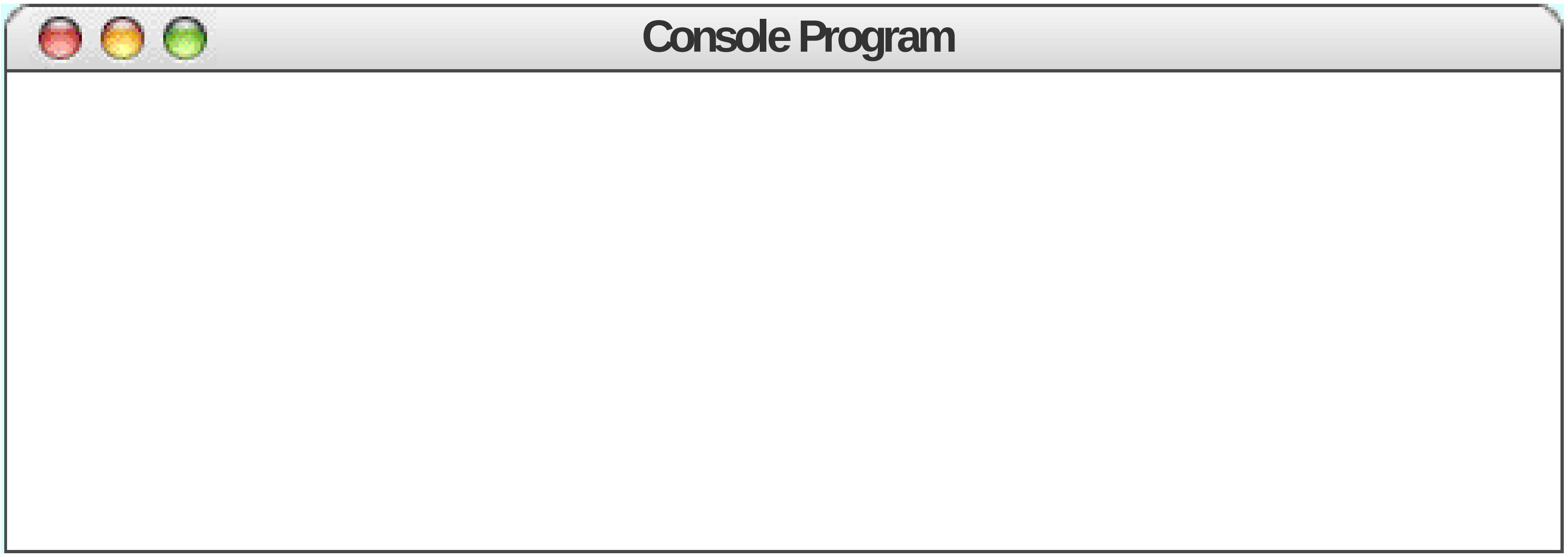


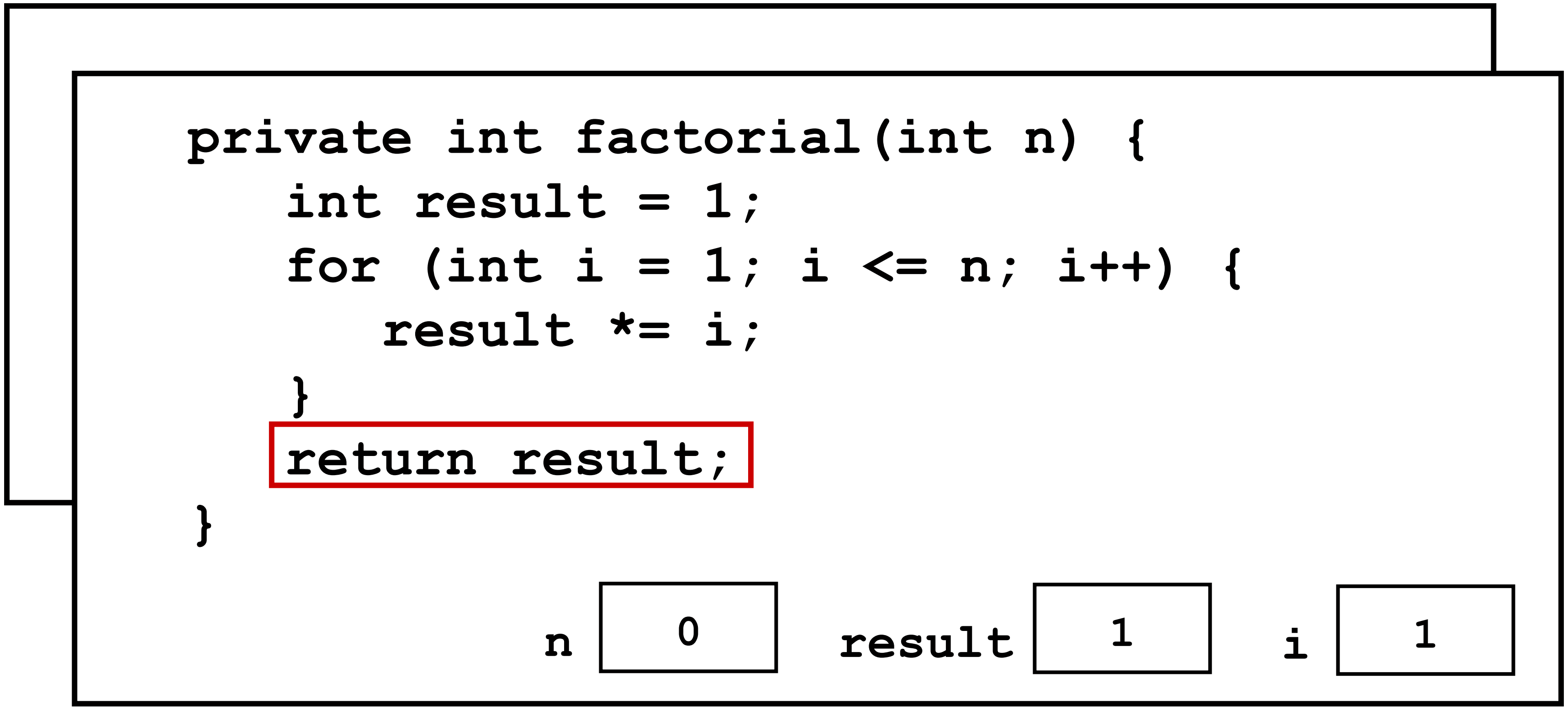
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



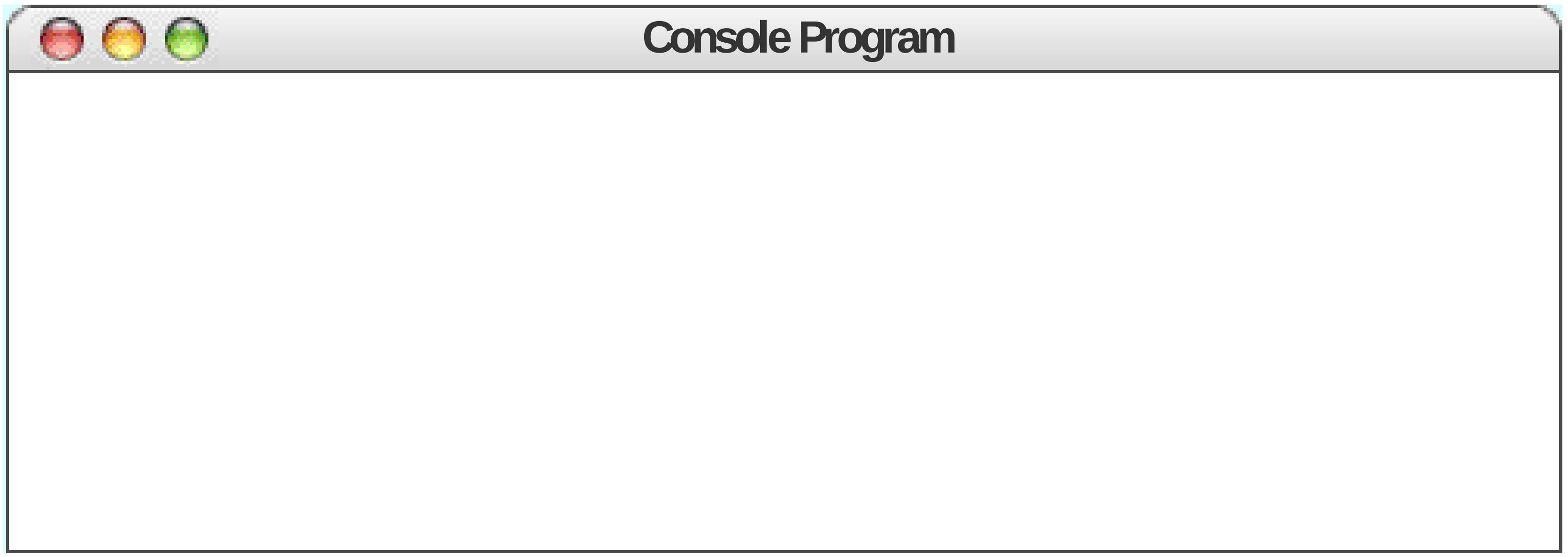


```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```





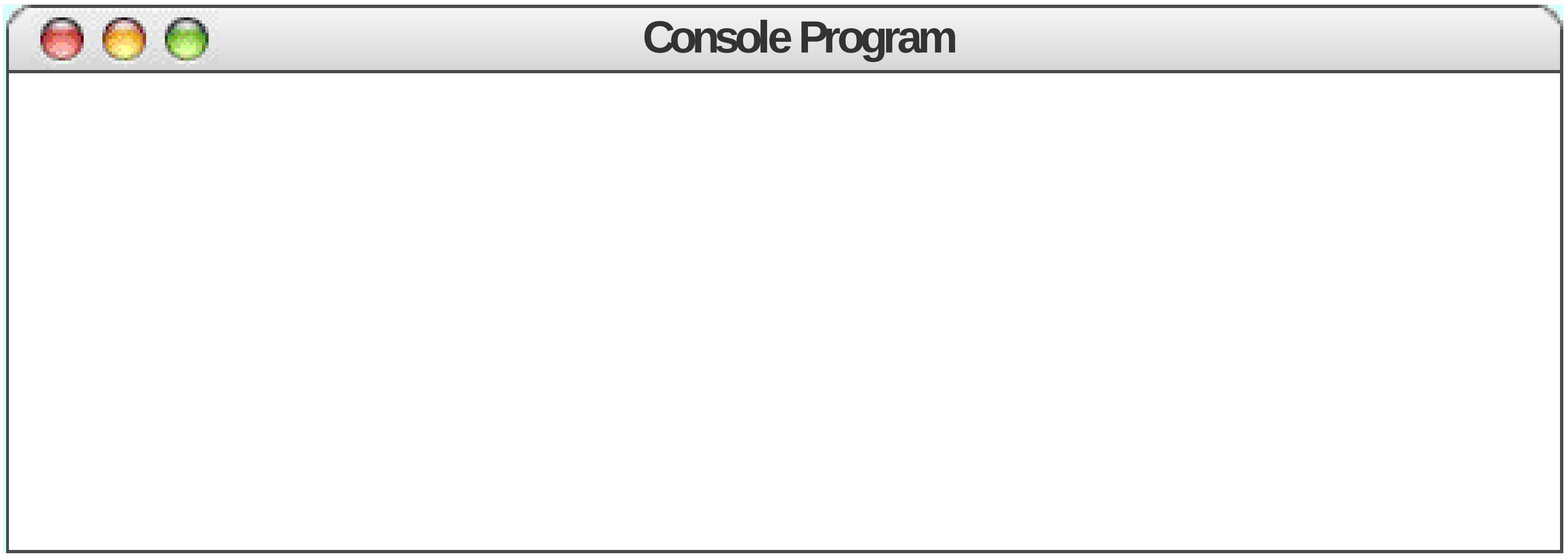
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

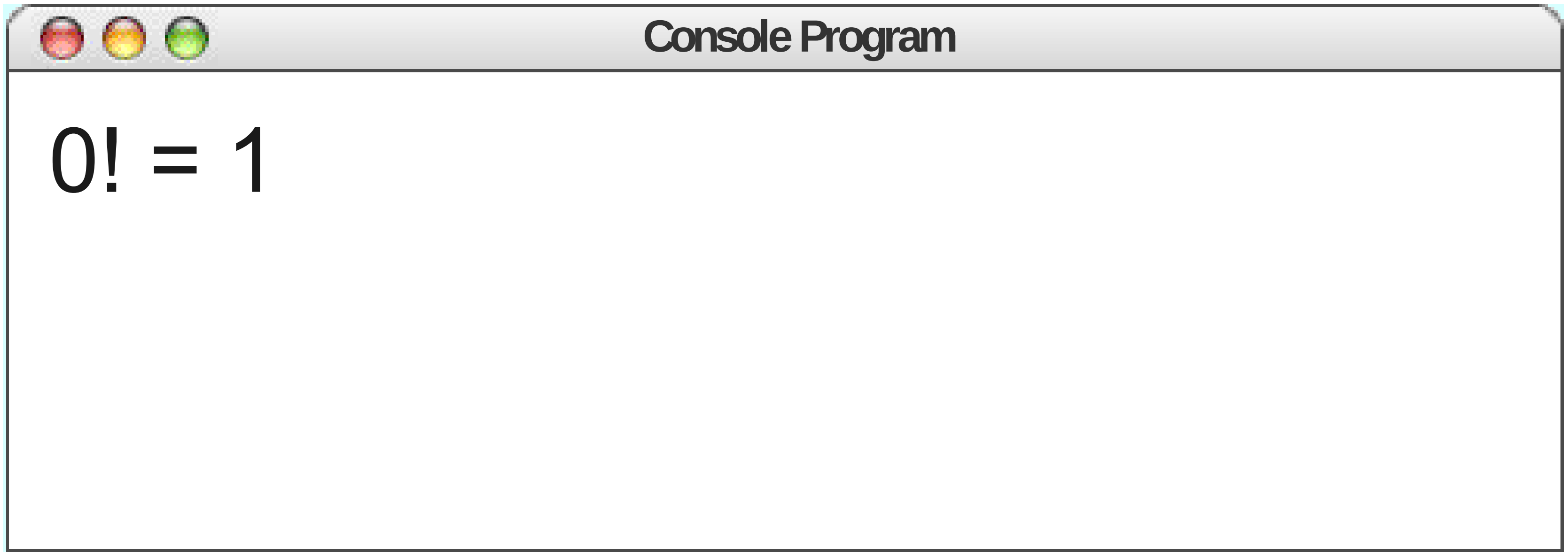
i $\boxed{0}$



```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

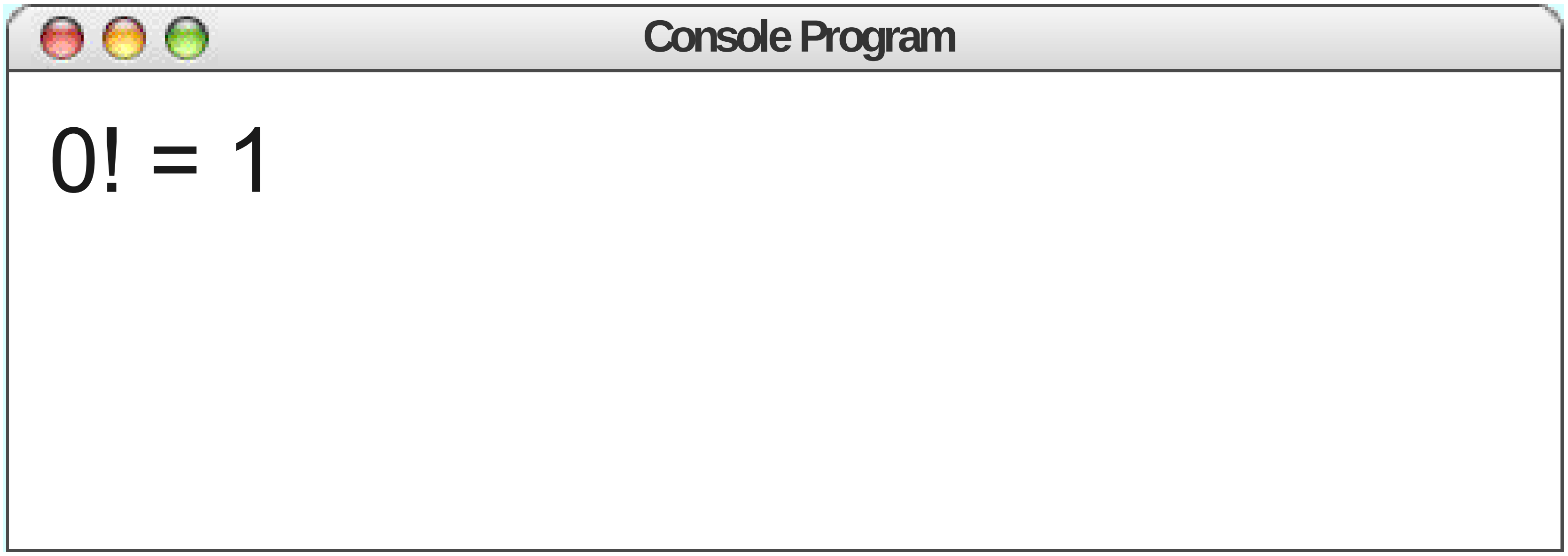
i $\boxed{0}$



```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

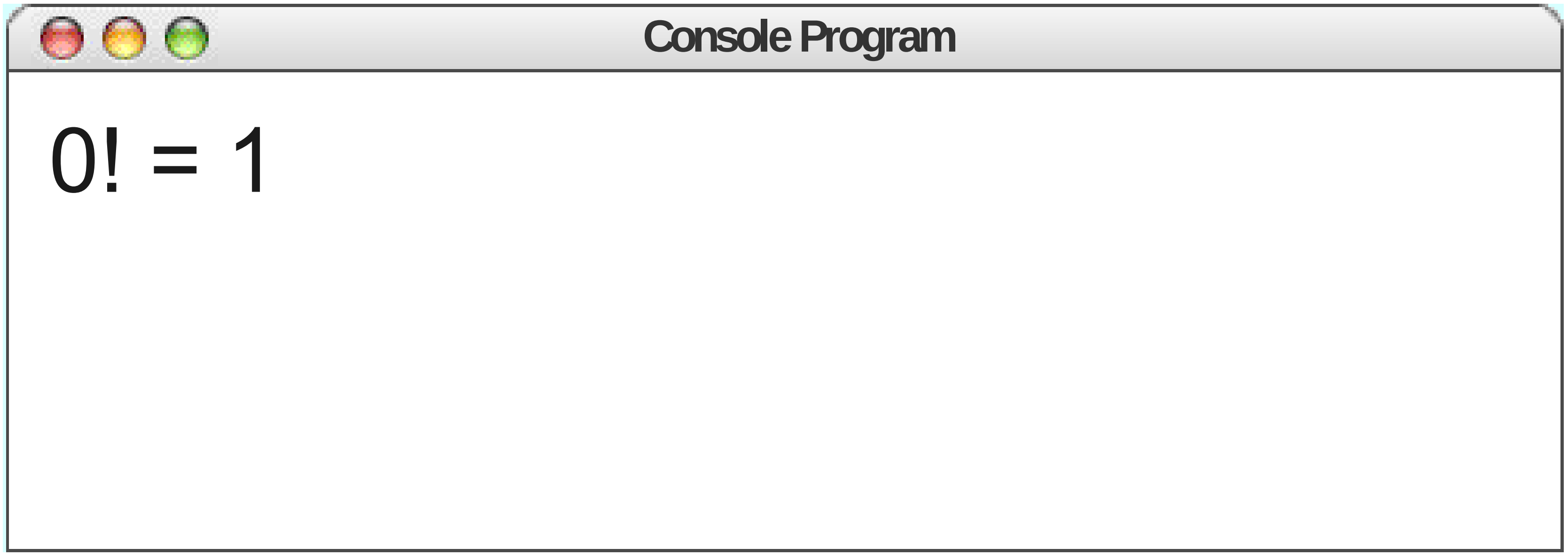
i $\boxed{1}$



```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

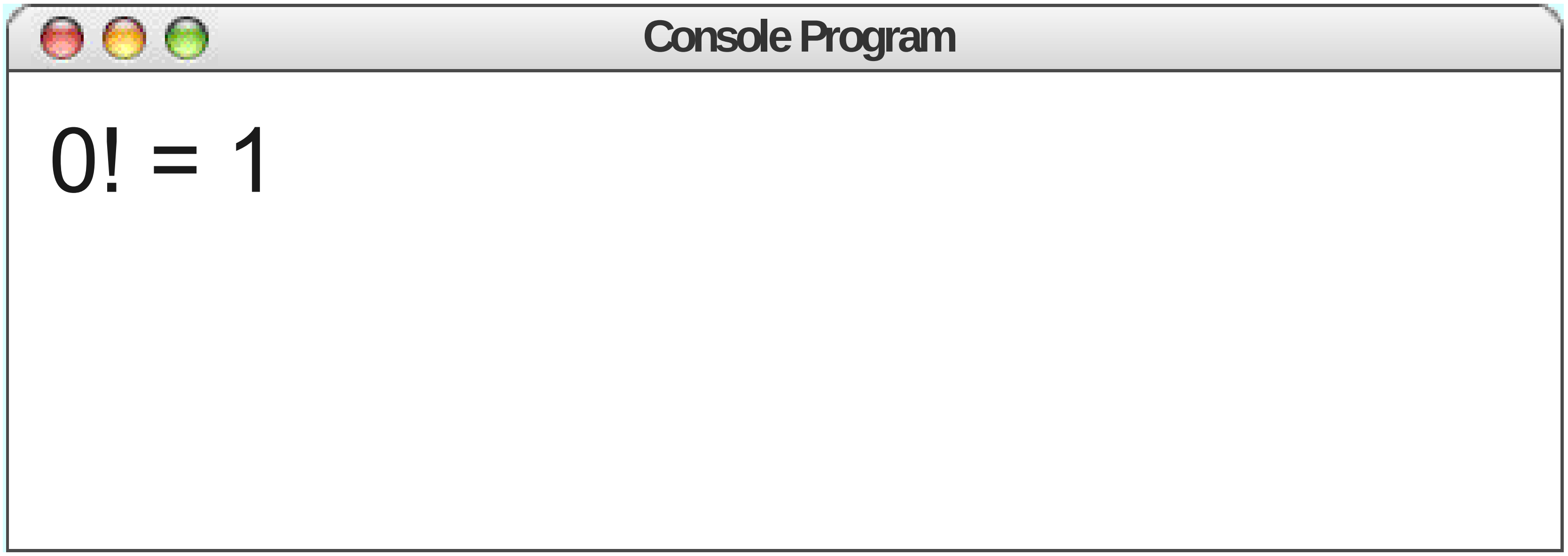
i $\boxed{1}$



```
0! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

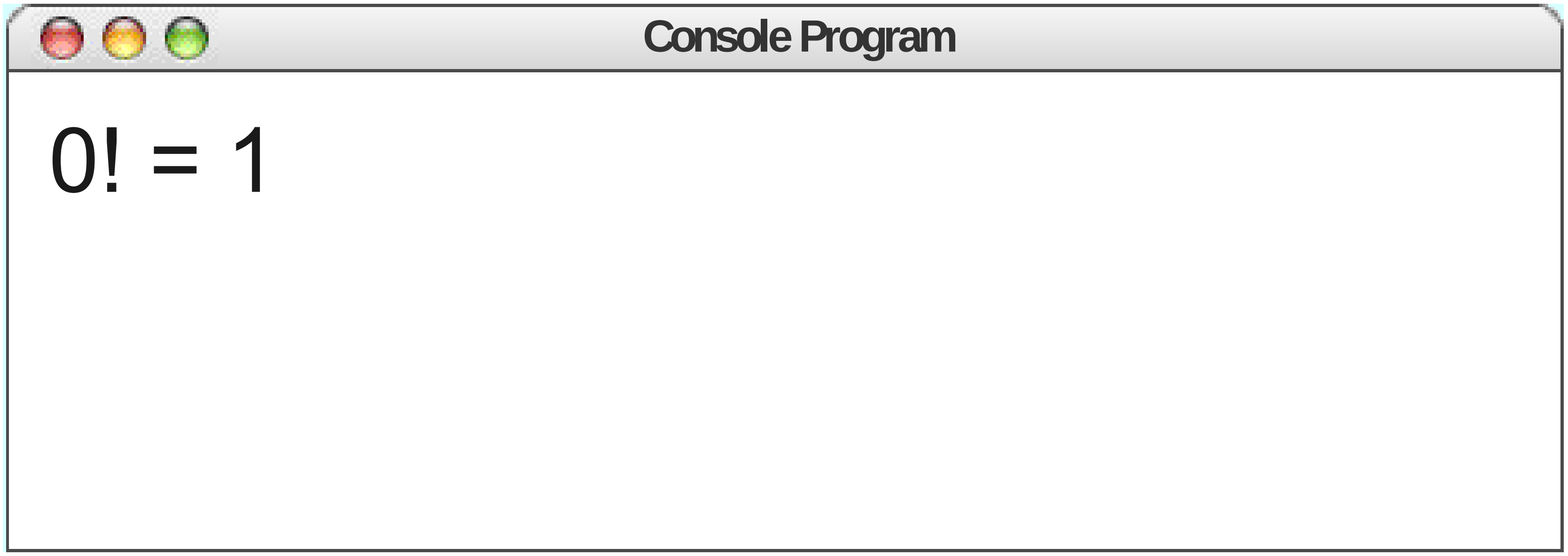
i $\boxed{1}$



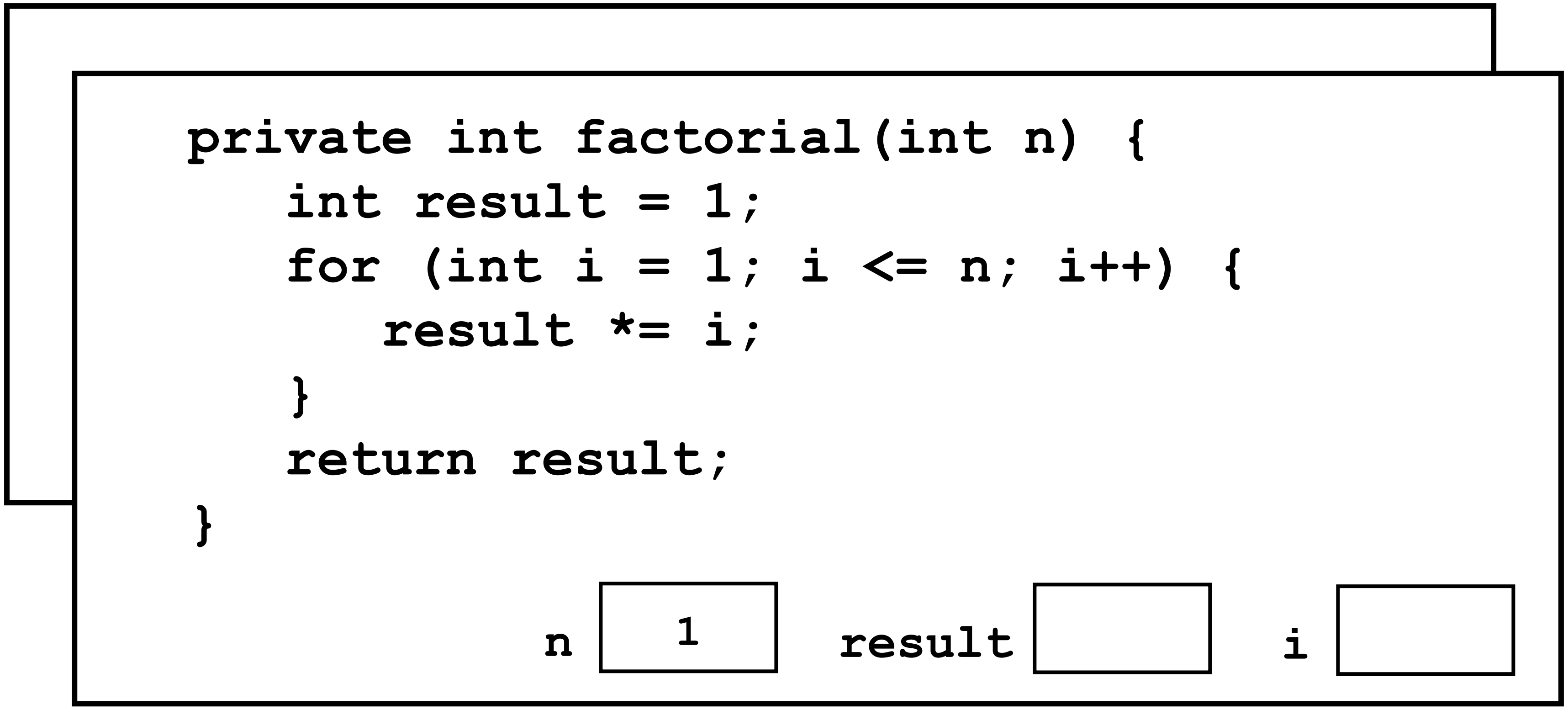
```
0! = 1
```

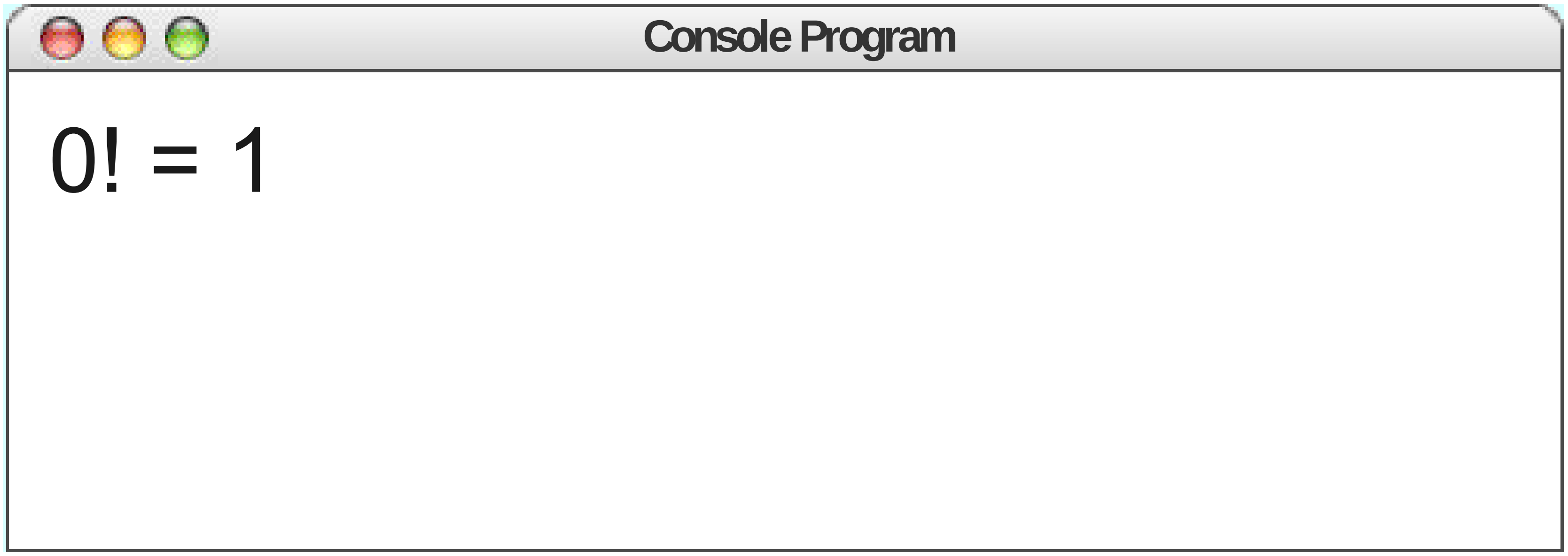
```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + $\boxed{factorial(i)}$);
   }
}
```

```
i  1
```

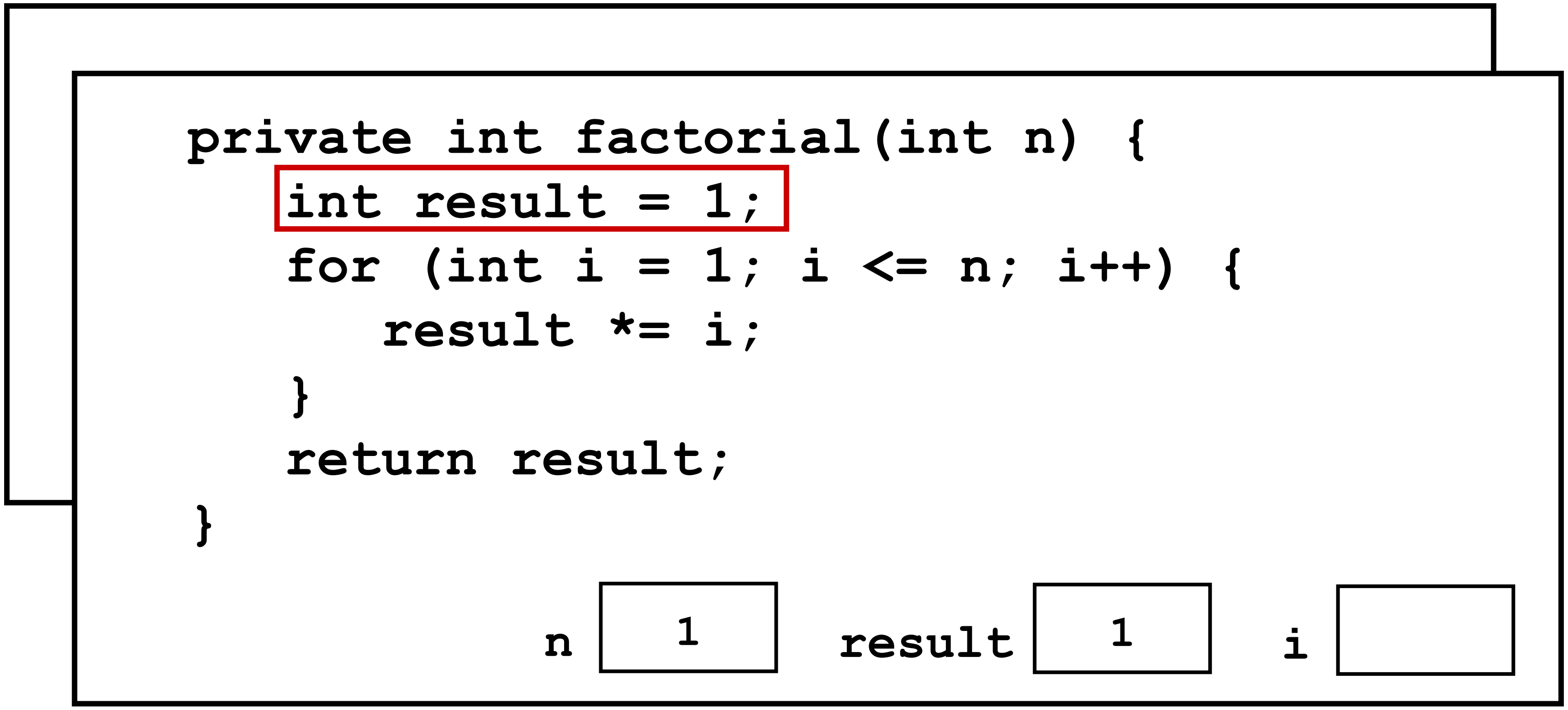


```
0! = 1
```

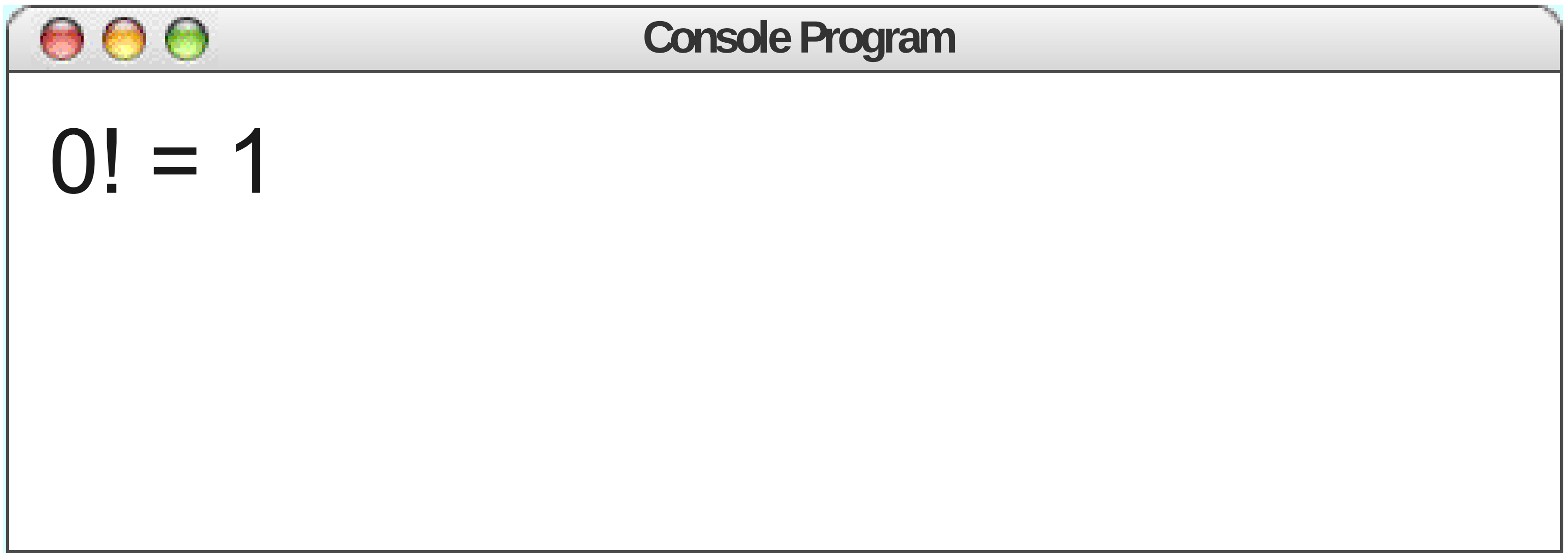

```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



```
0! = 1
```

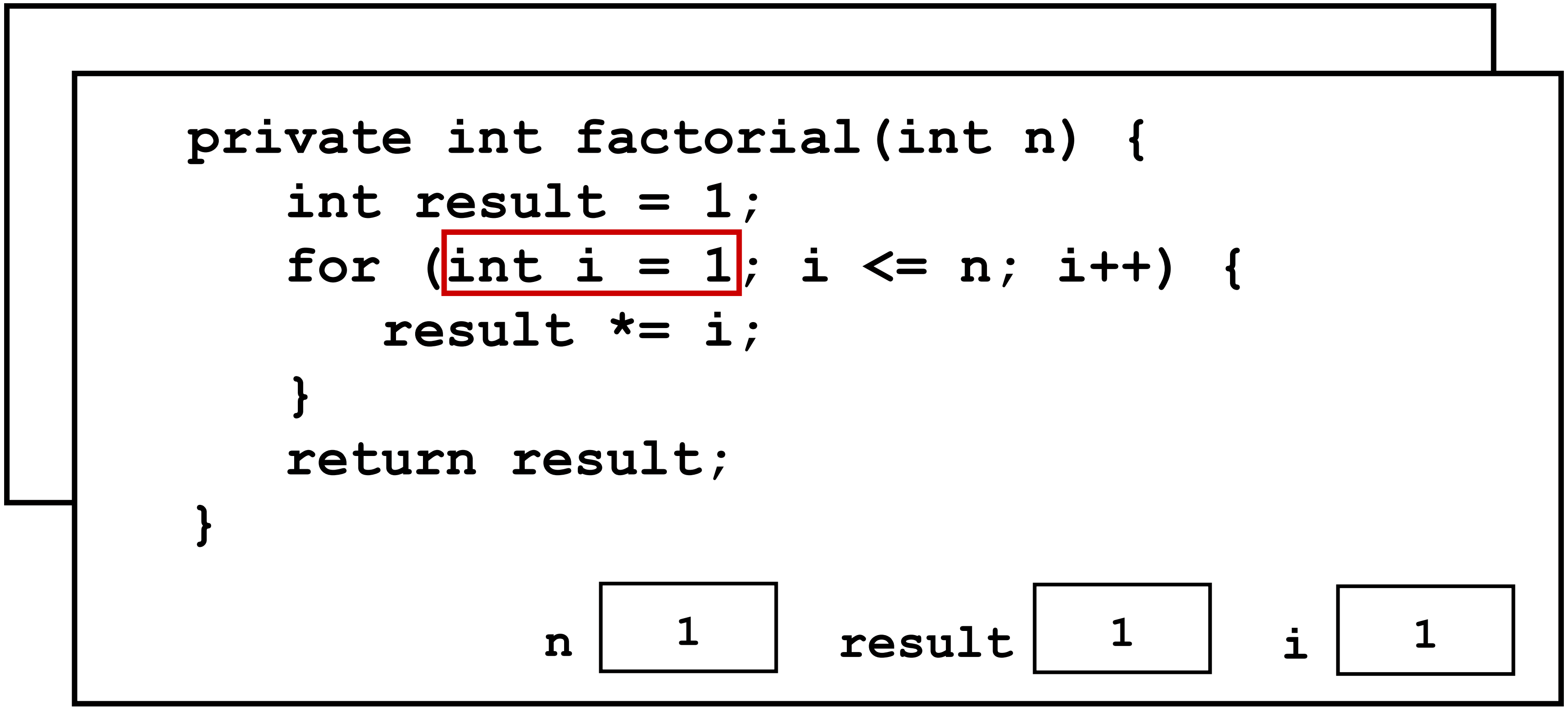


```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



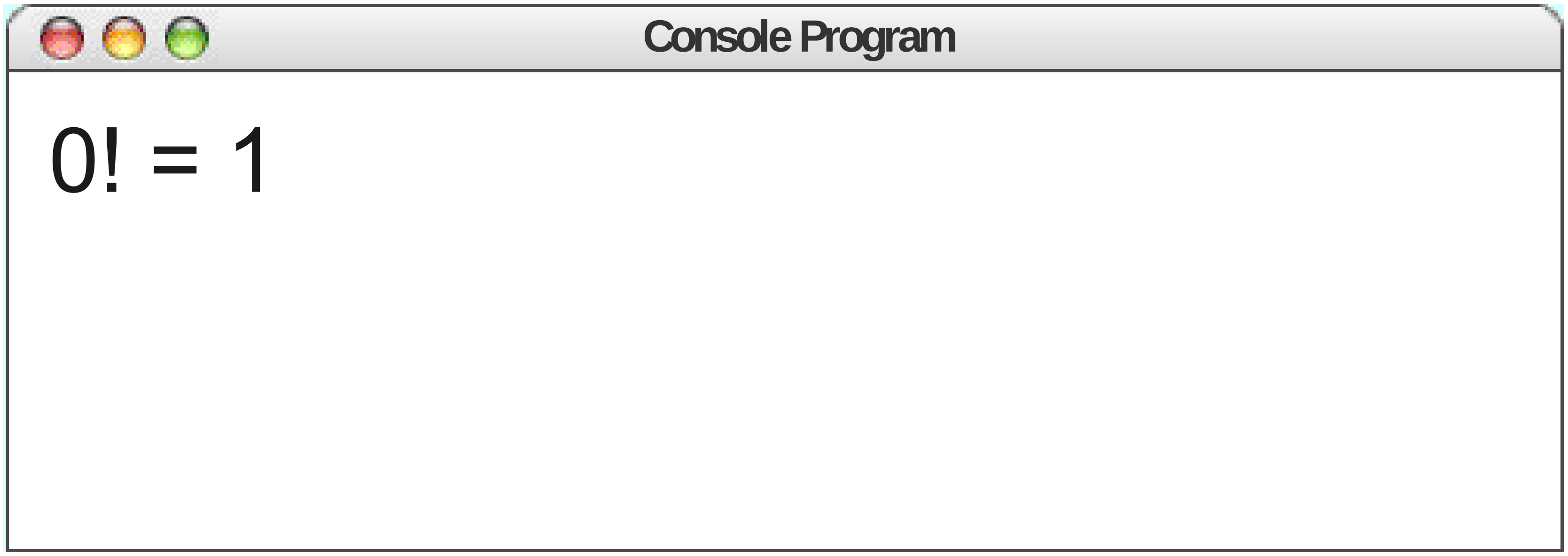
Console Program

```
0! = 1
```

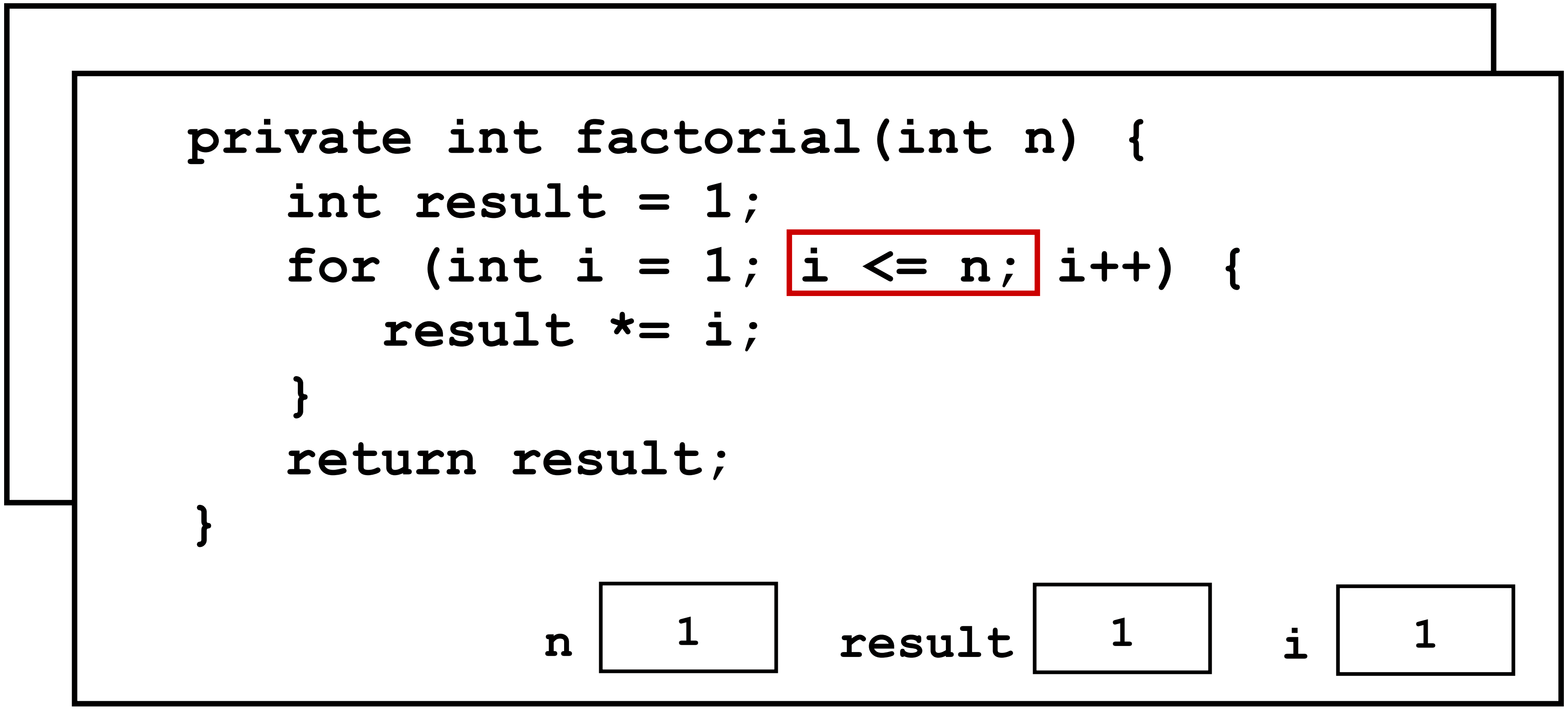


```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```

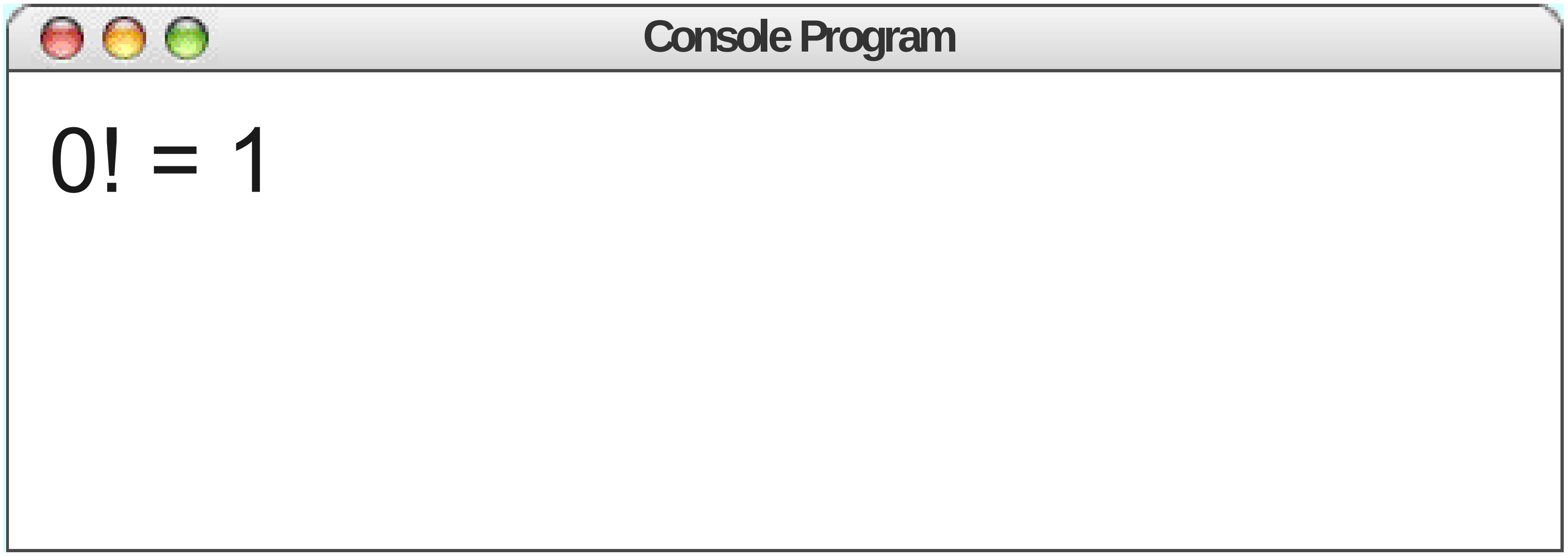
n 1   result 1   i 1



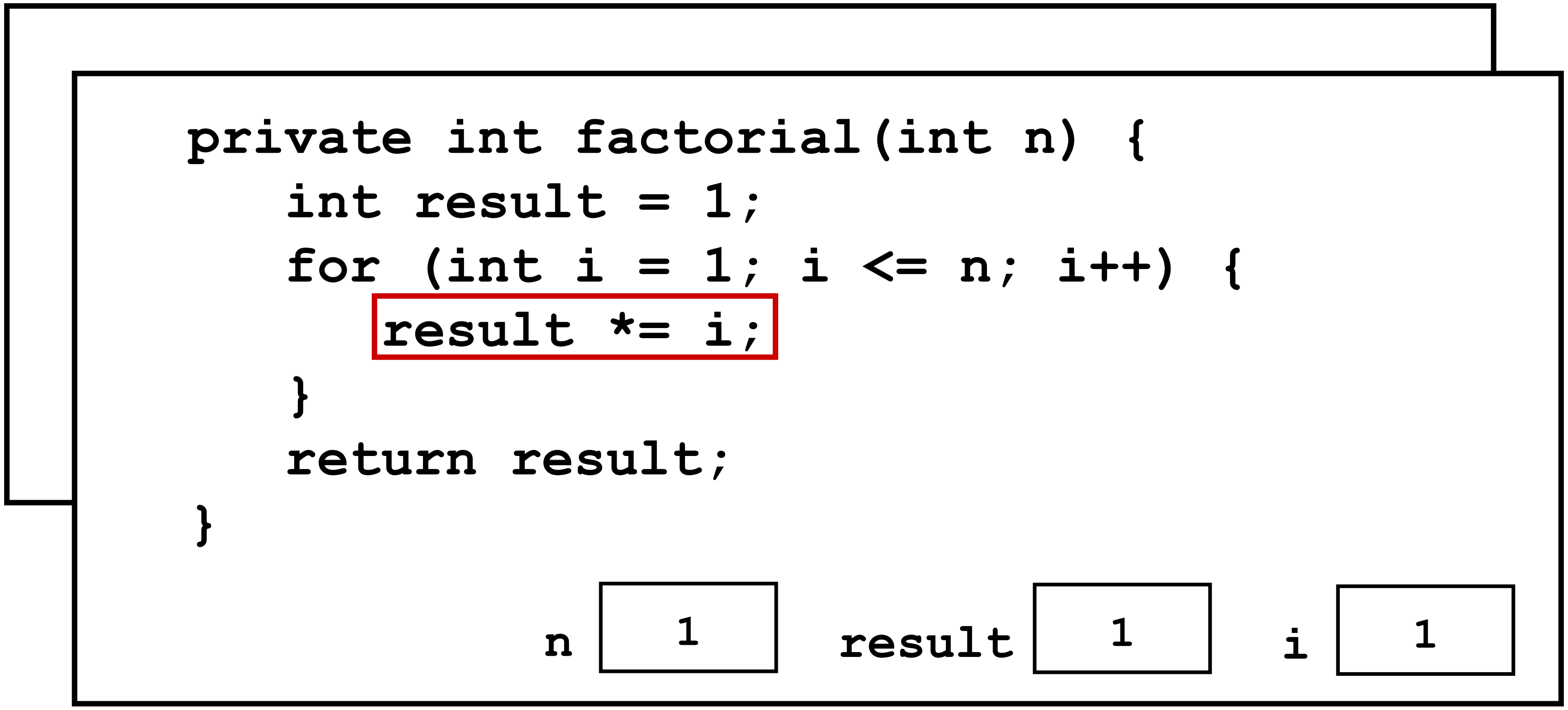
```
0! = 1
```



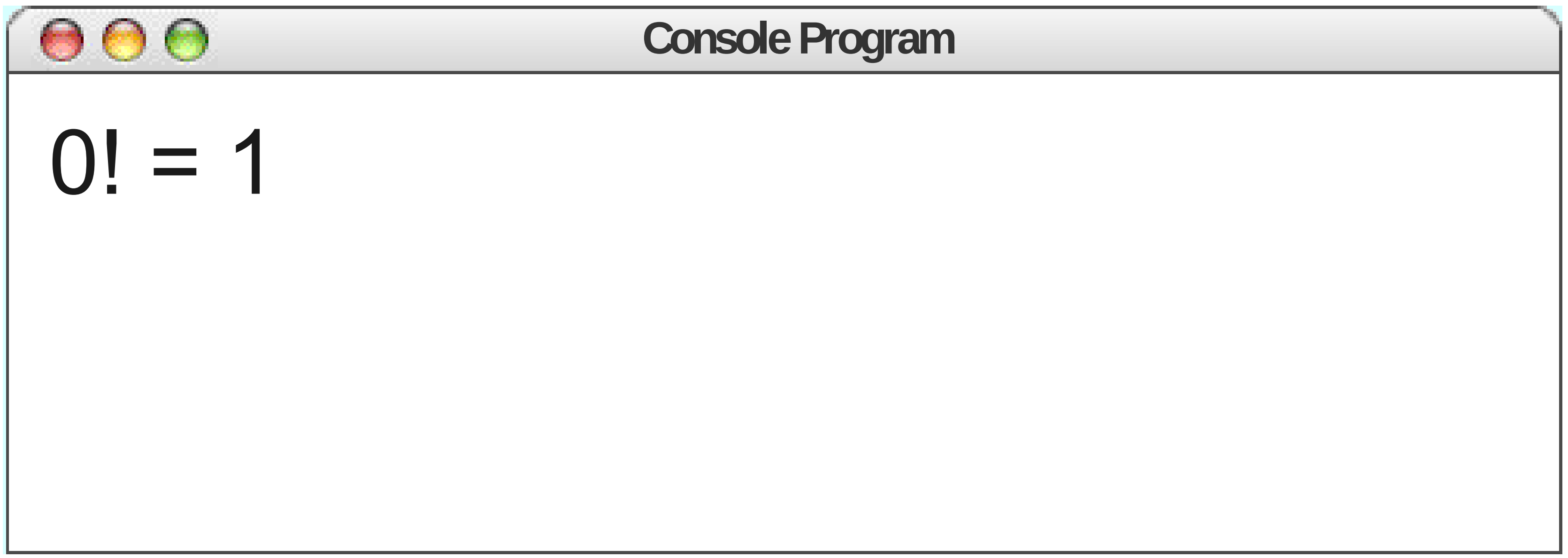
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



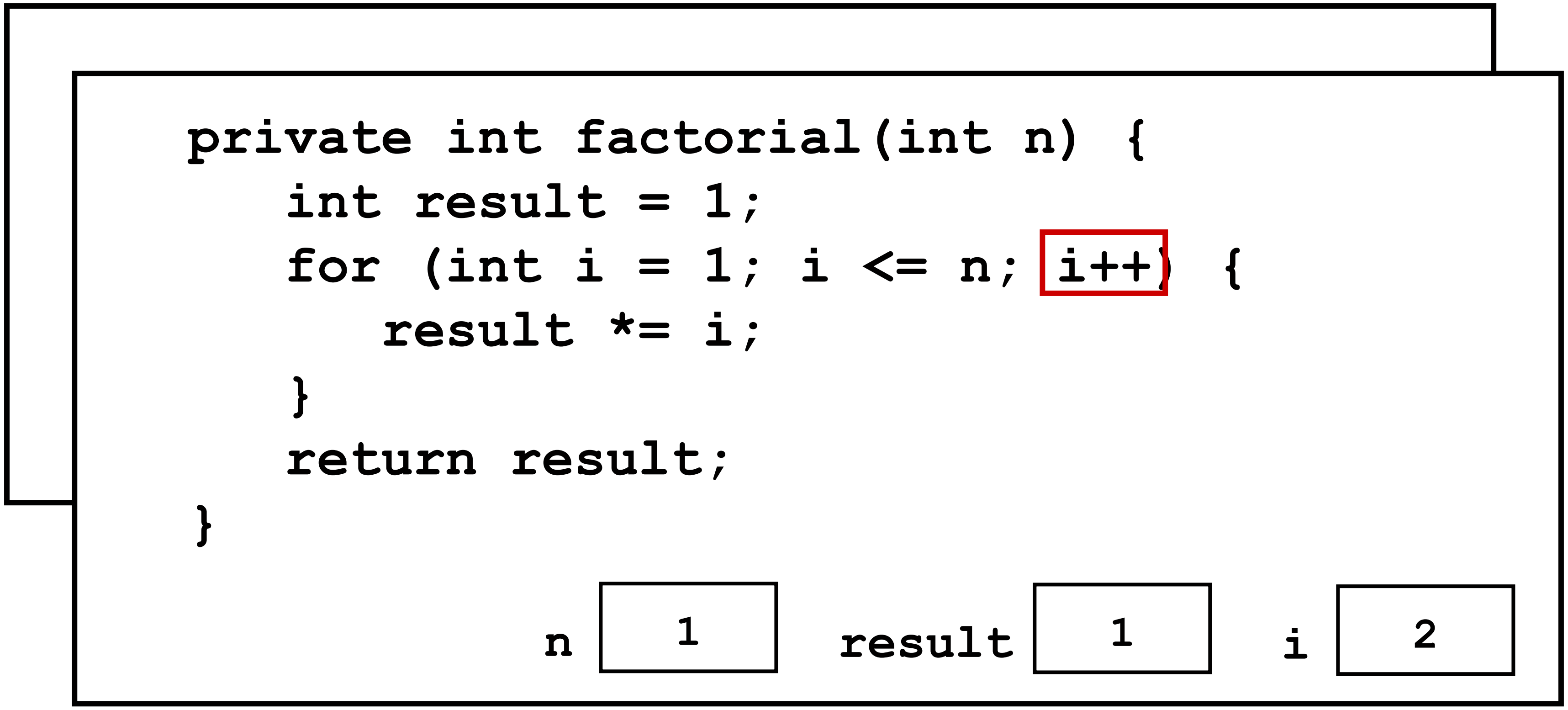
```
0! = 1
```



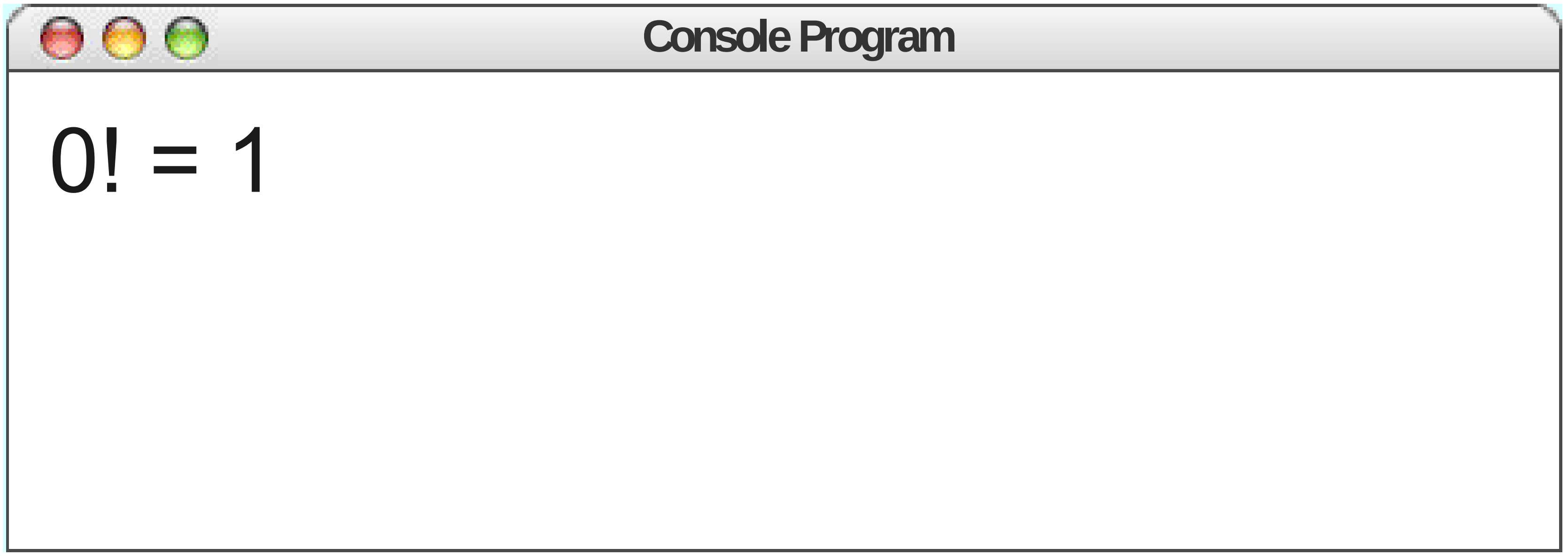
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



```
0! = 1
```



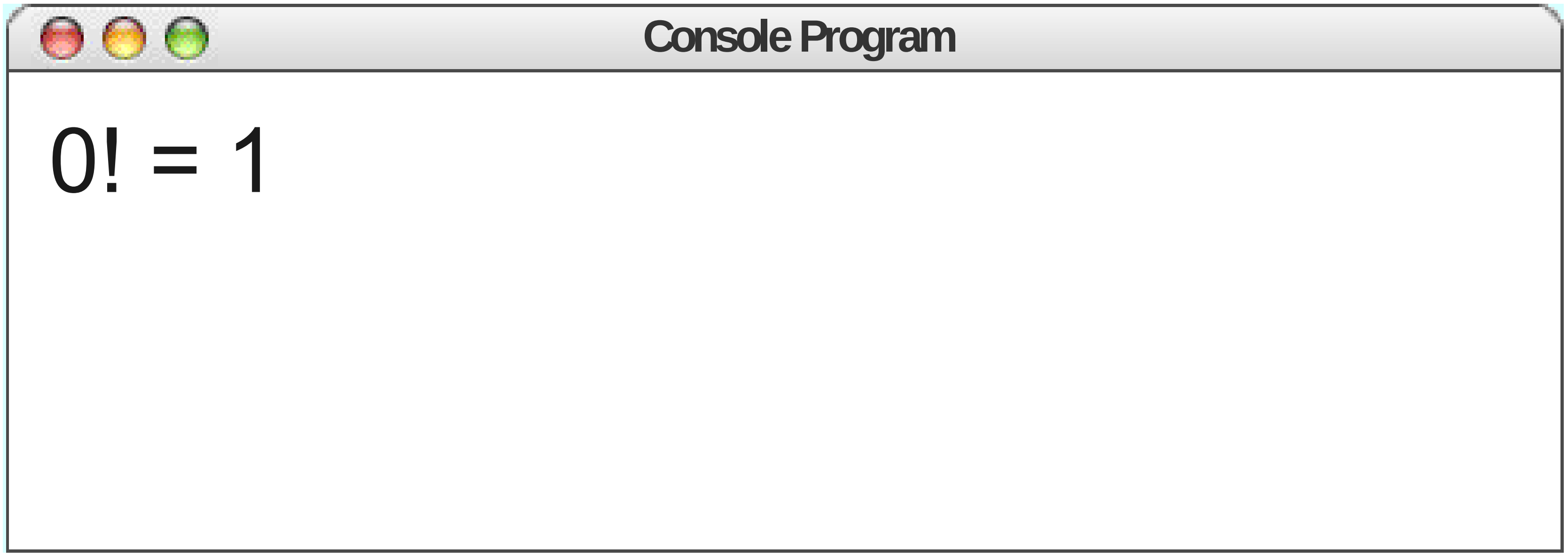
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



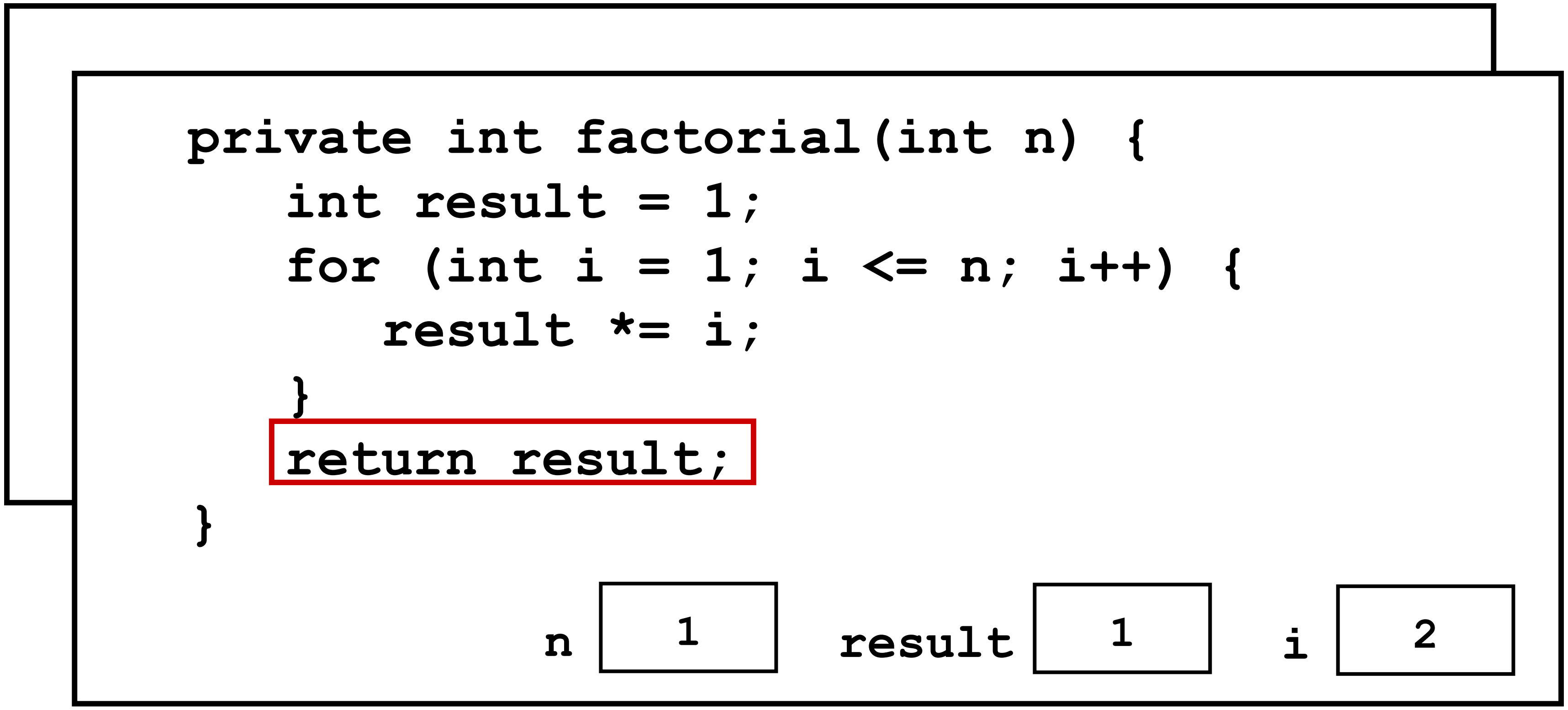
```
0! = 1
```

```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```

n 1   result 1   i 2

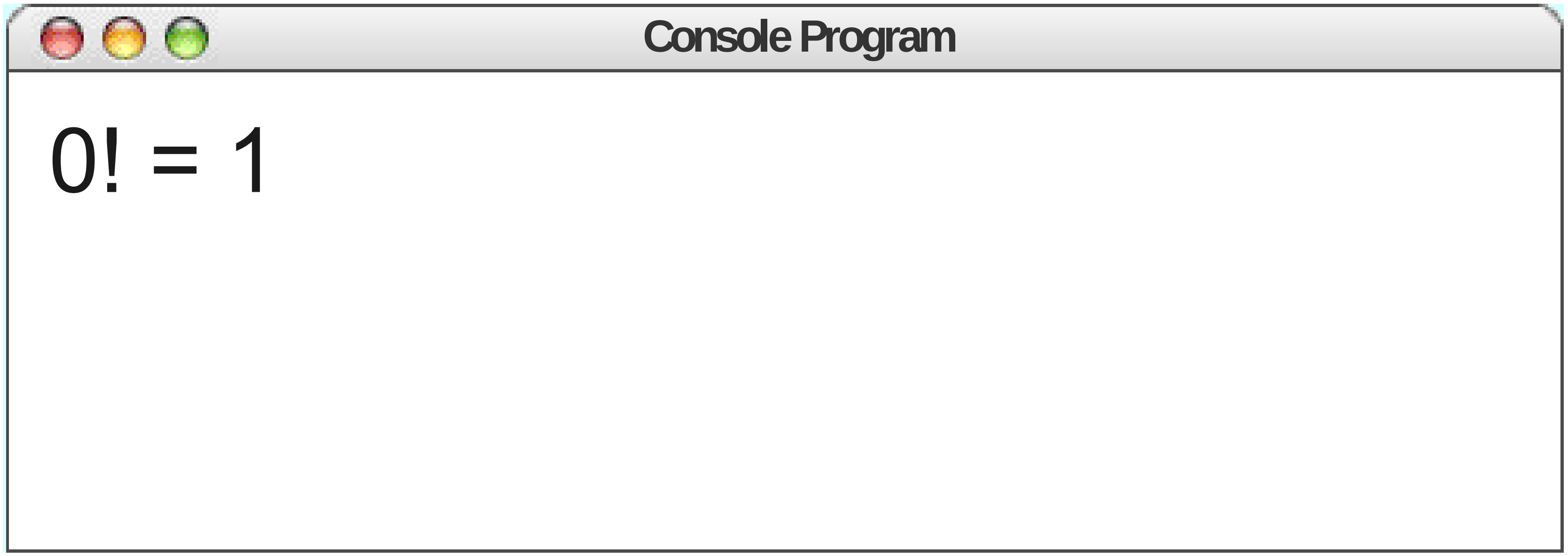


```
0! = 1
```



```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

n 1 result 1 i 2

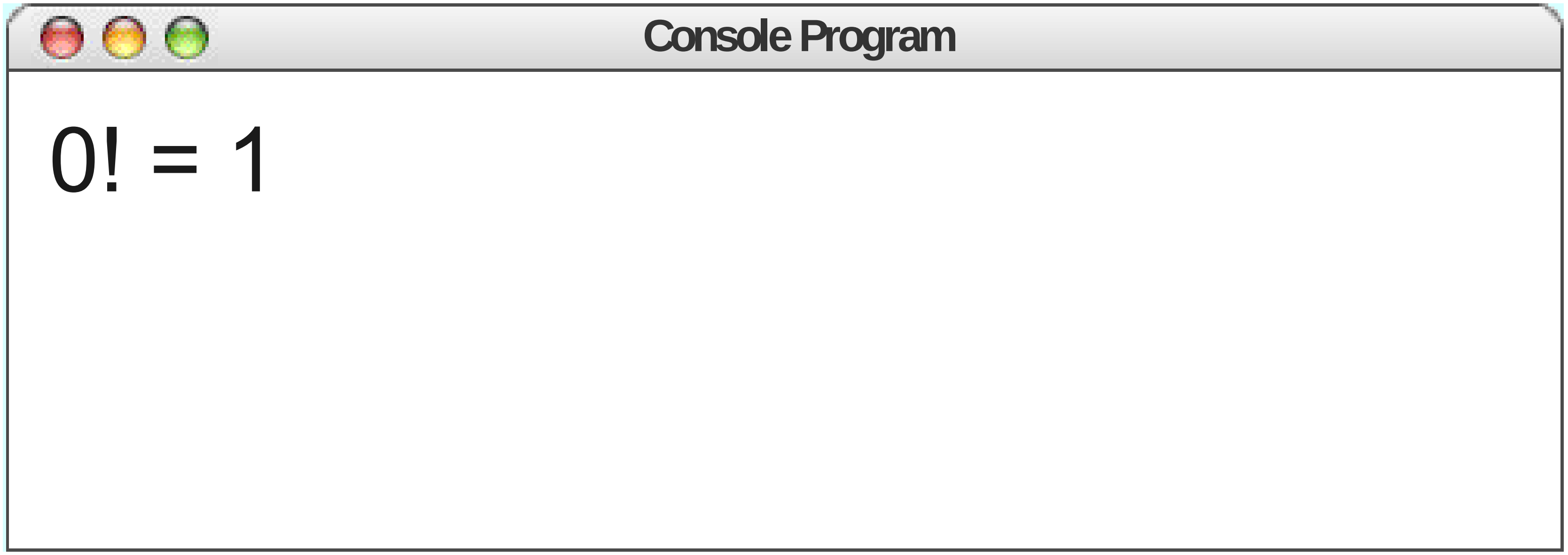


```
0! = 1
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

i $\boxed{1}$

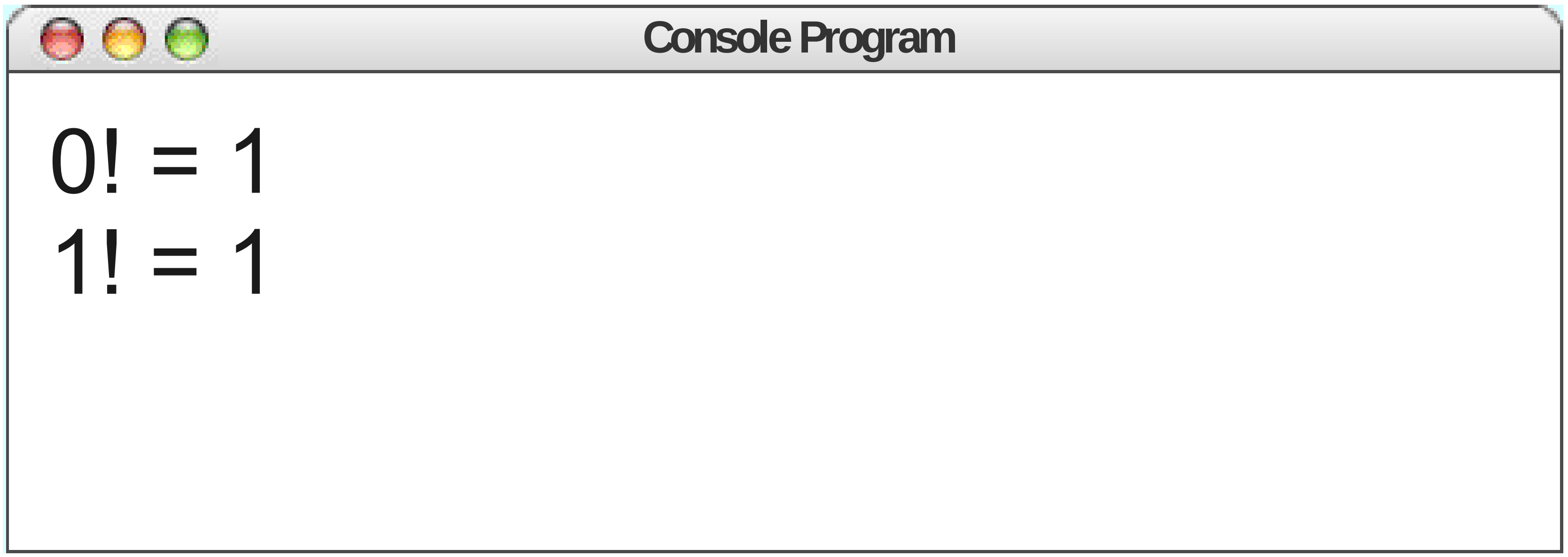


```
0! = 1
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

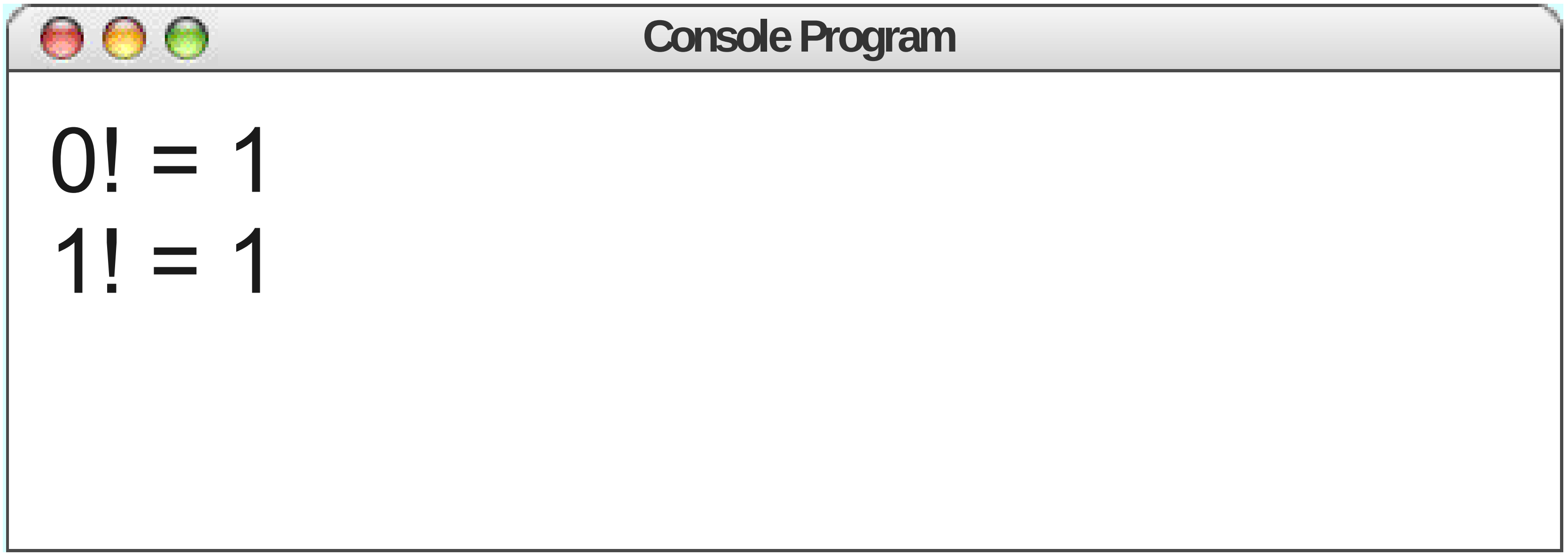
i $\boxed{1}$



```
0! = 1
1! = 1
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

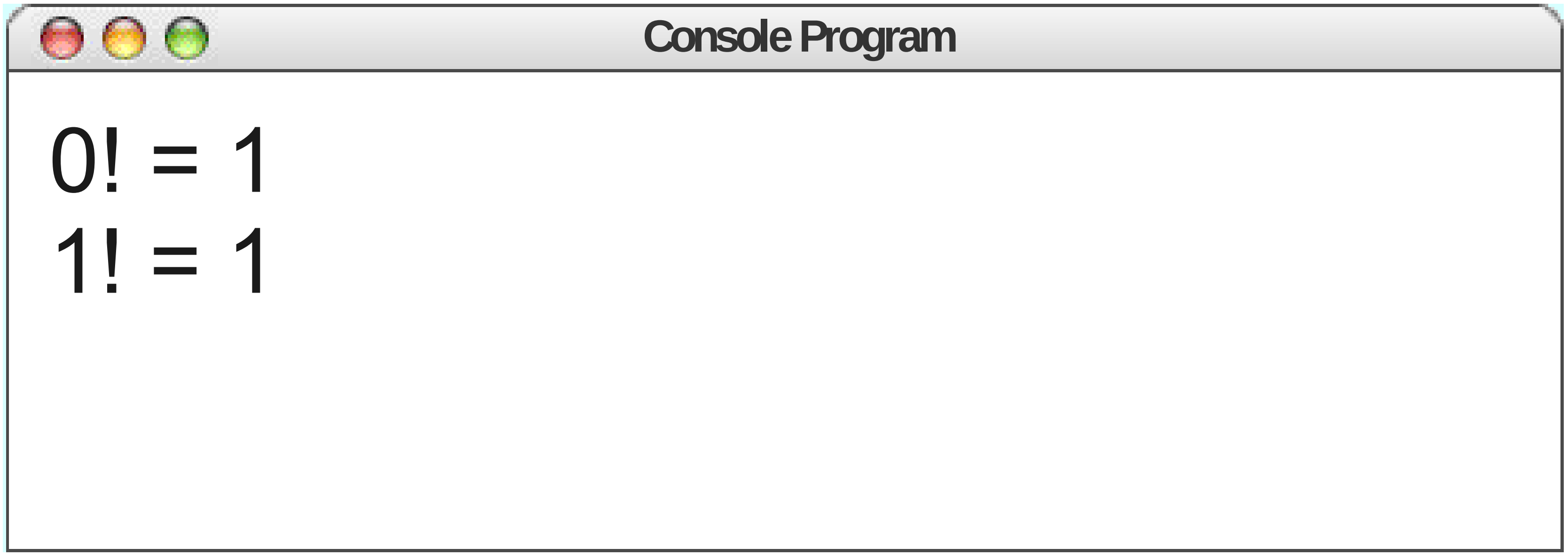
```
i   2
```



```
0! = 1
1! = 1
```

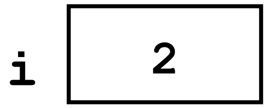
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

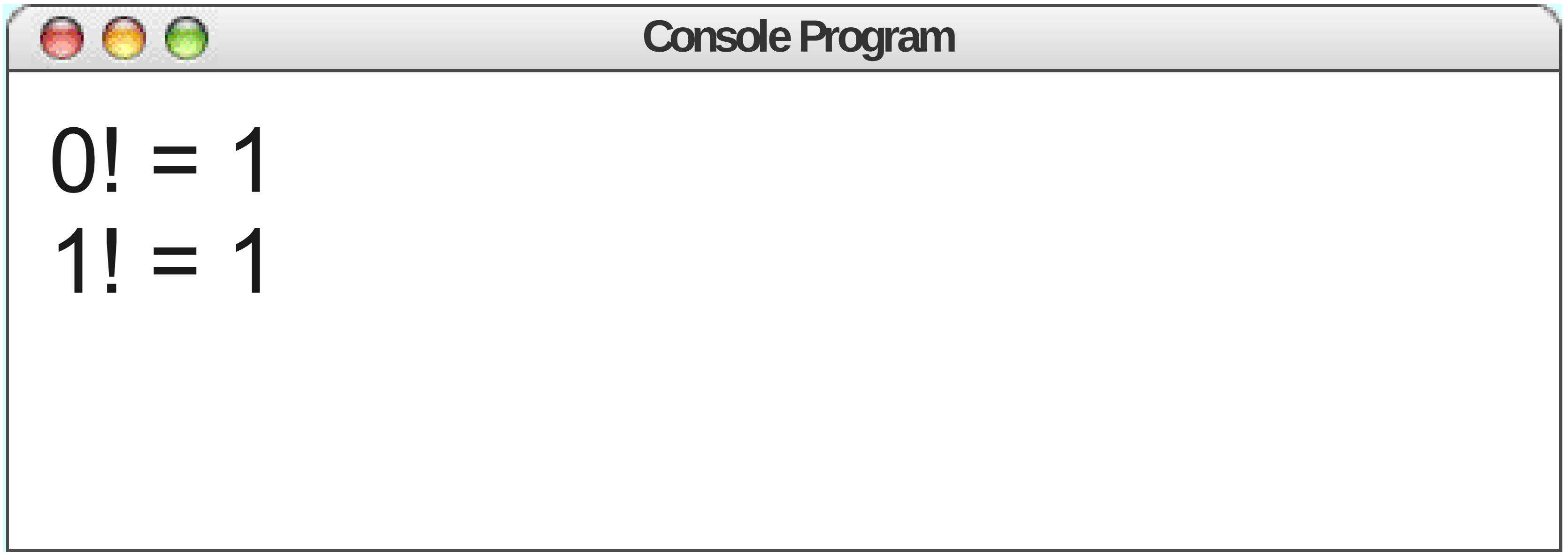
i 2



```
0! = 1
1! = 1
```

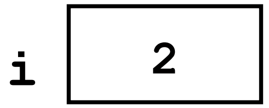
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

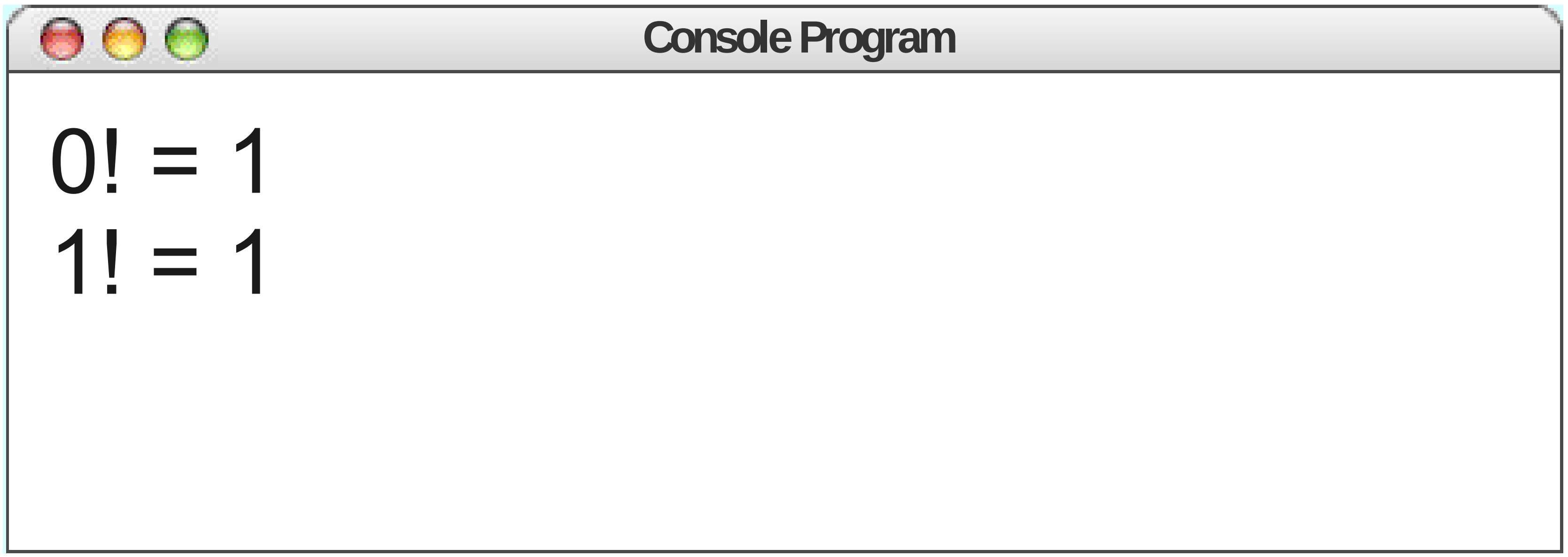




```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```



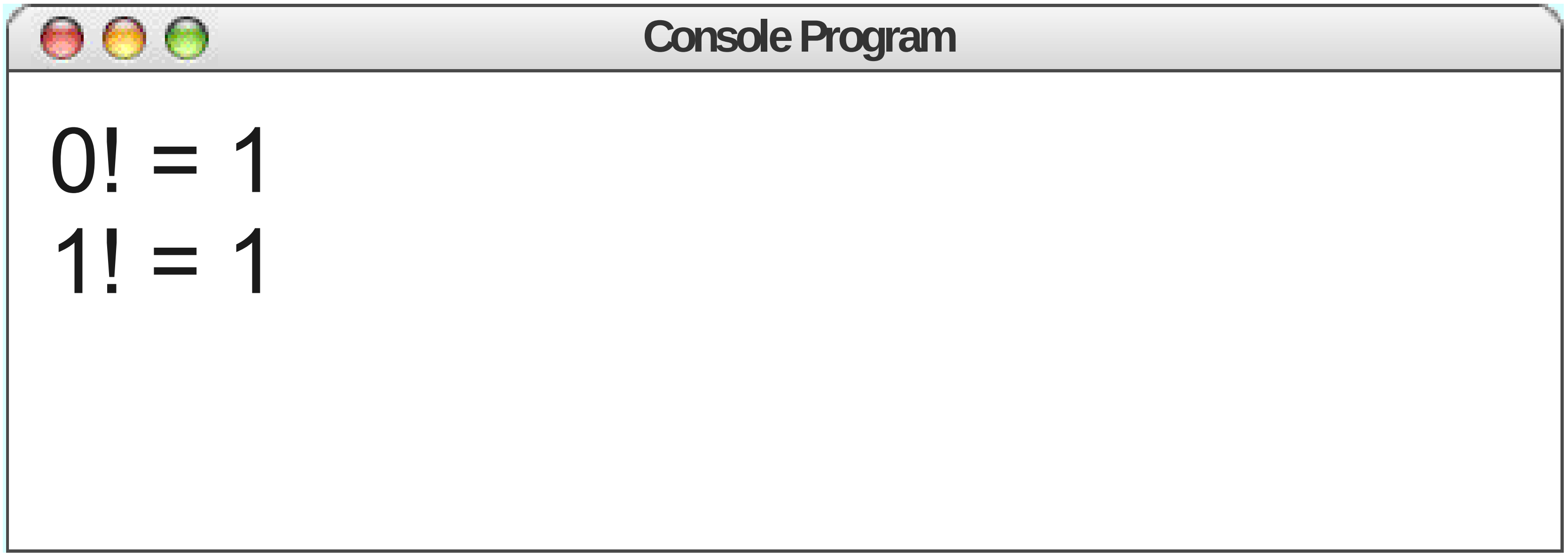


```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

$$\underbrace{\texttt{factorial(i)}}_{\boxed{2}}$$

i $\boxed{2}$

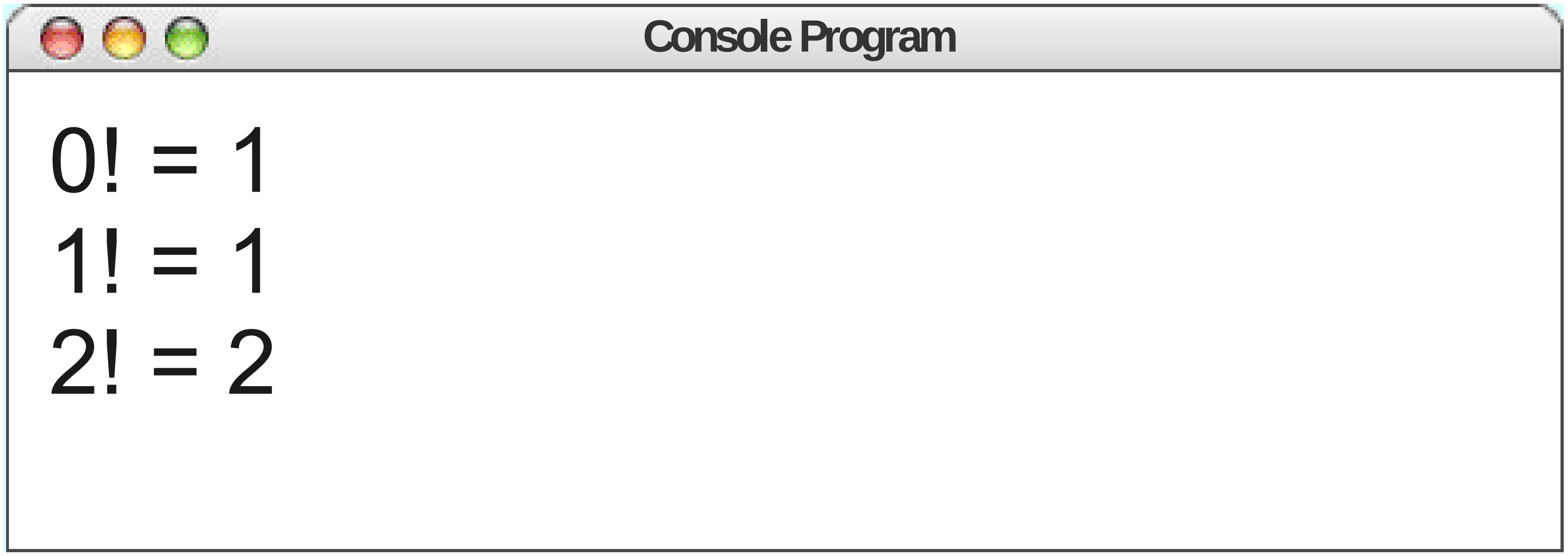


```
0! = 1
1! = 1
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

$\underbrace{\texttt{factorial(i)}}_{\boxed{2}}$

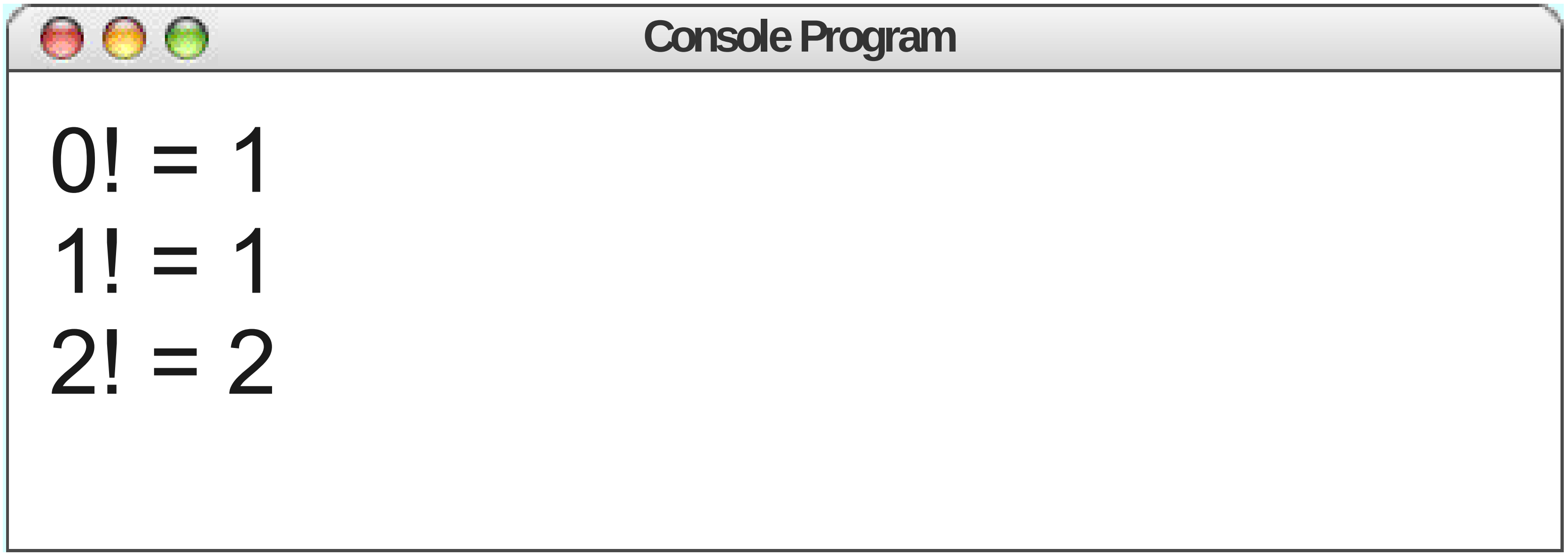
i $\boxed{2}$



```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

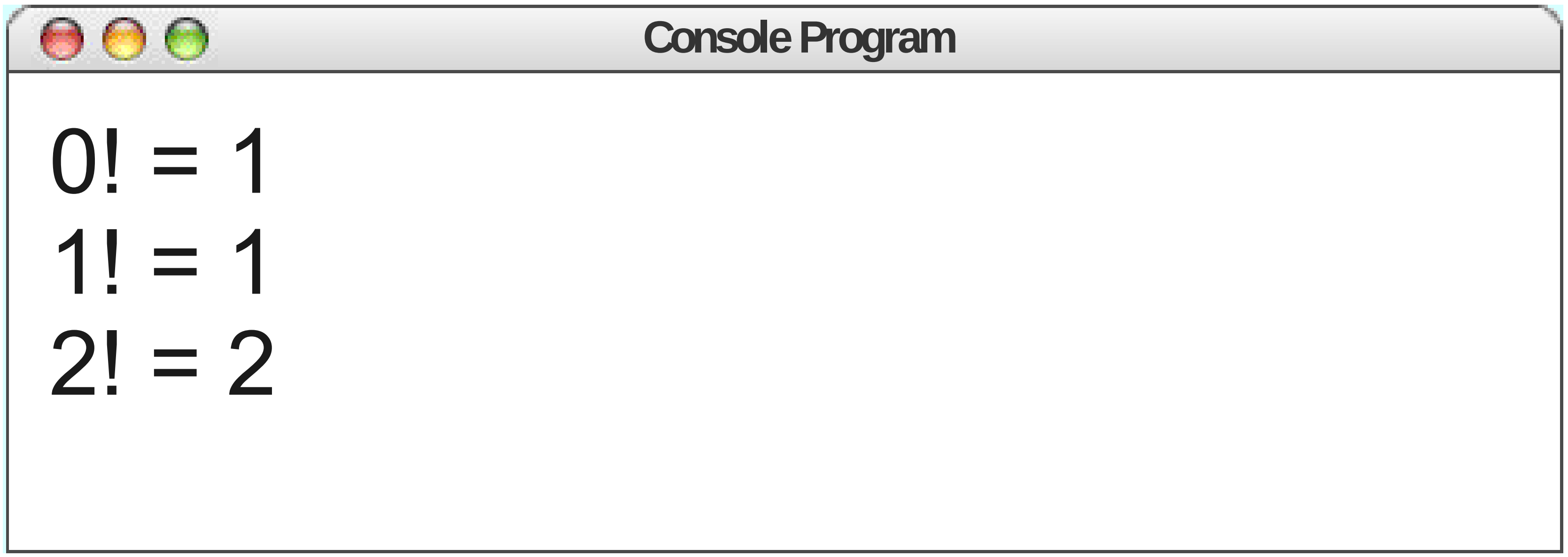
i $\boxed{3}$



```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

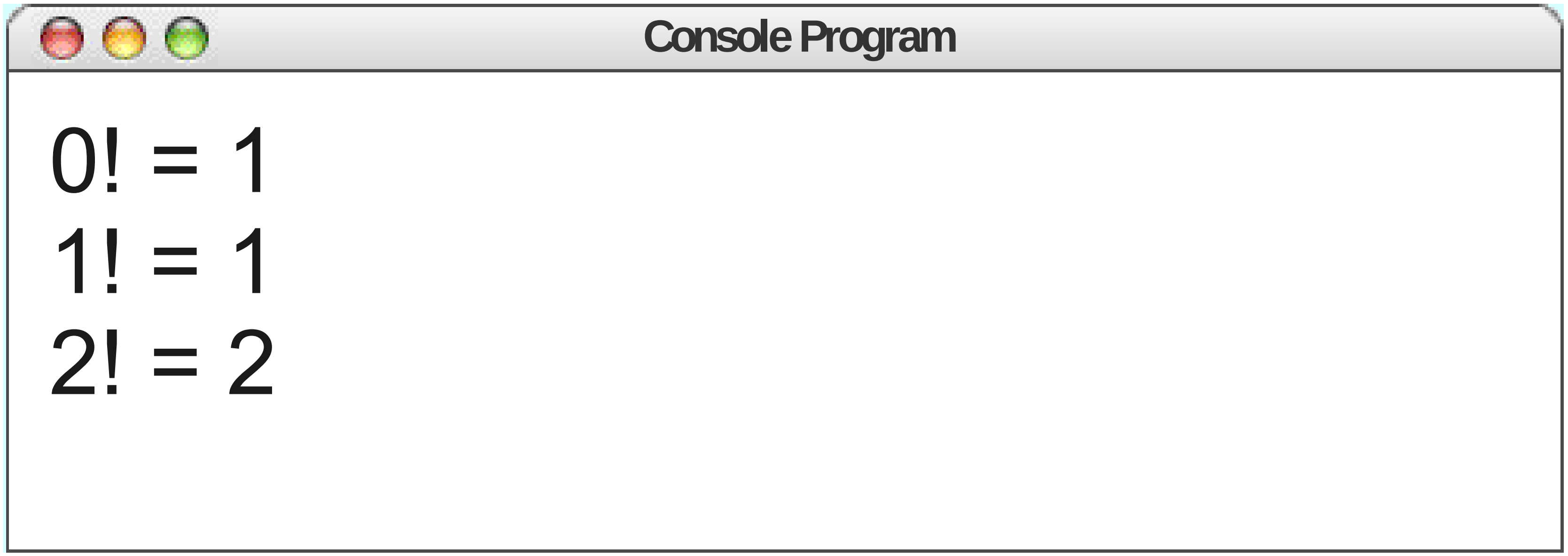
i $\boxed{3}$



```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

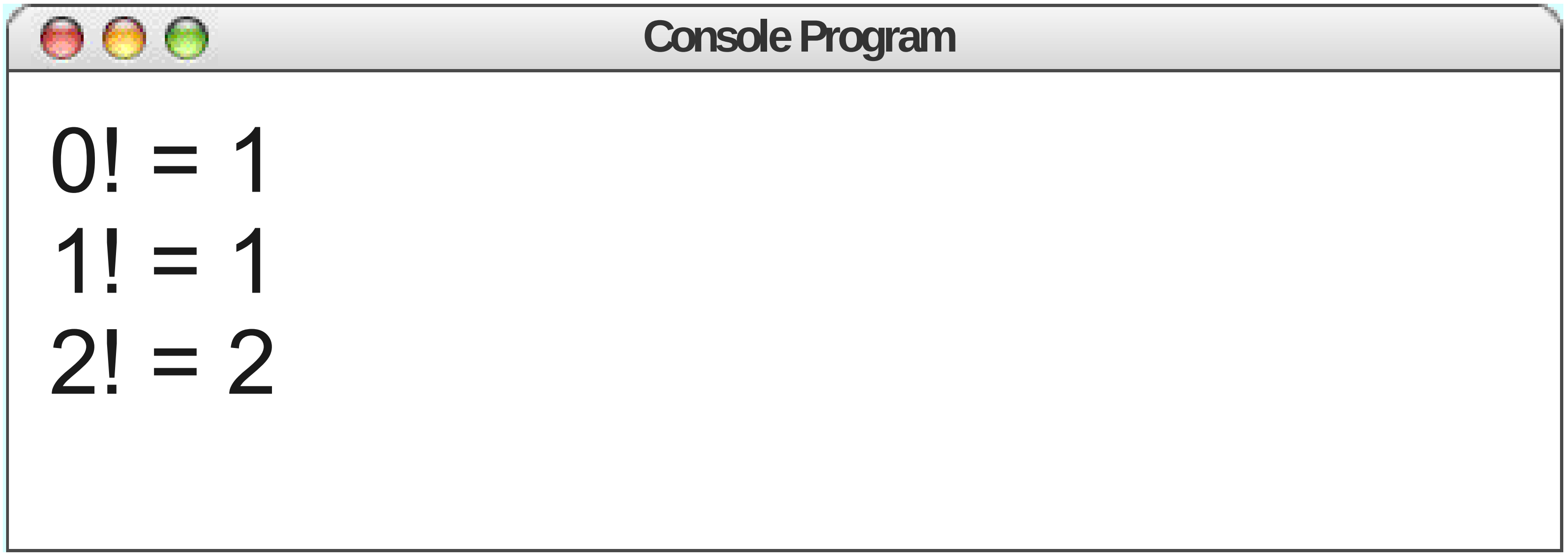
```
i  3
```



```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + $\boxed{factorial(i)}$);
    }
}
```

i $\boxed{3}$

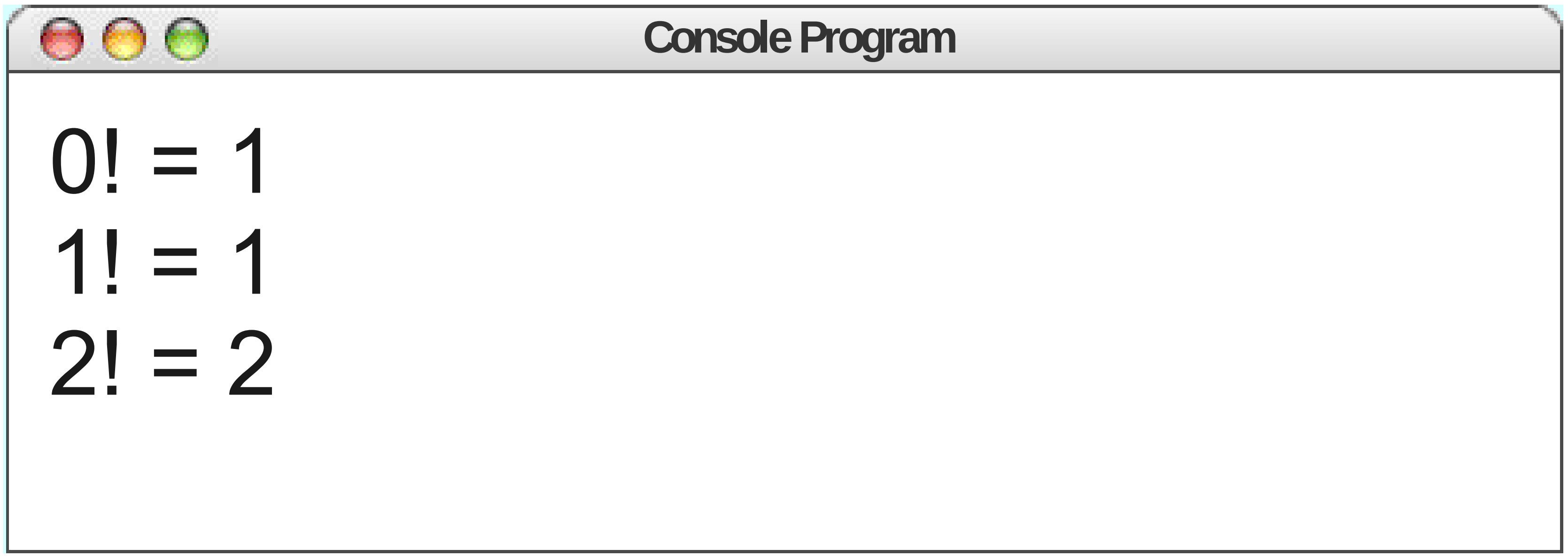


```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

$\underbrace{\texttt{factorial(i)}}_{\boxed{6}}$

i $\boxed{3}$

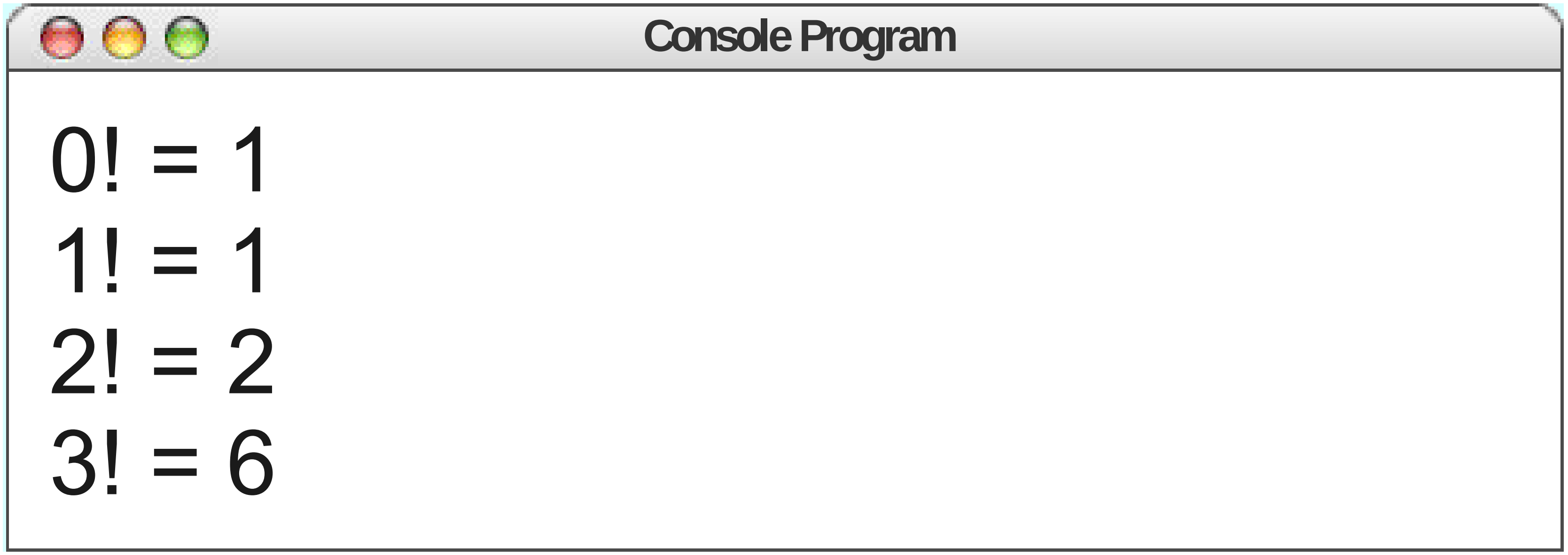


```
0! = 1
1! = 1
2! = 2
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

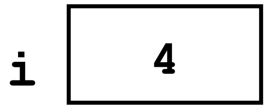
$\underbrace{\texttt{factorial(i)}}_{\boxed{6}}$

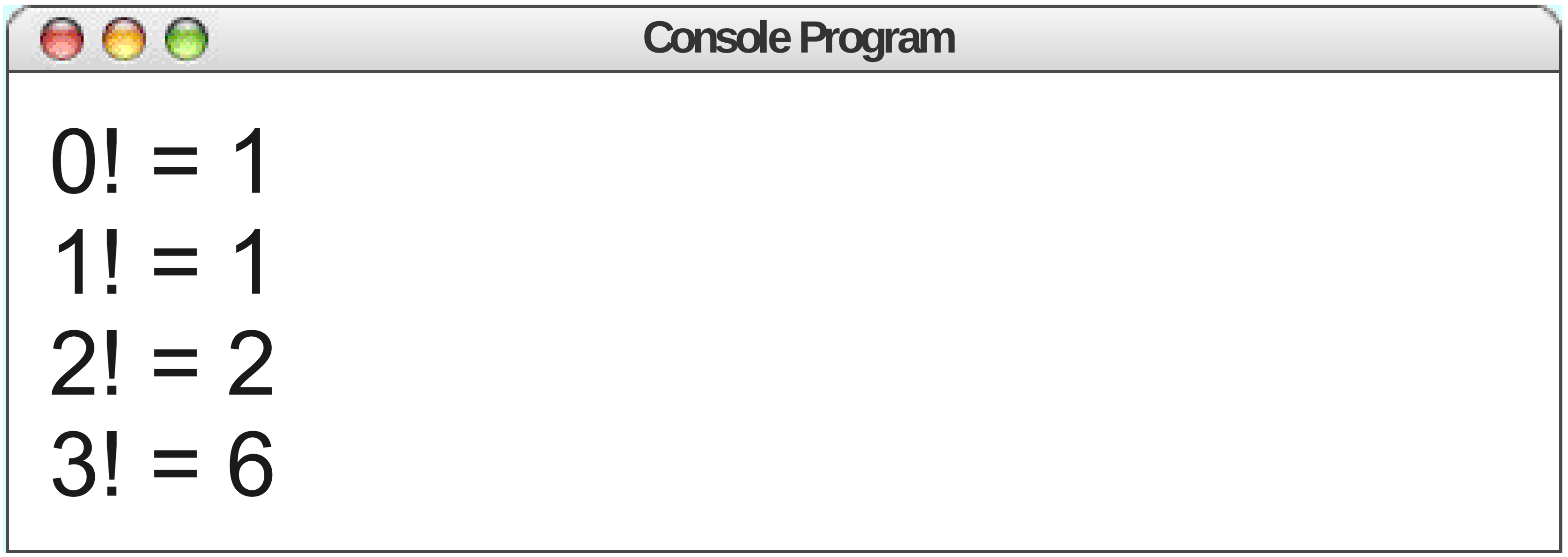
i $\boxed{3}$



```
0! = 1
1! = 1
2! = 2
3! = 6
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

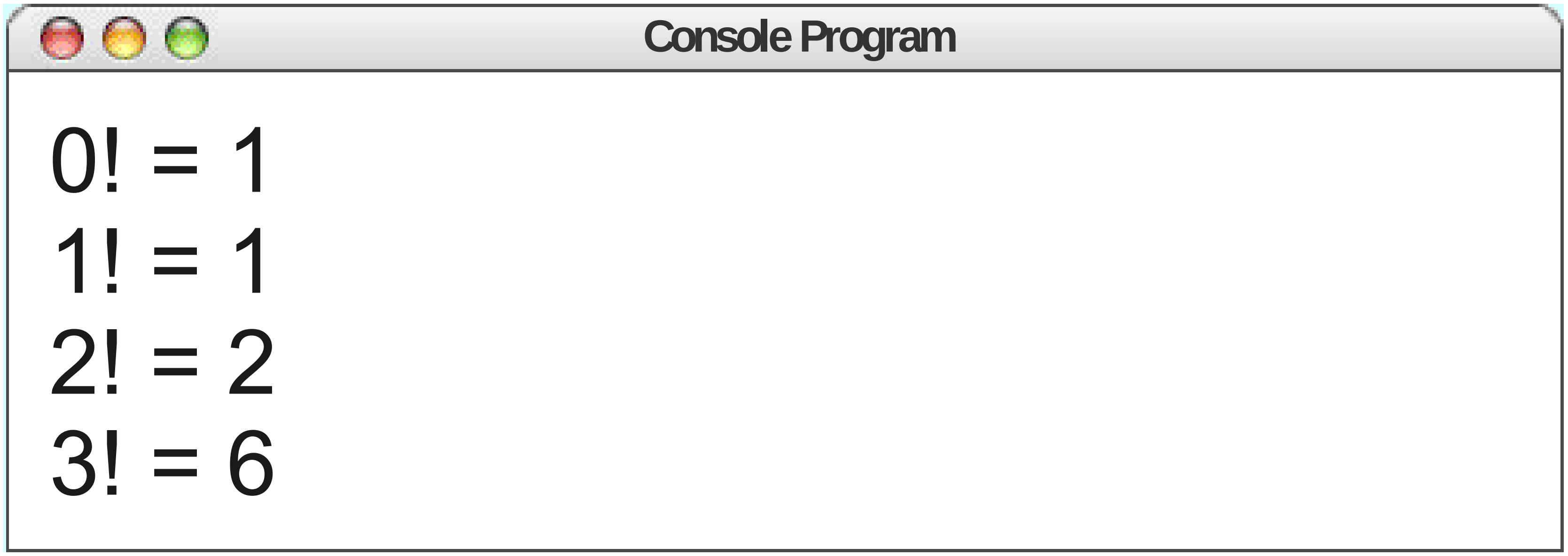




```
0! = 1
1! = 1
2! = 2
3! = 6
```

```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i $\boxed{4}$



```
0! = 1
1! = 1
2! = 2
3! = 6
```

Retiring Young

Pass-by-Value

- Java methods pass their parameters by **value**.
- The method gets a *copy* of its parameters, not the actual parameters themselves.

```
private void myMethod(int x) {
    x = 137;
}
public void run() {
    int x = 42;
    myMethod(x);
    println("The value of x is " + x);
}
```

This statement prints 42, not 137.

Slowing Things Down

The **`pause`** Method

- The **`pause`** method has the signature

```
public void pause(double milliseconds);
```

- **`pause`** waits the specified number of milliseconds, then returns.
- Examples:
  - **`pause(1000);`** waits for one second
  - **`pause(50);`** waits for one twentieth of a second.

Operations on the `GObject` Class

The following operations apply to all `GObjects`:

object`.setColor(`*color*`)` Sets the color of the object to the specified color constant.
object`.setLocation(`*x*`, `*y*`)` Changes the location of the object to the point (*x*, *y*).
object`.move(`*dx*`, `*dy*`)` Moves the object on the screen by adding *dx* and *dy* to its current coordinates.

Standard color names defined in the `java.awt` package:

`Color.BLACK`	`Color.RED`	`Color.BLUE`
`Color.DARK_GRAY`	`Color.YELLOW`	`Color.MAGENTA`
`Color.GRAY`	`Color.GREEN`	`Color.ORANGE`
`Color.LIGHT_GRAY`	`Color.CYAN`	`Color.PINK`
`Color.WHITE`		

Graphic courtesy of Eric Roberts

Operations on the **`GObject`** Class

The following operations apply to all **`GObjects`**:

object`.`**`setColor`**`(`*color*`)`
Sets the color of the object to the specified color constant.

object`.`**`setLocation`**`(`*x*`,` *y*`)`
Changes the location of the object to the point (*x*, *y*).

object`.`**`move`**`(`*dx*`,` *dy*`)`
Moves the object on the screen by adding *dx* and *dy* to its current coordinates.

Standard color names defined in the `java.awt` package:

`Color.BLACK`	`Color.RED`	`Color.BLUE`
`Color.DARK_GRAY`	`Color.YELLOW`	`Color.MAGENTA`
`Color.GRAY`	`Color.GREEN`	`Color.ORANGE`
`Color.LIGHT_GRAY`	`Color.CYAN`	`Color.PINK`
`Color.WHITE`		

Graphic courtesy of Eric Roberts

Animation

- By repositioning objects after they have been added to the canvas, we can create animations.
- General pattern for animation:

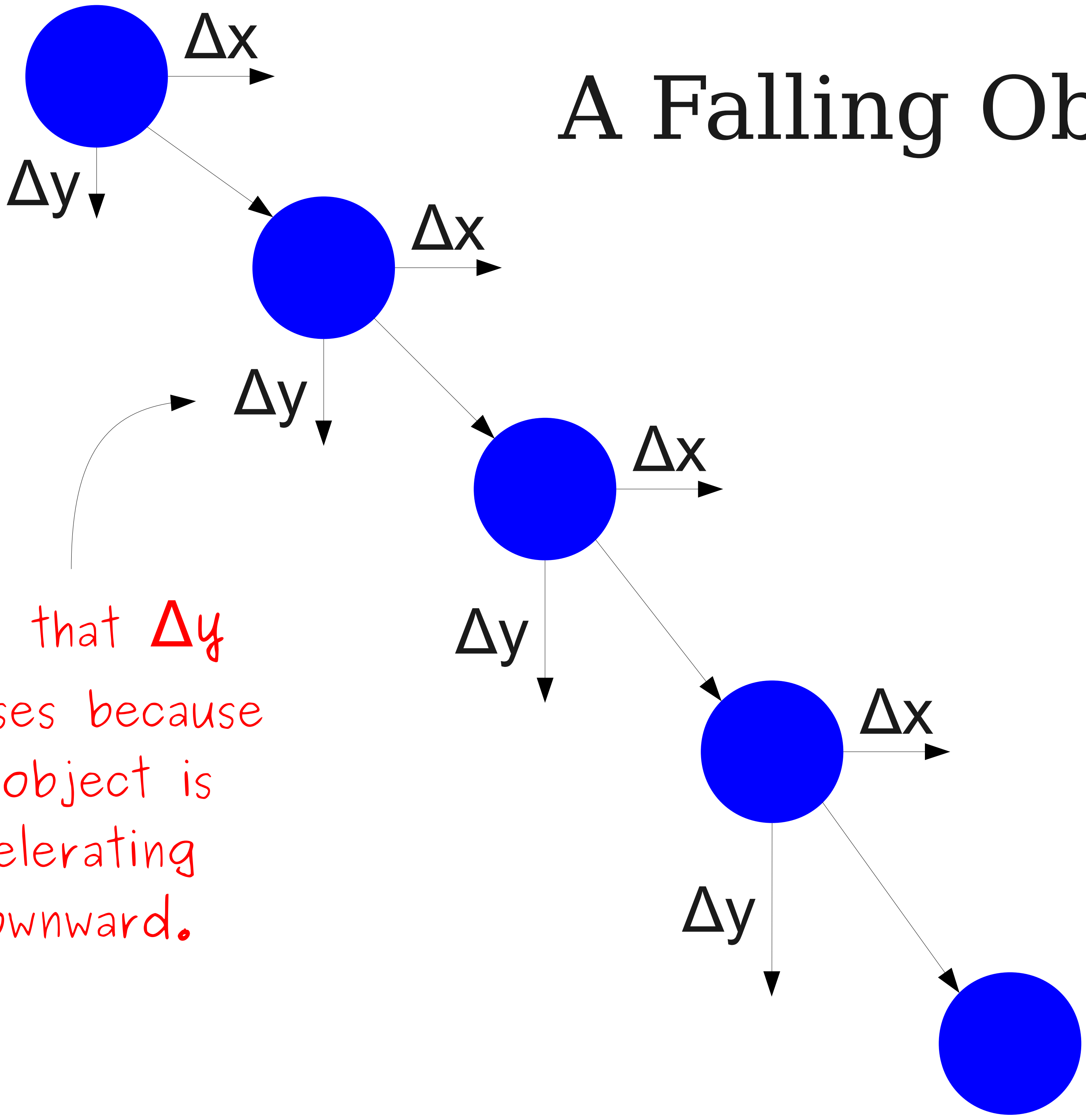
```
while (not-done-condition) {
    update graphics
    pause(pause-time);
}
```

Physics Simulation

http://physbam.stanford.edu/~fedkiw/animations/glass00.avi

http://physbam.stanford.edu/~fedkiw/animations/motion_smoke.avi

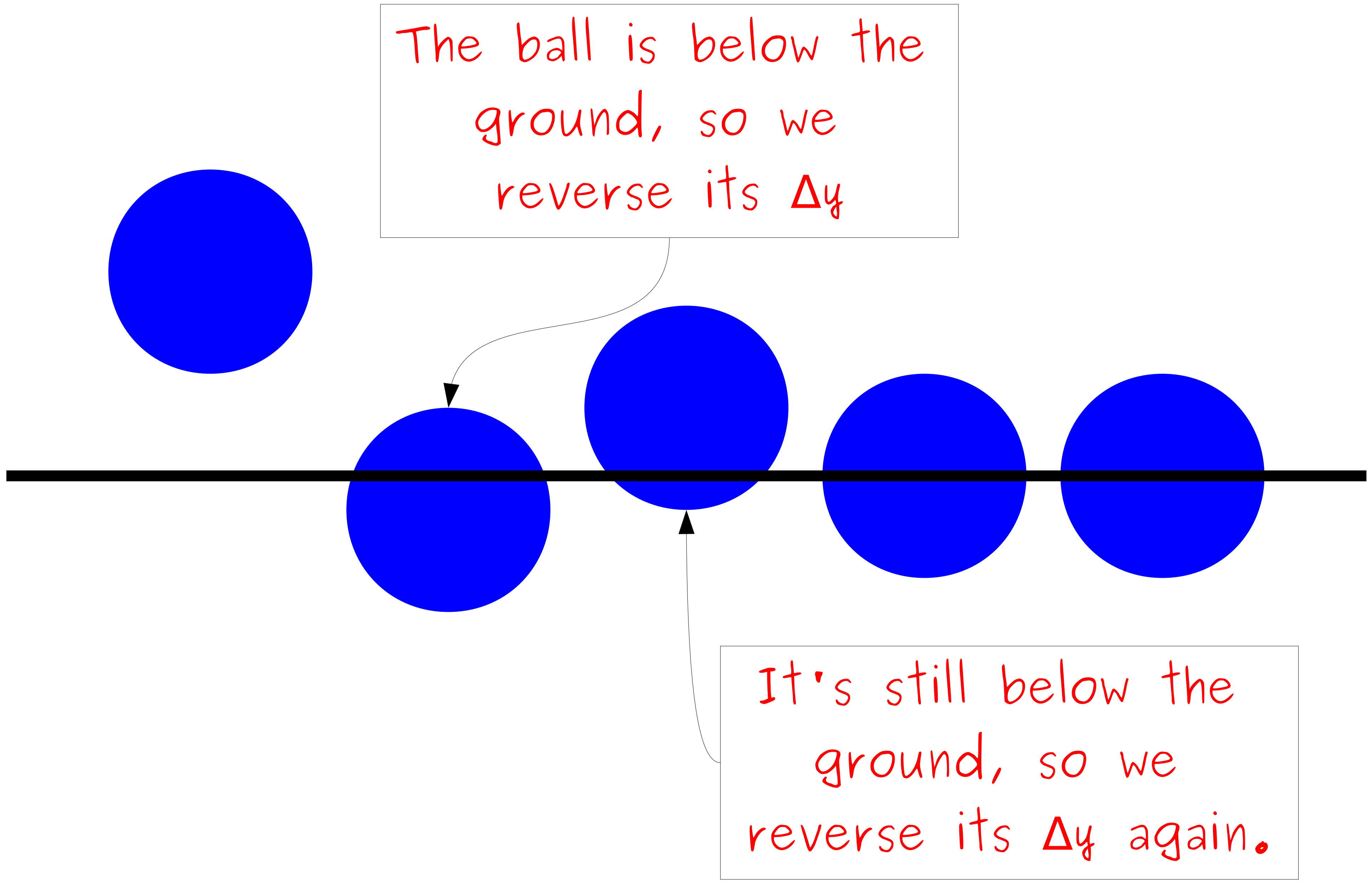
A Falling Object



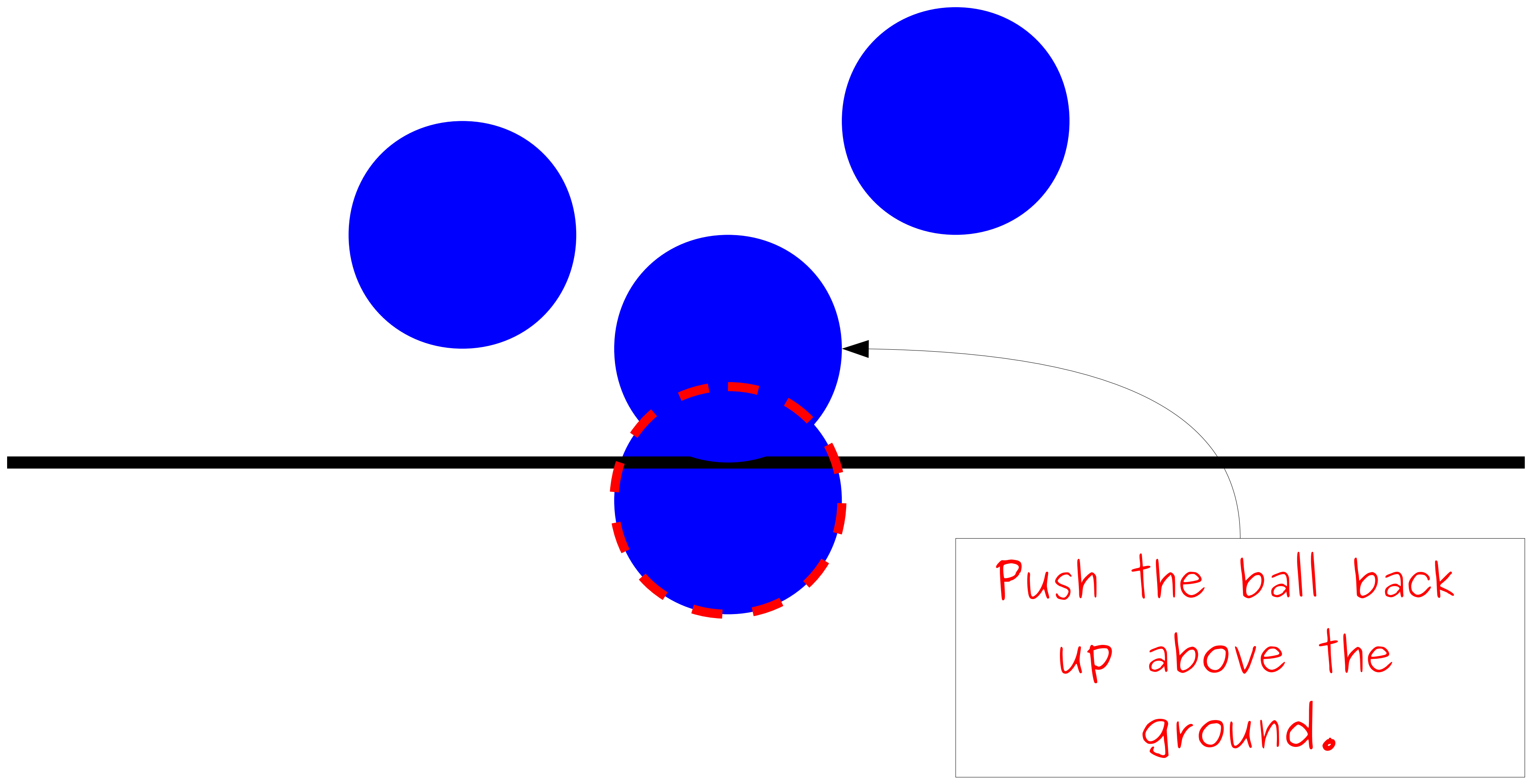
Note that Δy increases because the object is accelerating downward.

Let's Code It Up!

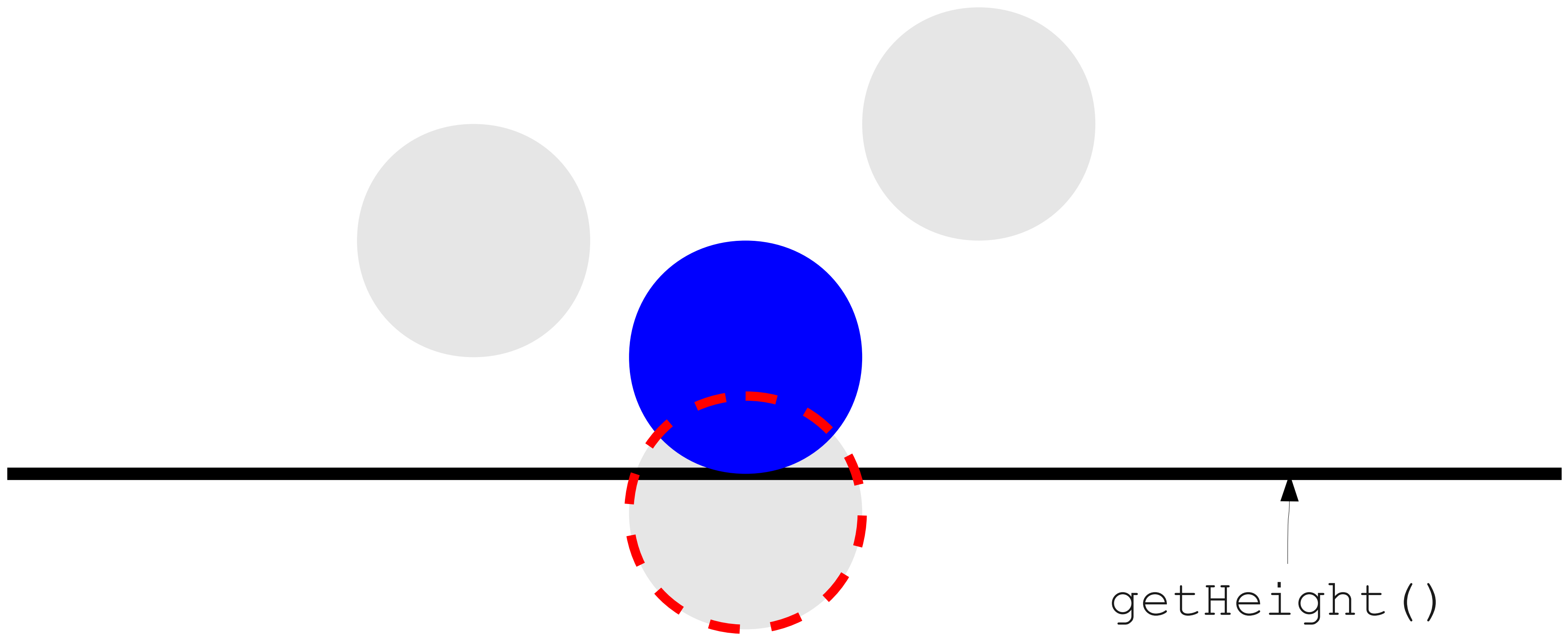
A Sticky Situation


The ball is below the ground, so we reverse its Δy
It's still below the ground, so we reverse its Δy again.

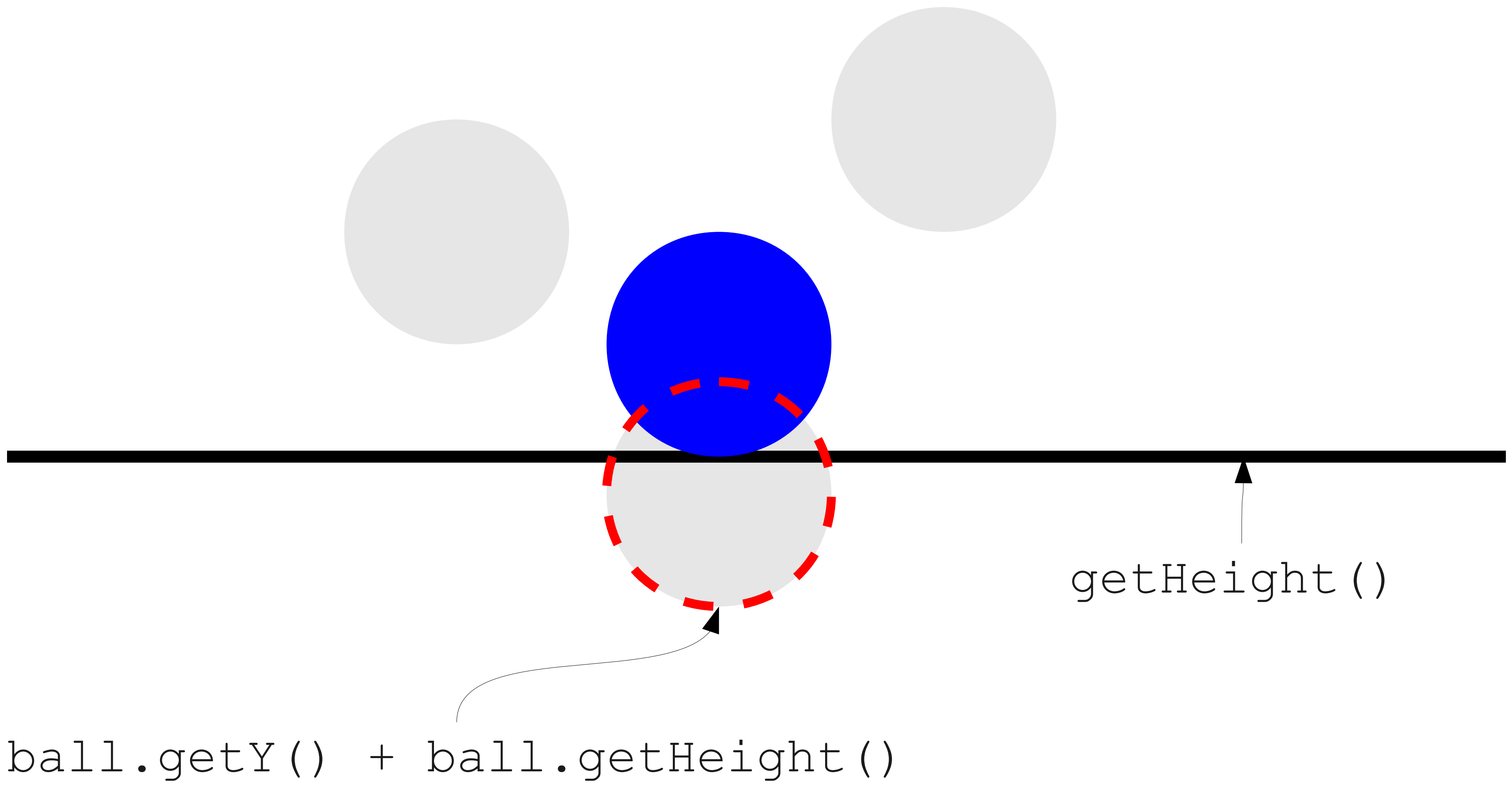
Unsticking the Situation



Unsticking the Situation



Unsticking the Situation


getHeight()
ball.getY() + ball.getHeight()