Methods

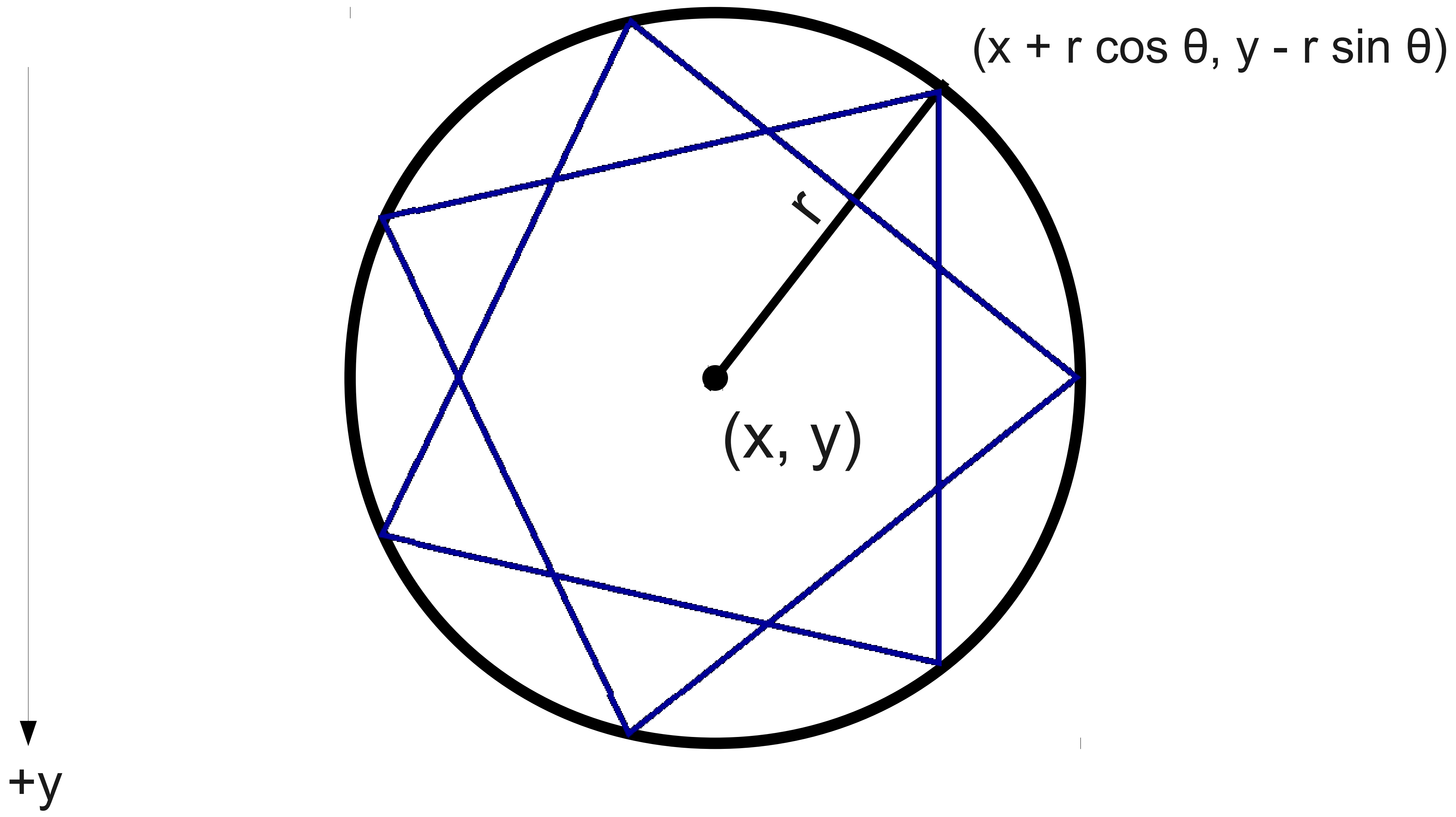
Friday Four Square Today!

Gates, 4:15PM

An Interesting Radio Show

This American Life:
“Mr. Daisey and the Apple Factory”

http://www.thisamericanlife.org/radio-archives/episode/454/mr-daisey-and-the-apple-factory



Each point k is connected to point k + 2, after wrapping around.

Point k is at $\frac{k}{numSides} \times 360^{o}$

Passing Parameters

- A method can accept **parameters** when it is called.
- Syntax:

```
private void name(parameters) {
    /* … method body … */
}
```

- The values of the parameters inside the method are set when the method is called.
- The values of the parameters can vary between calls.

For more on the geometry
and properties of stars:

http://en.wikipedia.org/wiki/Star_polygon

Factorials

- The number **n factorial**, denoted **n!**, is

$$1 \times 2 \times 3 \times \ldots \times (n - 1) \times n$$

- For example:
  - $3! = 1 \times 2 \times 3 = 6$.
  - $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$
  - $0! = 1$ (by definition)
- Factorials show up everywhere:
  - Taylor series.
  - Counting ways to shuffle a deck of cards.
  - Determining how quickly computers can sort values.

Returning Values

- A method may produce a value that can be read by its caller.
- To indicate that a method returns a value, specify the type returned in the method declaration:

```
private type name(parameters) {
    /* … method body … */
}
```

- A value can be returned with the **`return`** statement:

```
return value;
```

Subtleties of **`return`**

- If a method has non-**`void`** return type, it must always return a value.

```
private int thisIsWrong(int x) {
    if (x == 5) {
        return 0;
    }
}
```

What do we return if **`x != 5`**?

Subtleties of **`return`**

- If a method has non-**`void`** return type, it must always return a value.

```
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    } else {
        return 1;
    }
}
```

Many Happy **`returns`**

- A method may have multiple return statements. The method ends as soon as **`return`** is executed.

```
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    } else {
        return 1;
    }
}
```

Many Happy **`returns`**

- A method may have multiple return statements. The method ends as soon as **`return`** is executed.

```
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    }
    return 1;
}
```

The only way we can get here is if x is not equal to 5.

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!
```

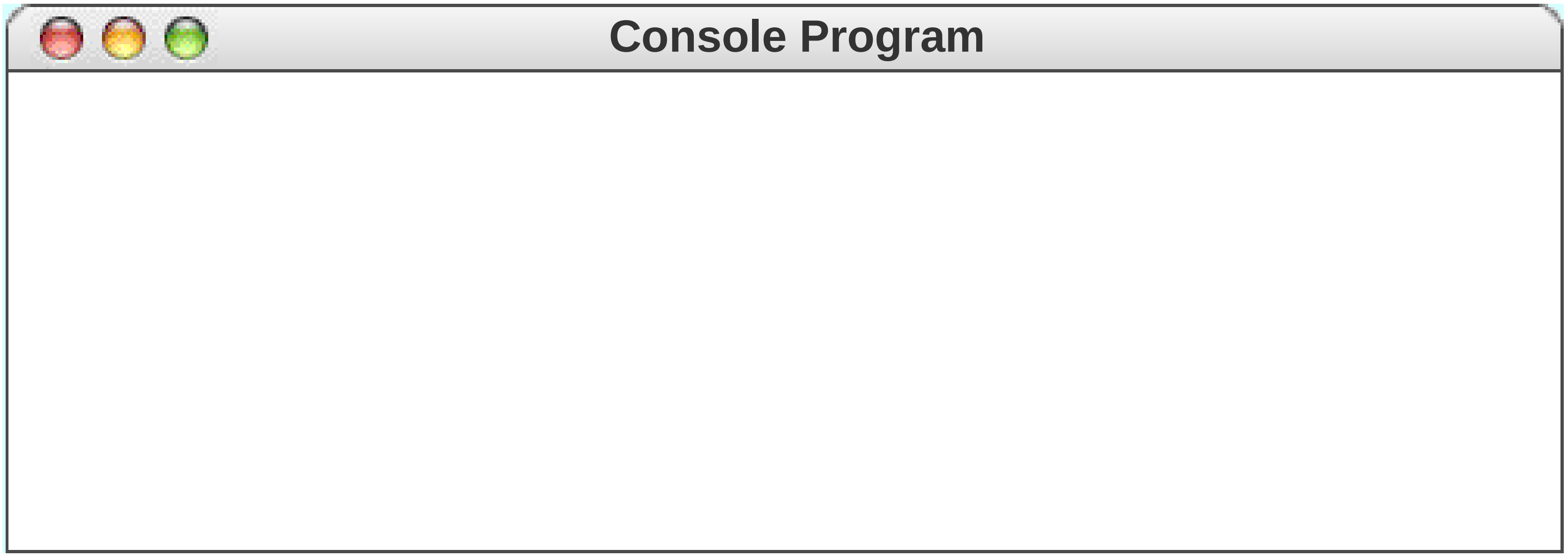
Scope of Method Calls

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

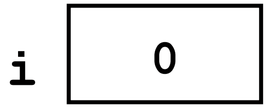
```
public void run() {
    int x = 5;
    someOtherMethod();
}
private void someOtherMethod() {
    x = 4; // Error!
}
```

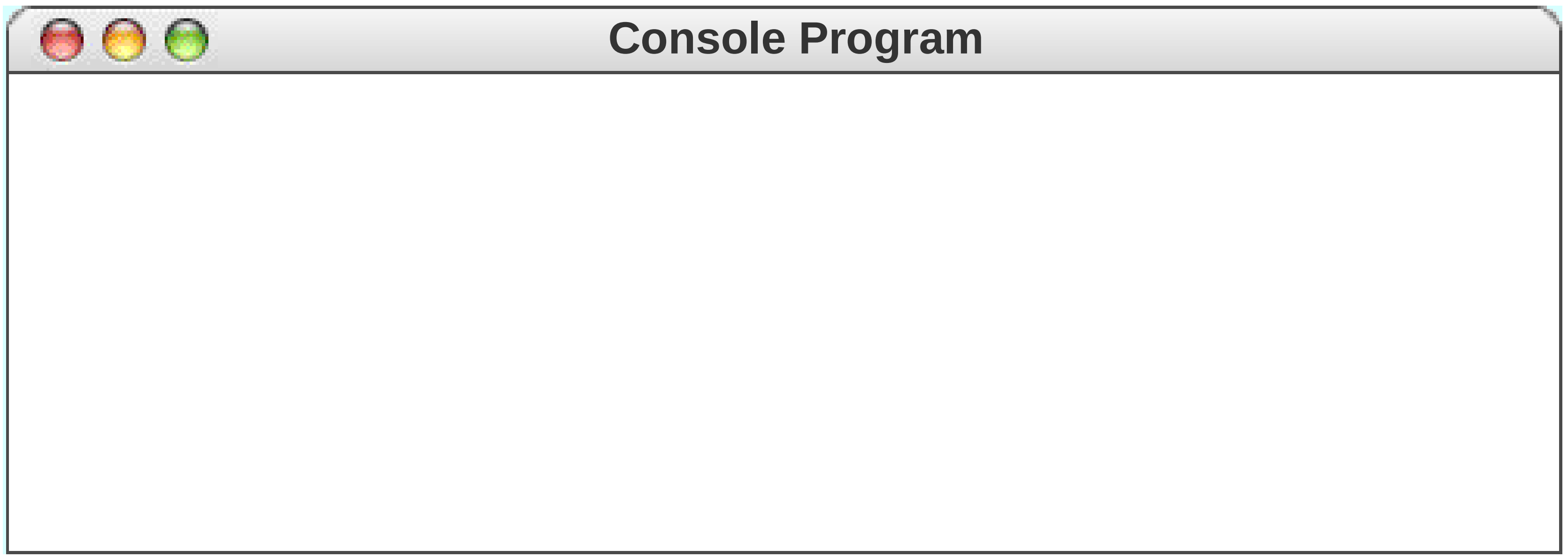
```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

i



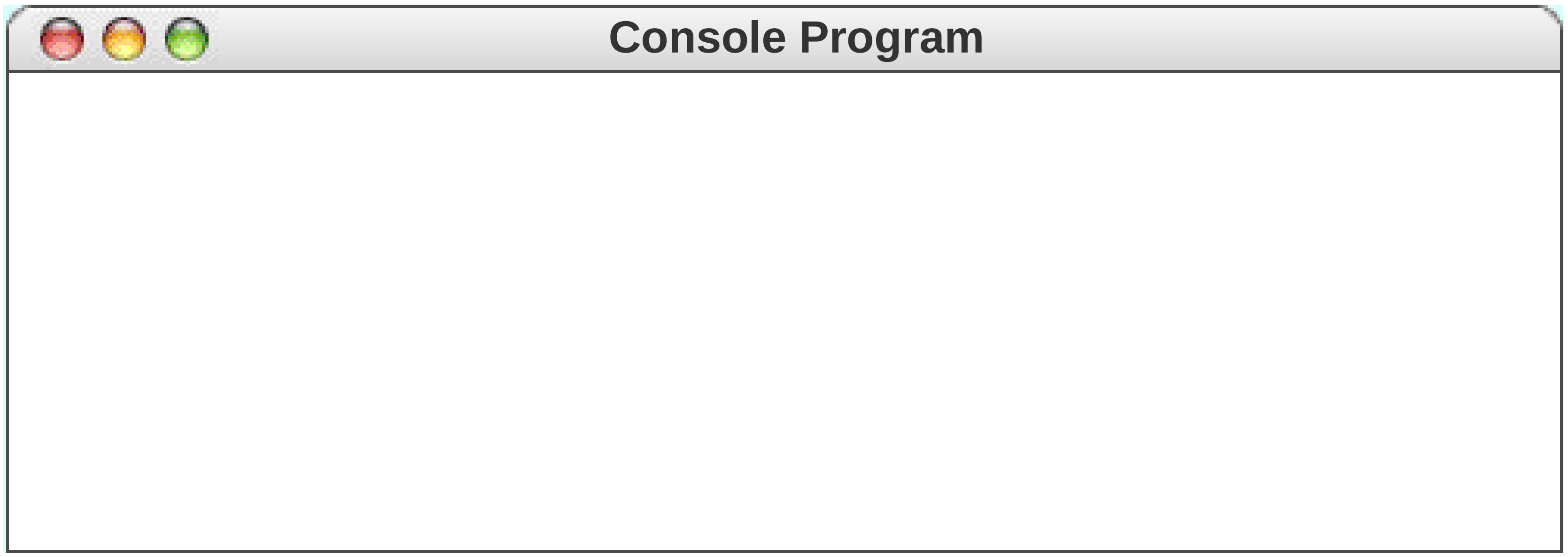
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```





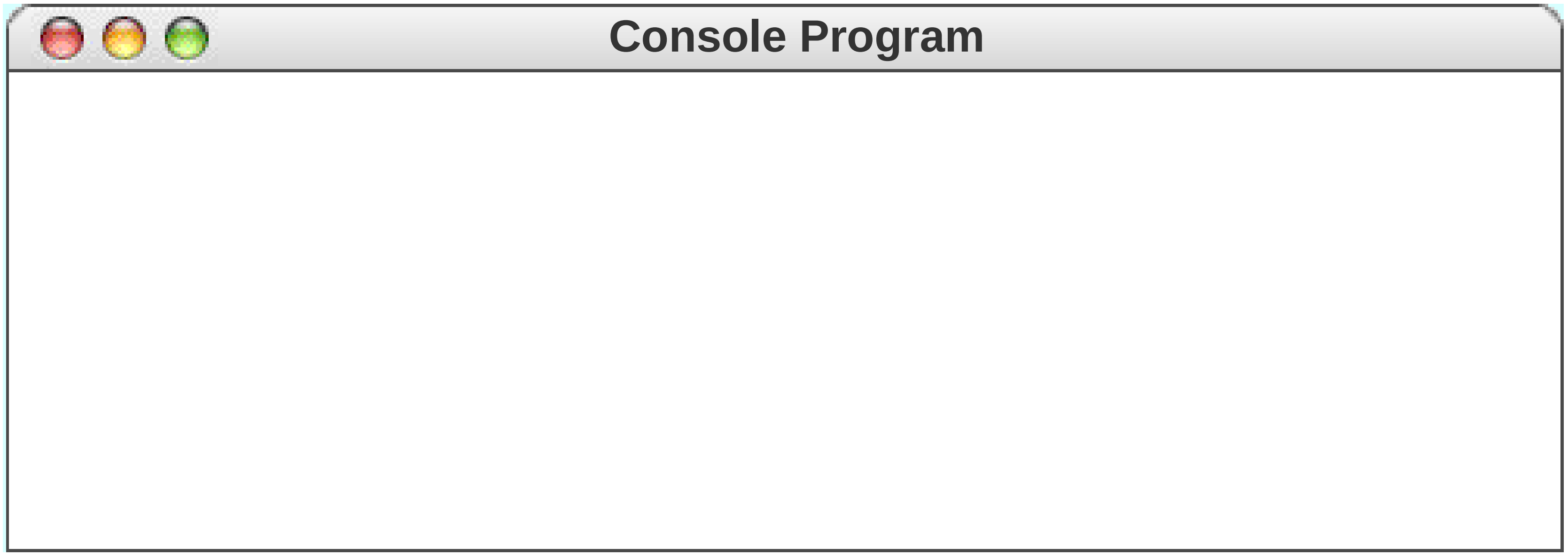
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

i $\boxed{0}$



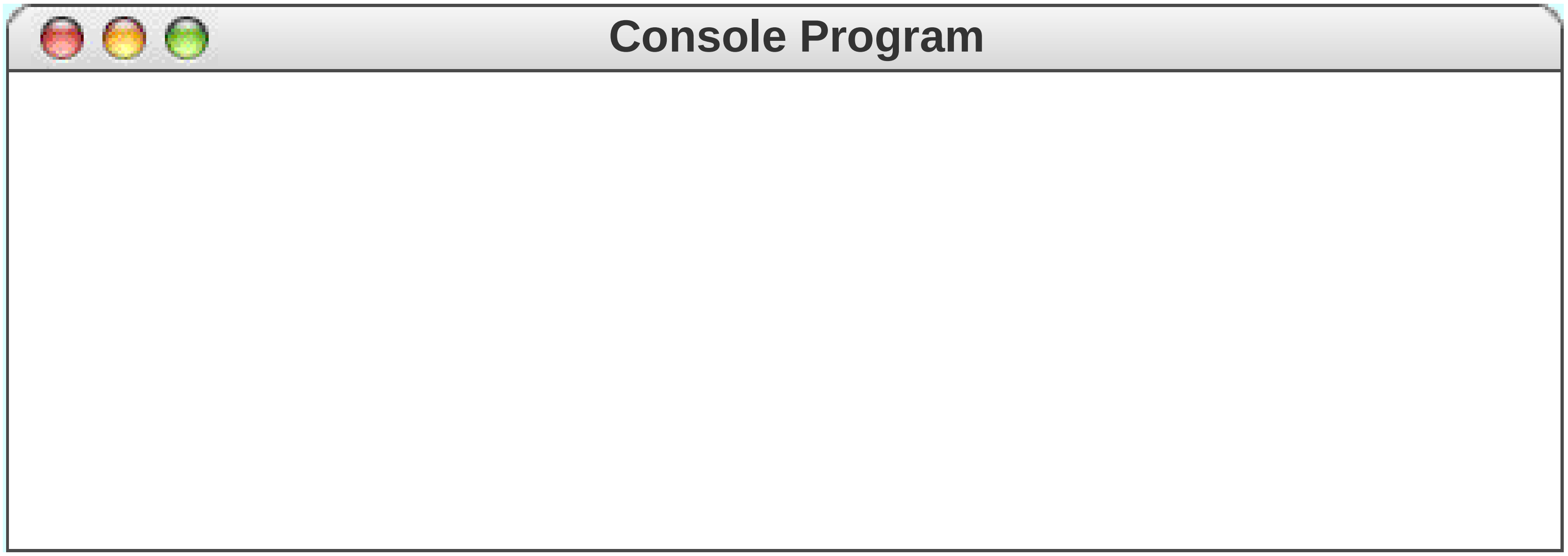
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

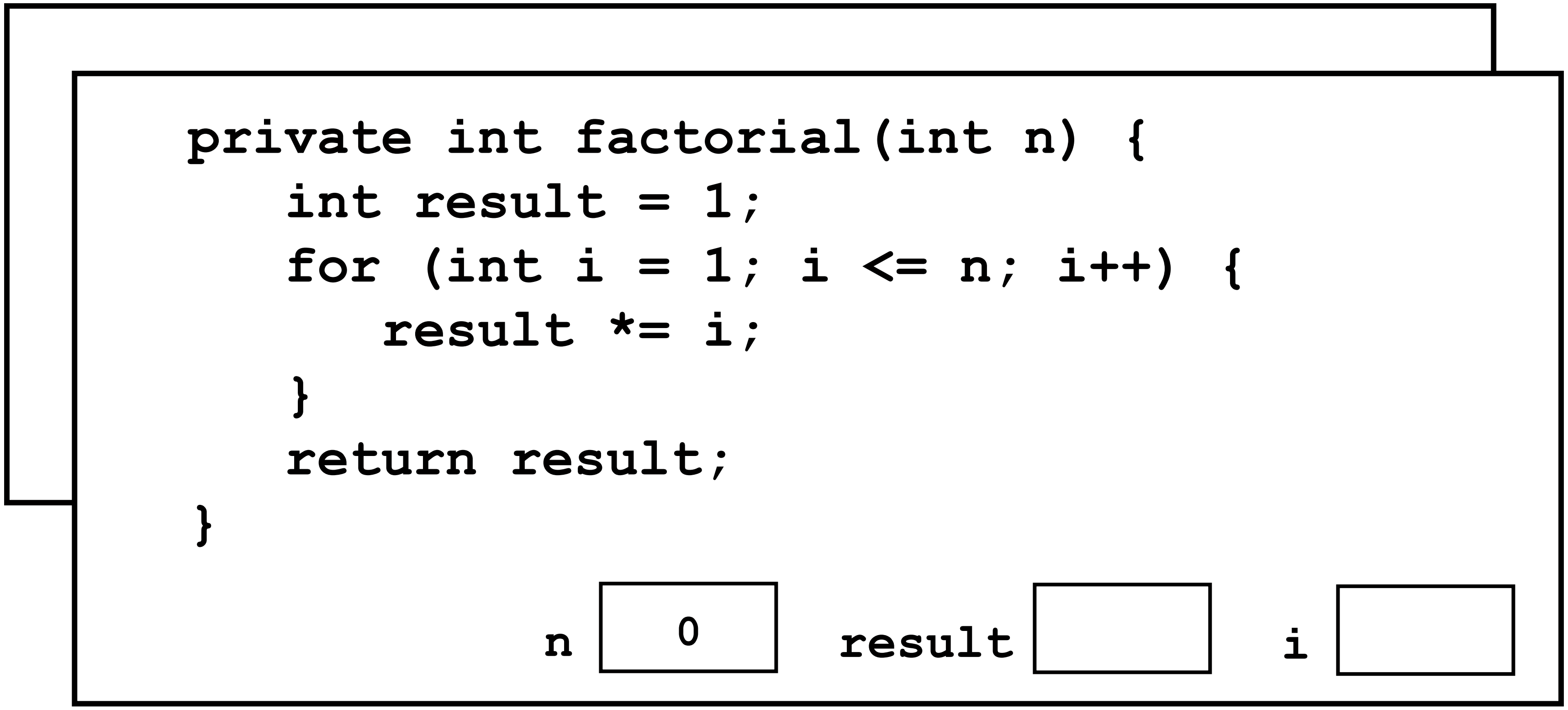
i 0



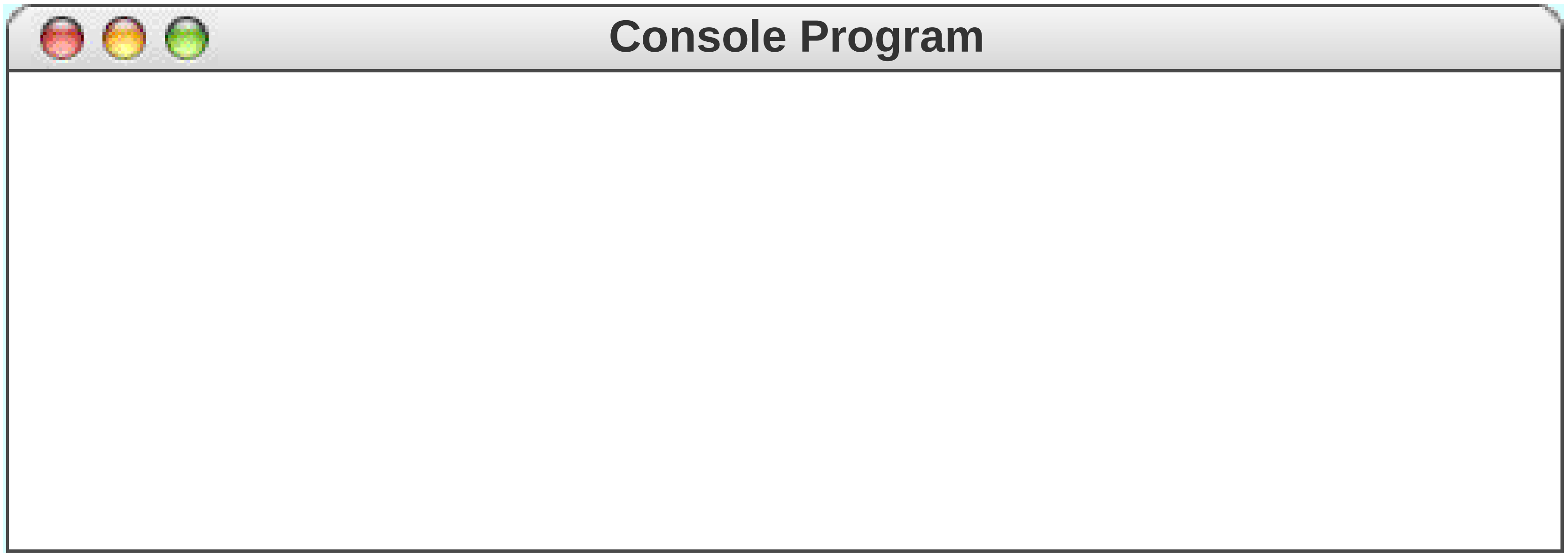
```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + $\boxed{\texttt{factorial(i)}}$);
   }
}
```

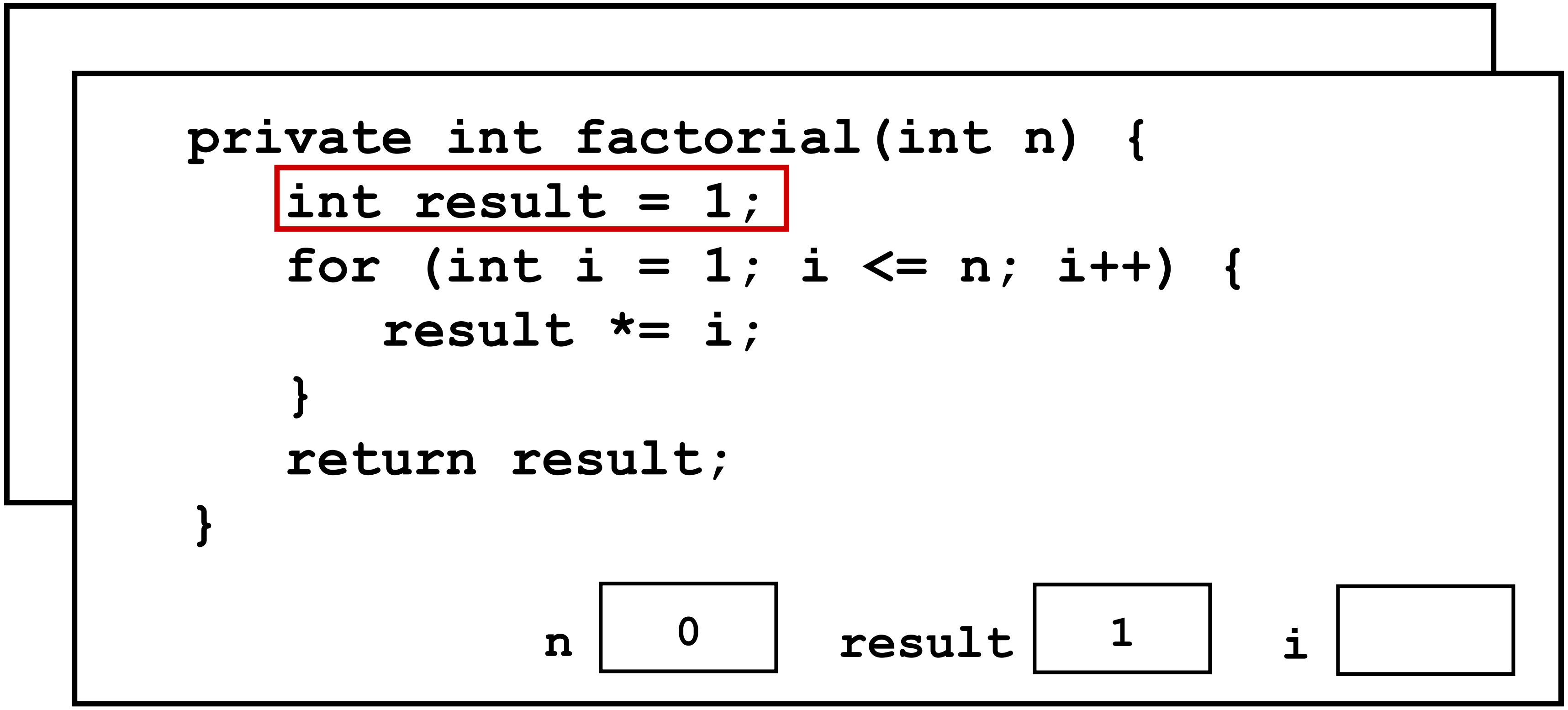
i $\boxed{0}$



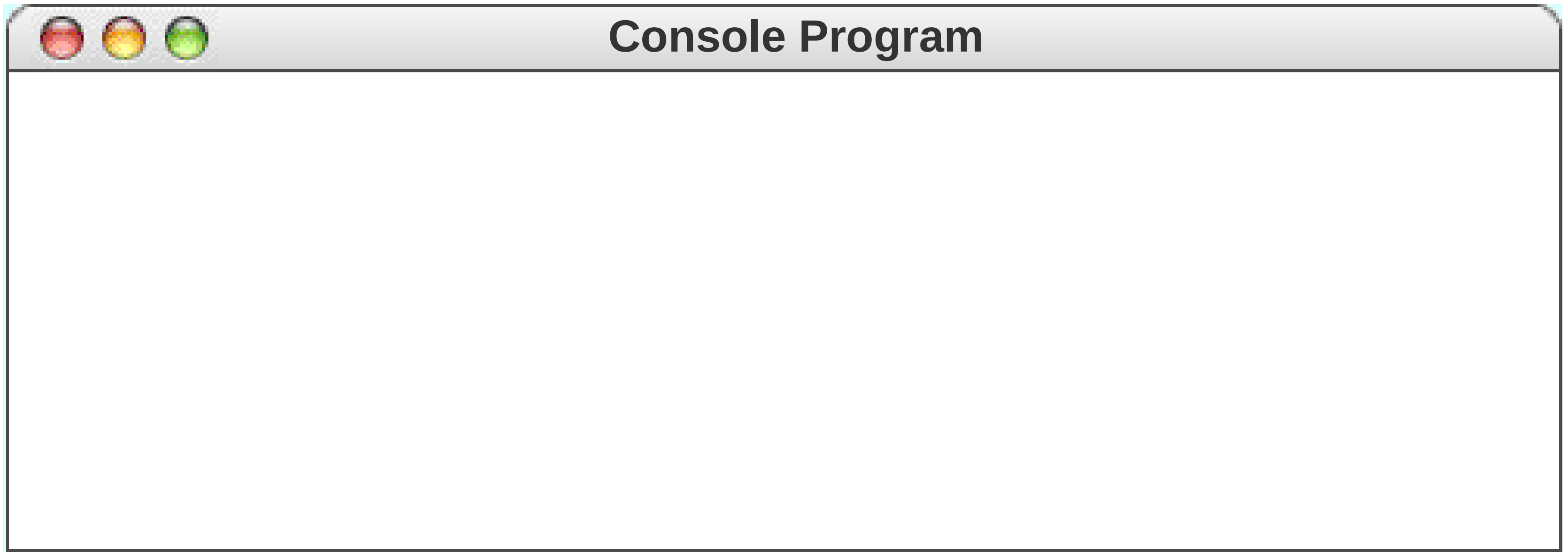


```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



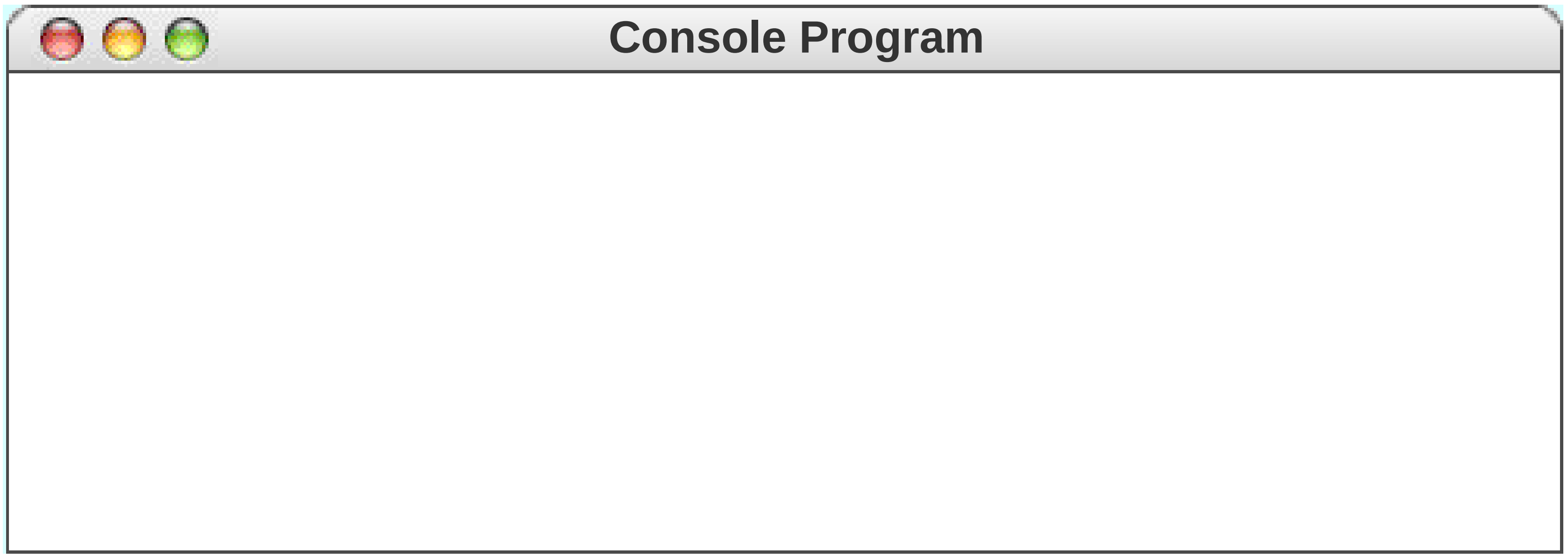


```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



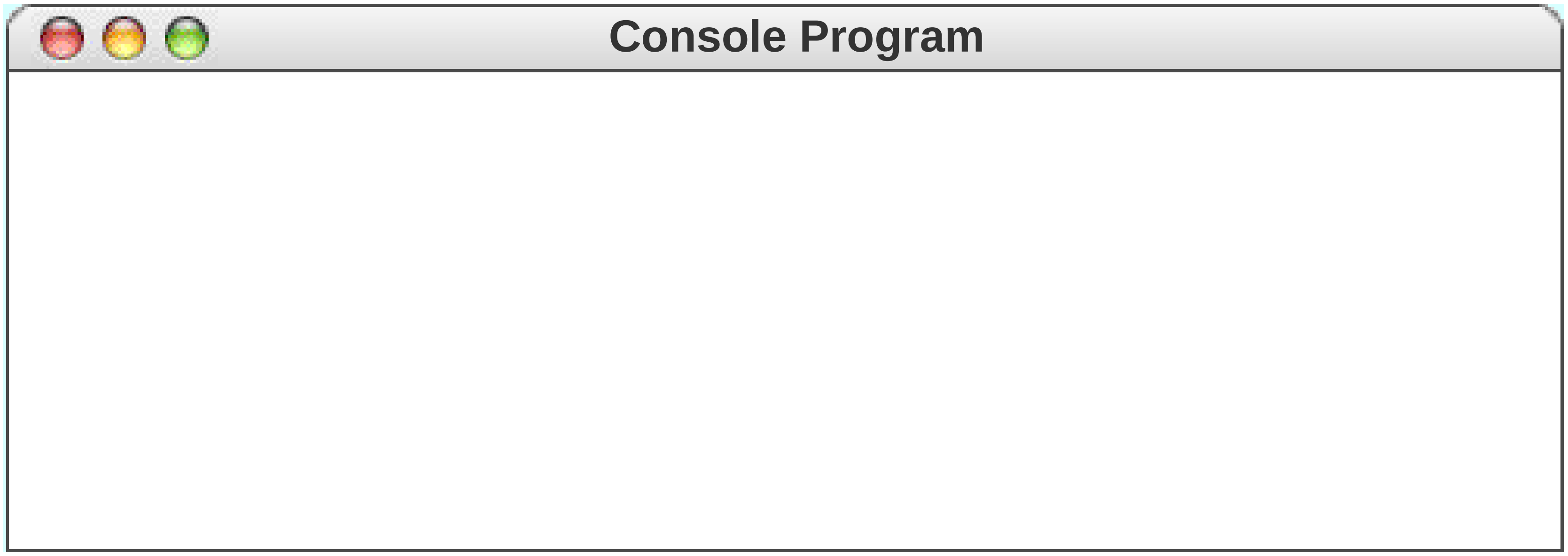
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```

```
n  0     result  1     i  1
```



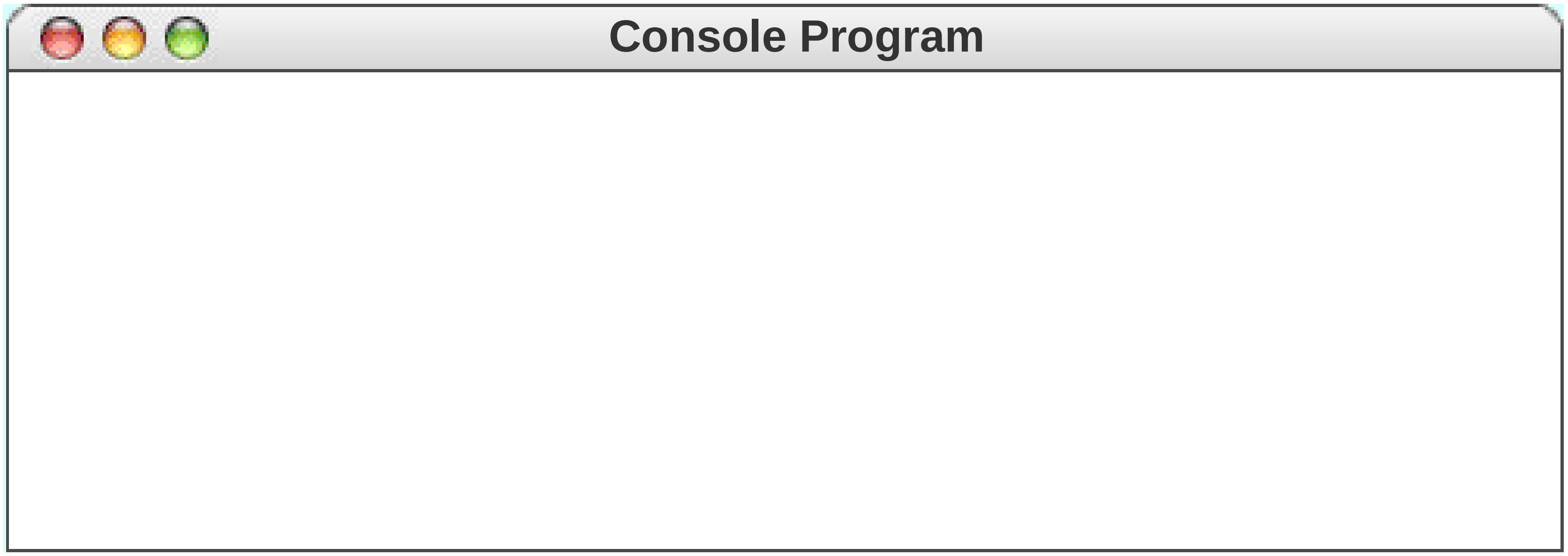
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```

n `0`   result `1`   i `1`



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

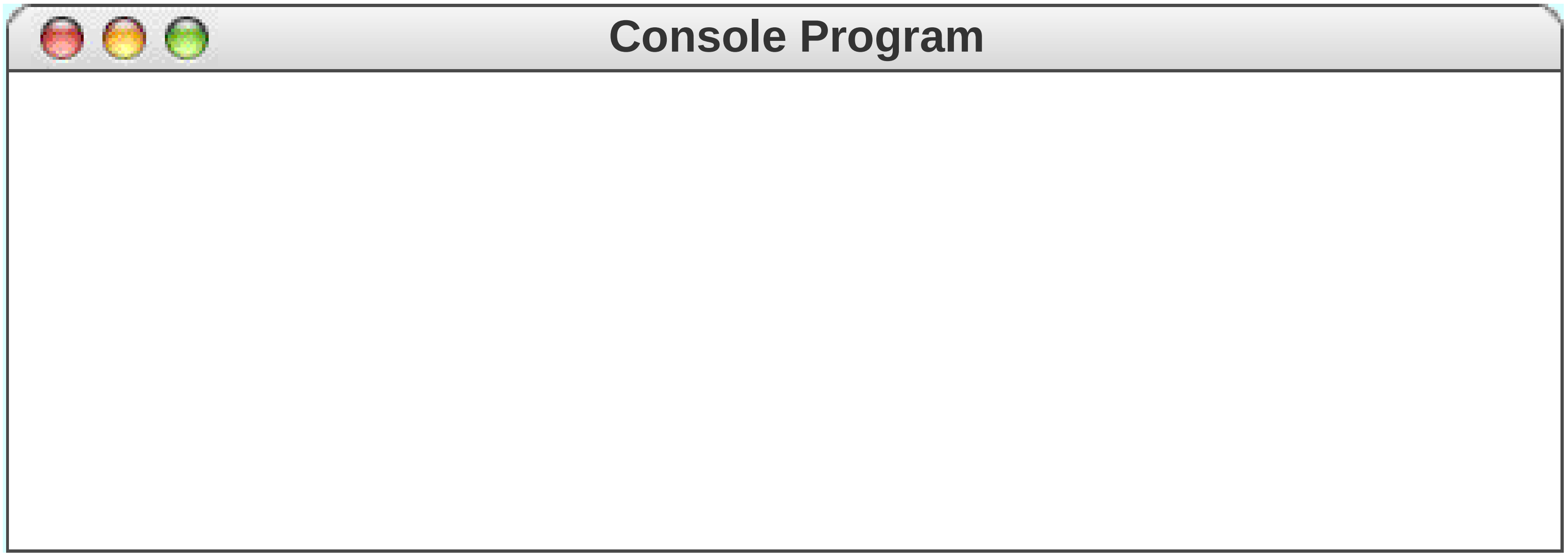
```
n   0      result   1      i   1
```

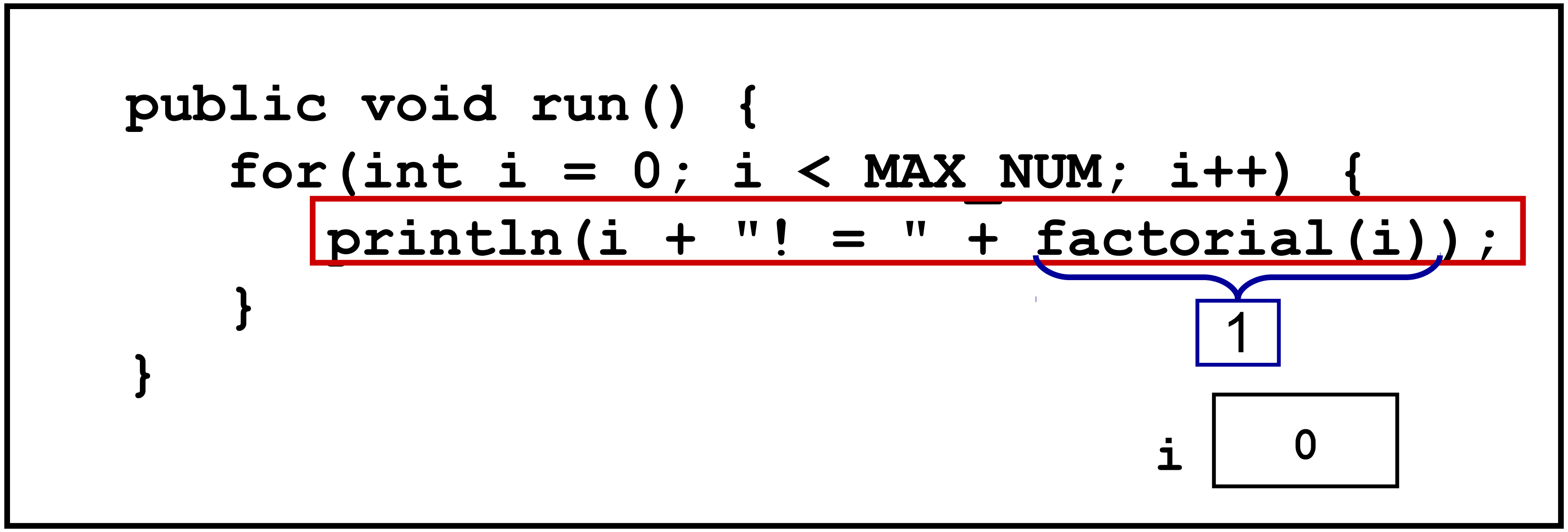


```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

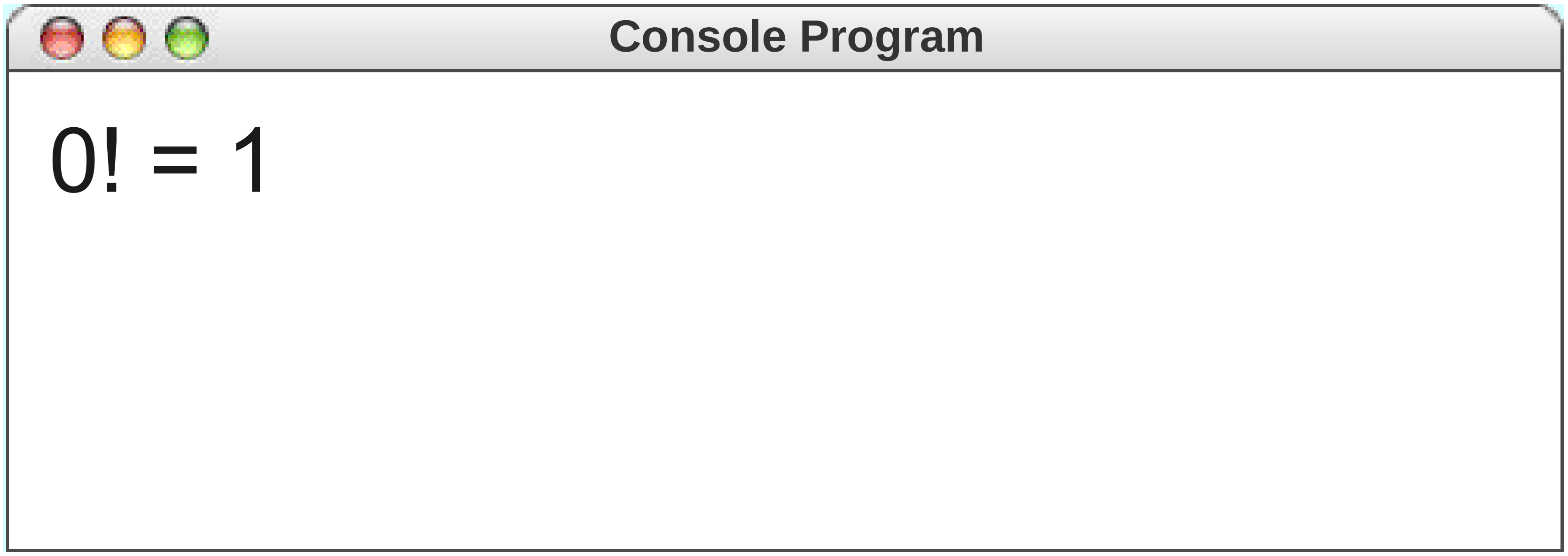
$\underbrace{\texttt{factorial(i)}}_{\boxed{1}}$

i $\boxed{0}$





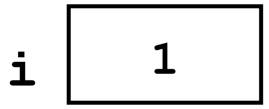
```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

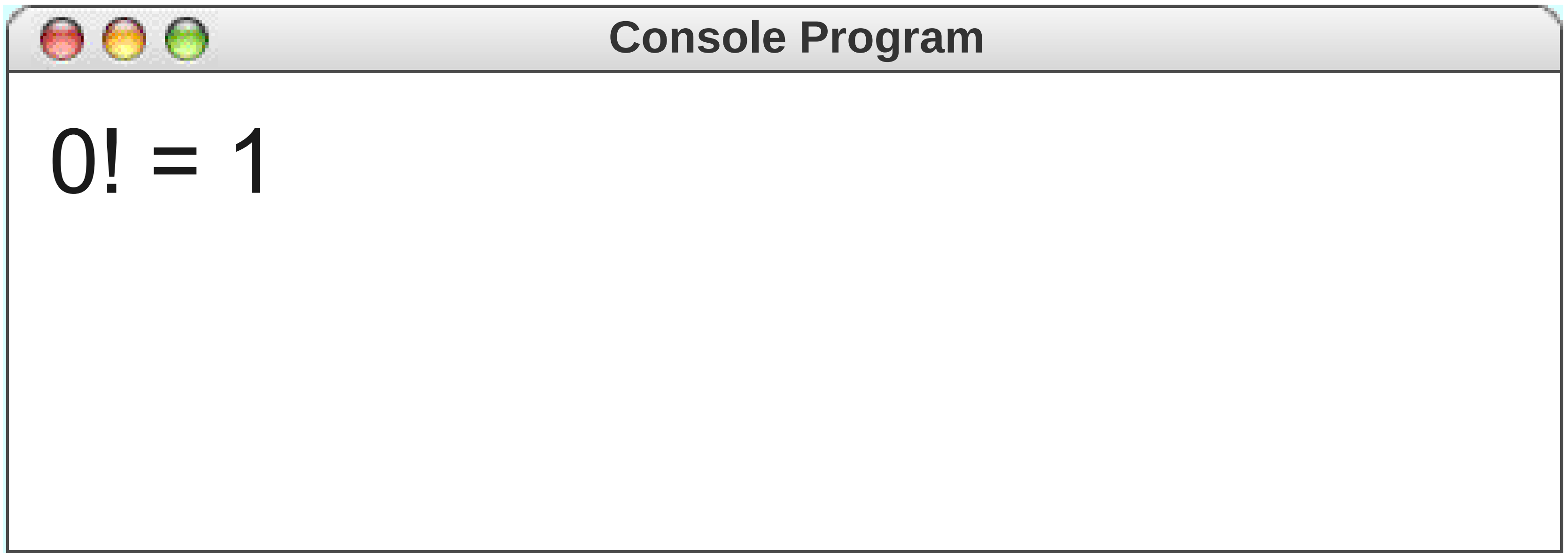


Console Program

```
0! = 1
```

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

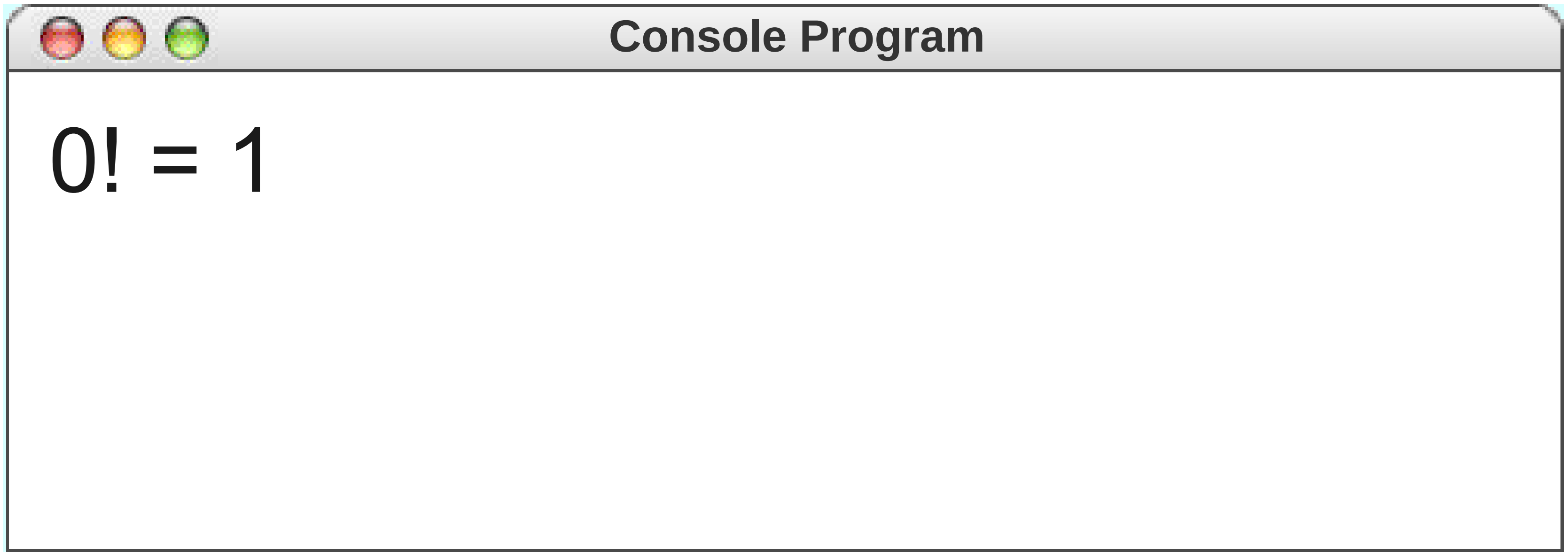




```
0! = 1
```

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

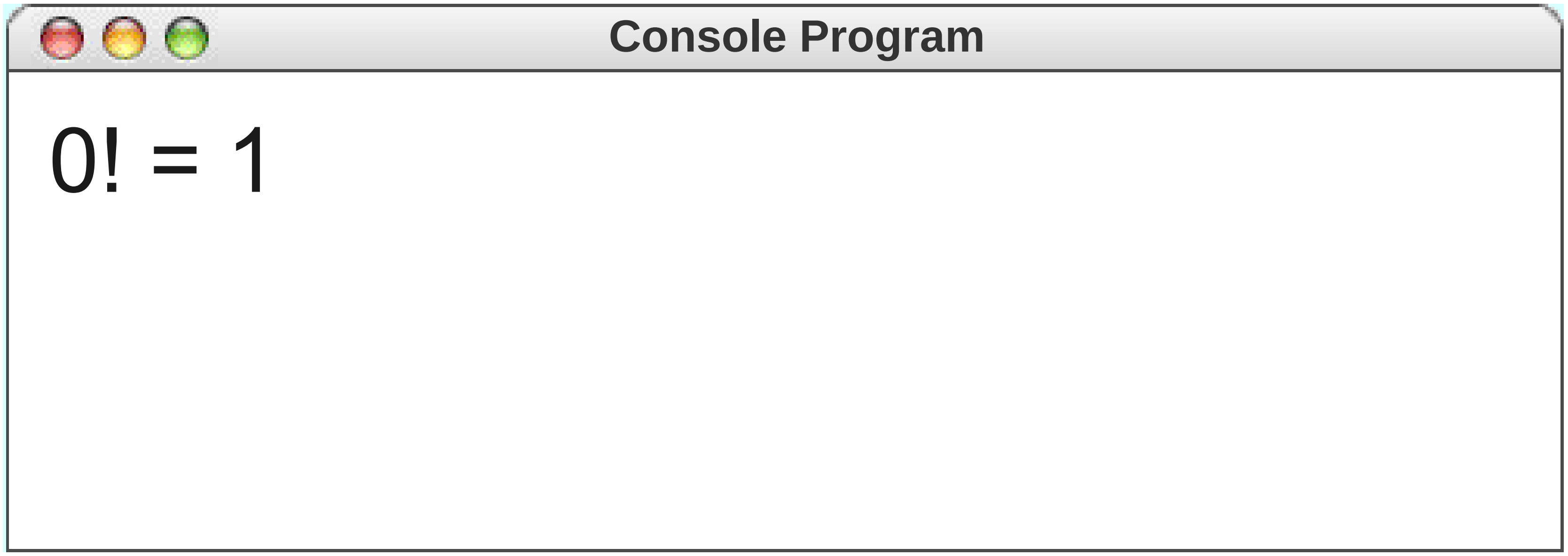
i 1



```
0! = 1
```

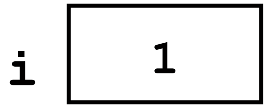
```
public void run() {
   for(int i = 0; i < MAX_NUM; i++) {
      println(i + "! = " + factorial(i));
   }
}
```

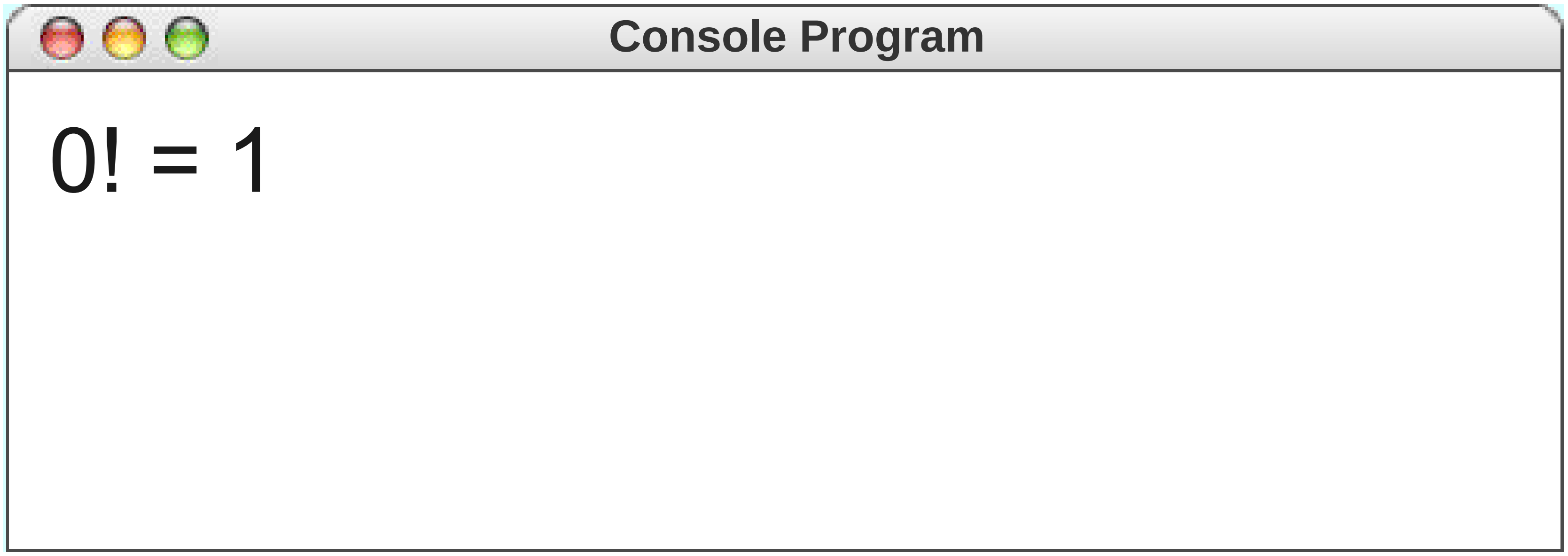
```
i  1
```



```
0! = 1
```

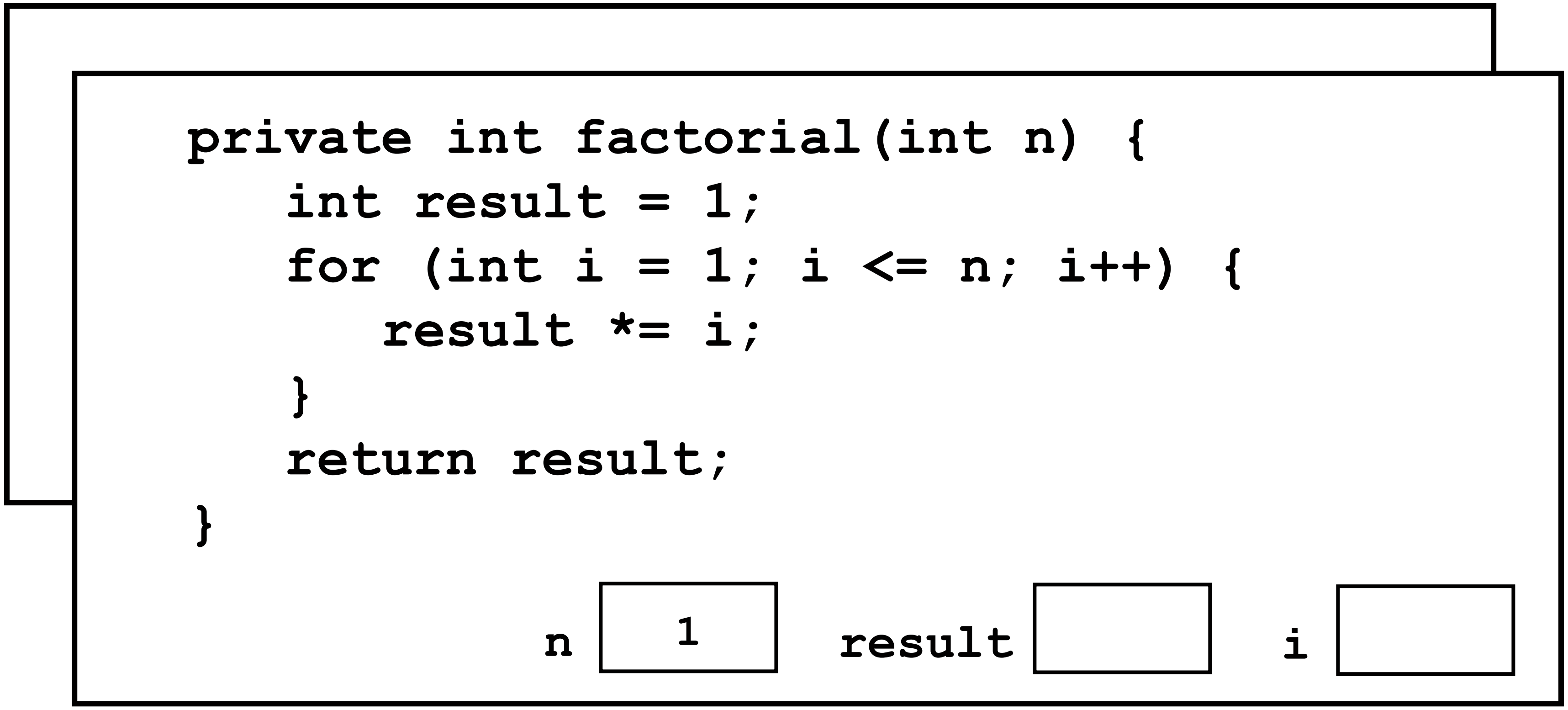
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

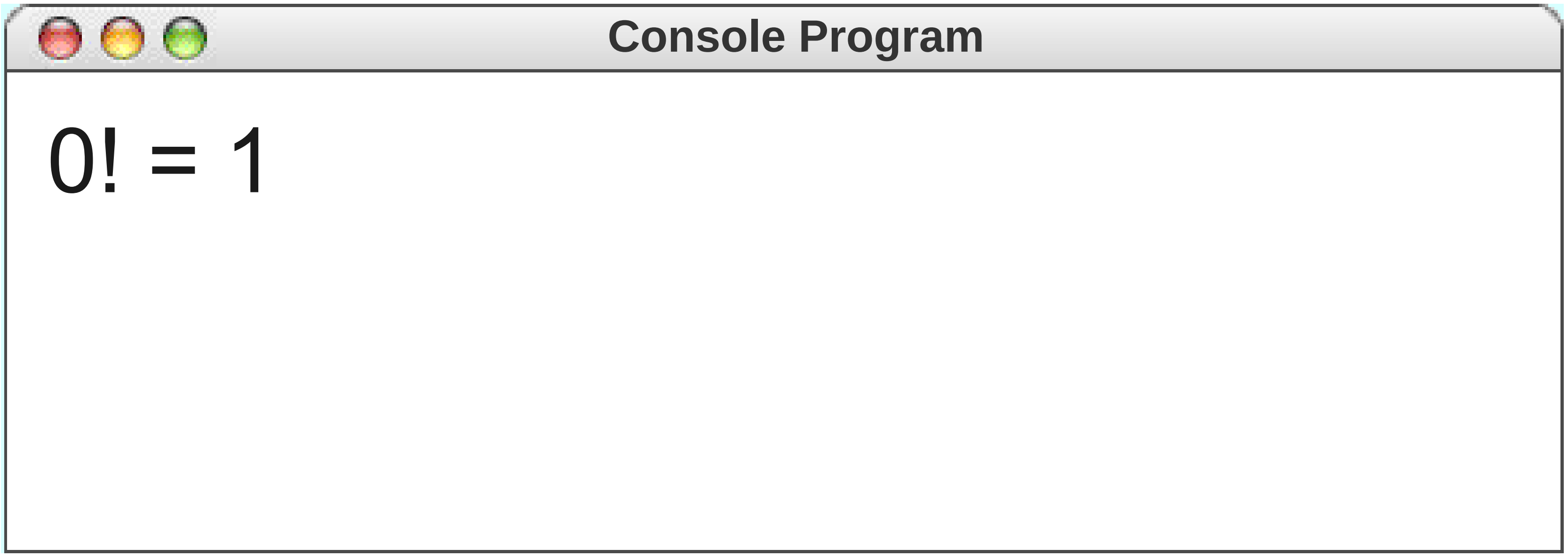




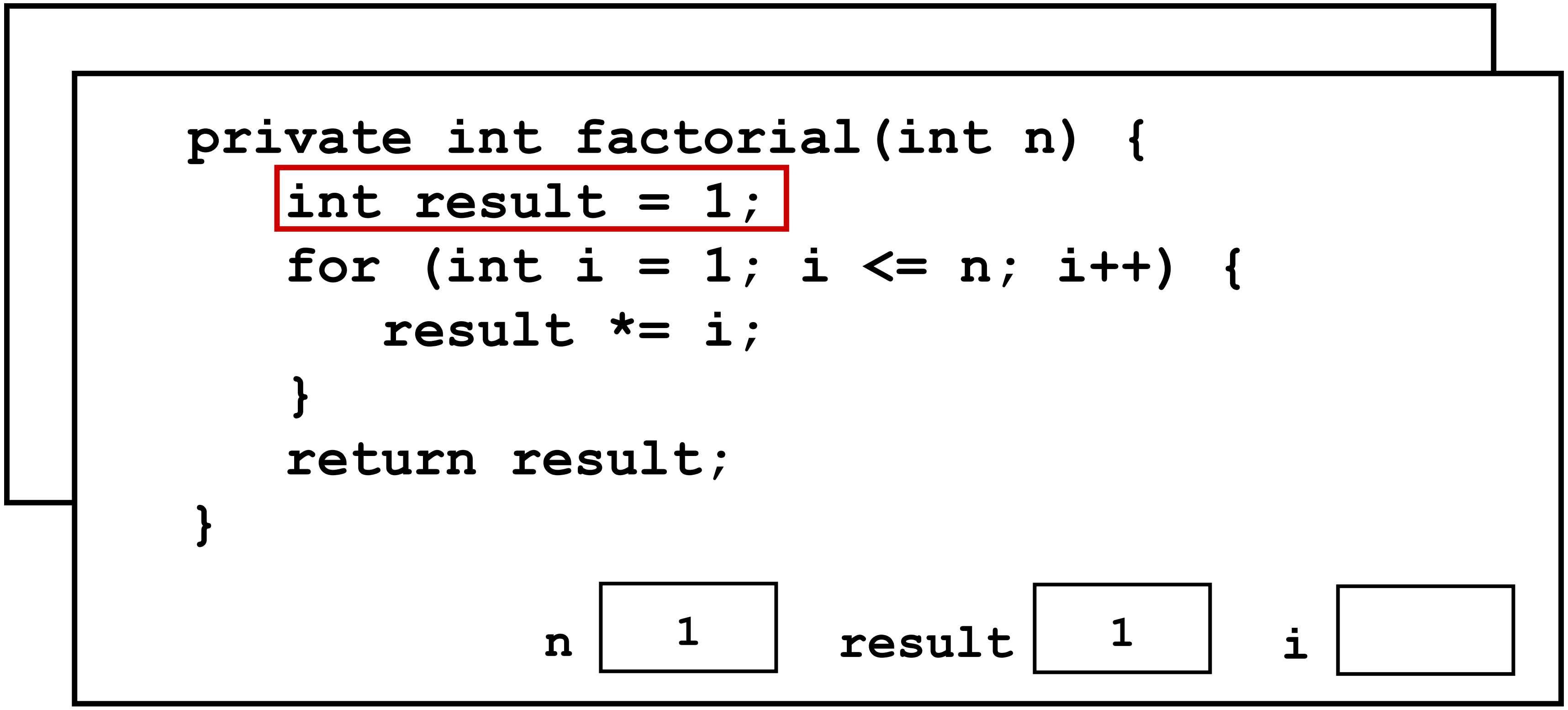
```
0! = 1
```

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

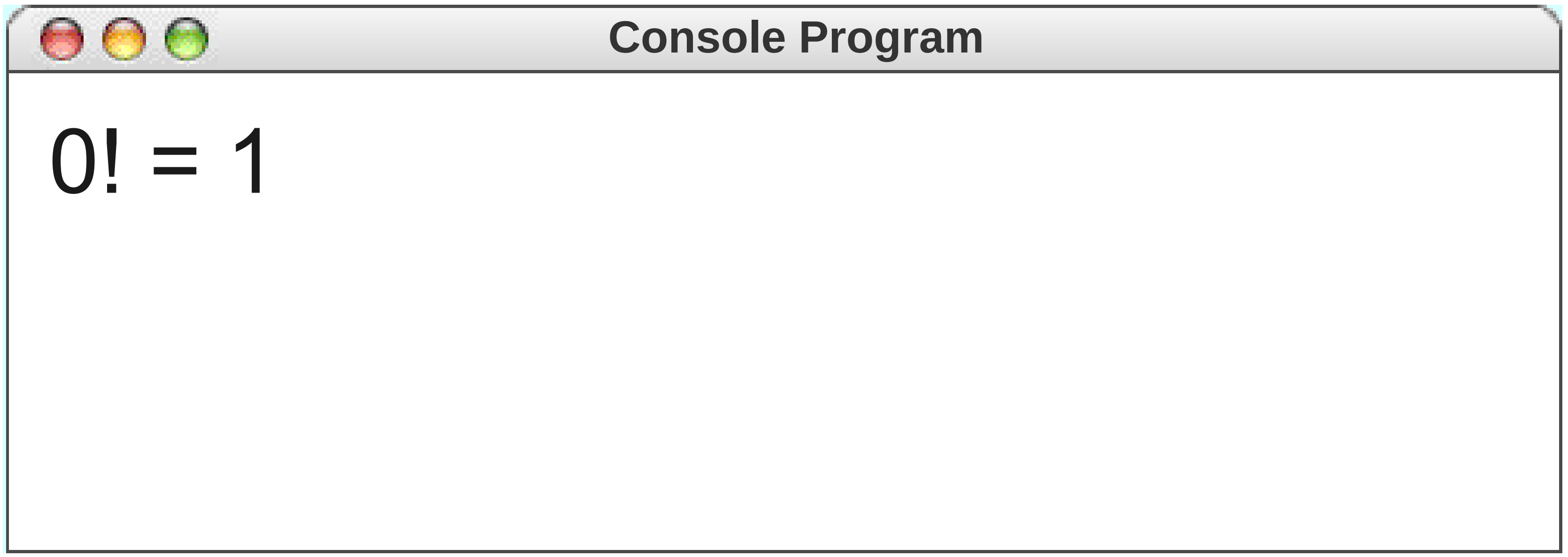




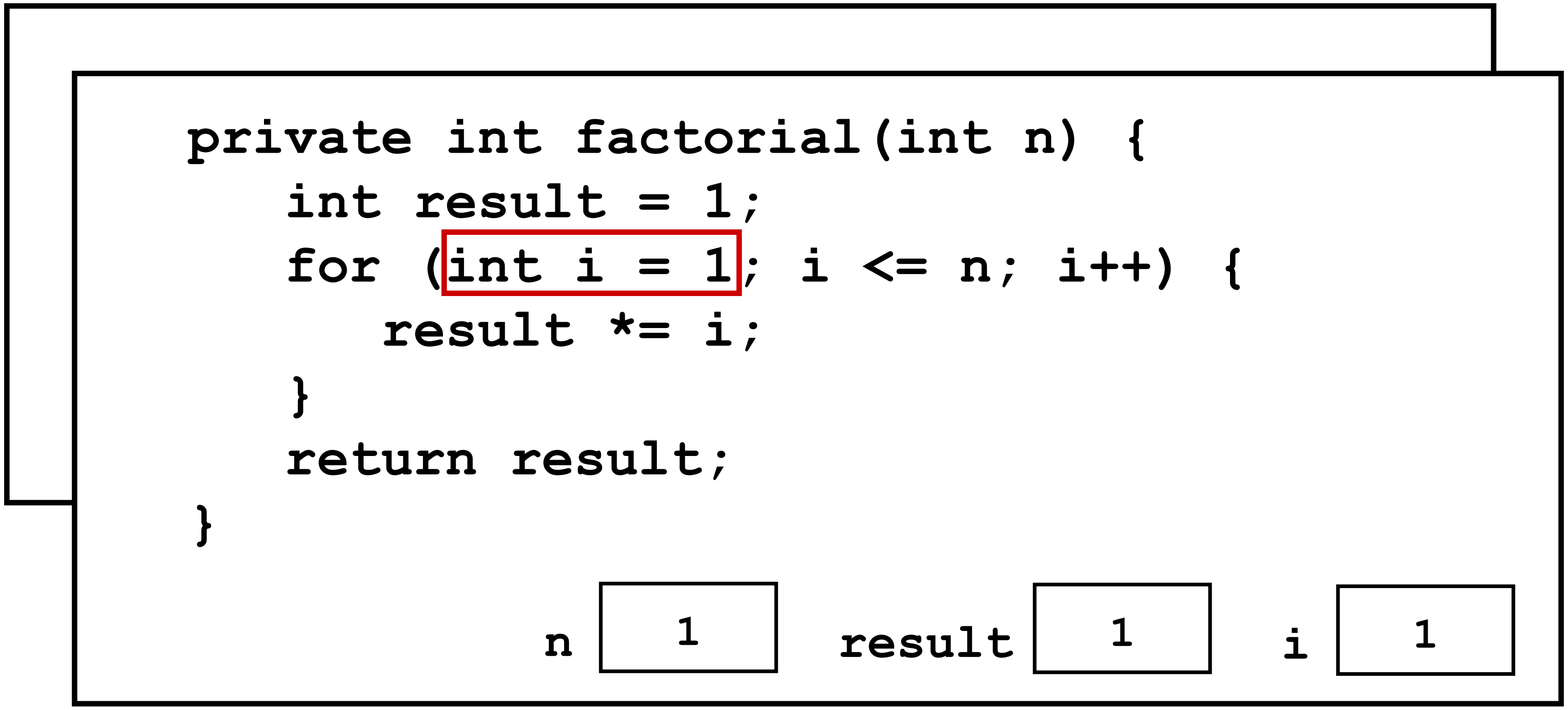
```
0! = 1
```



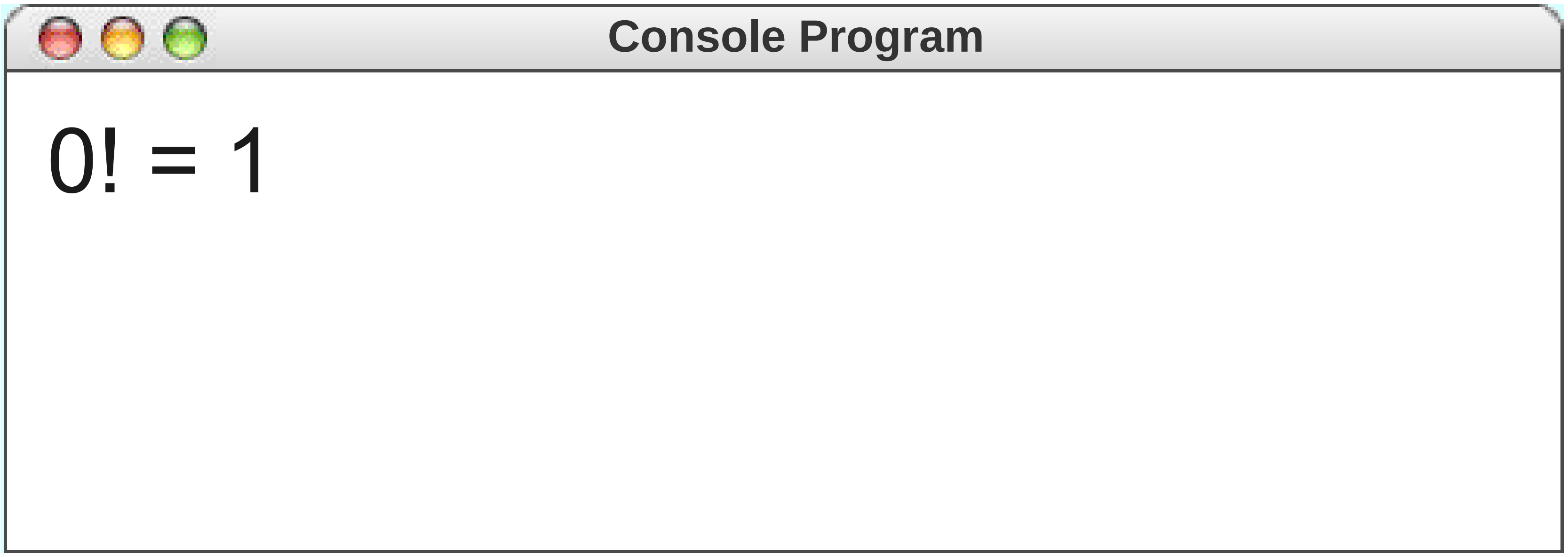
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



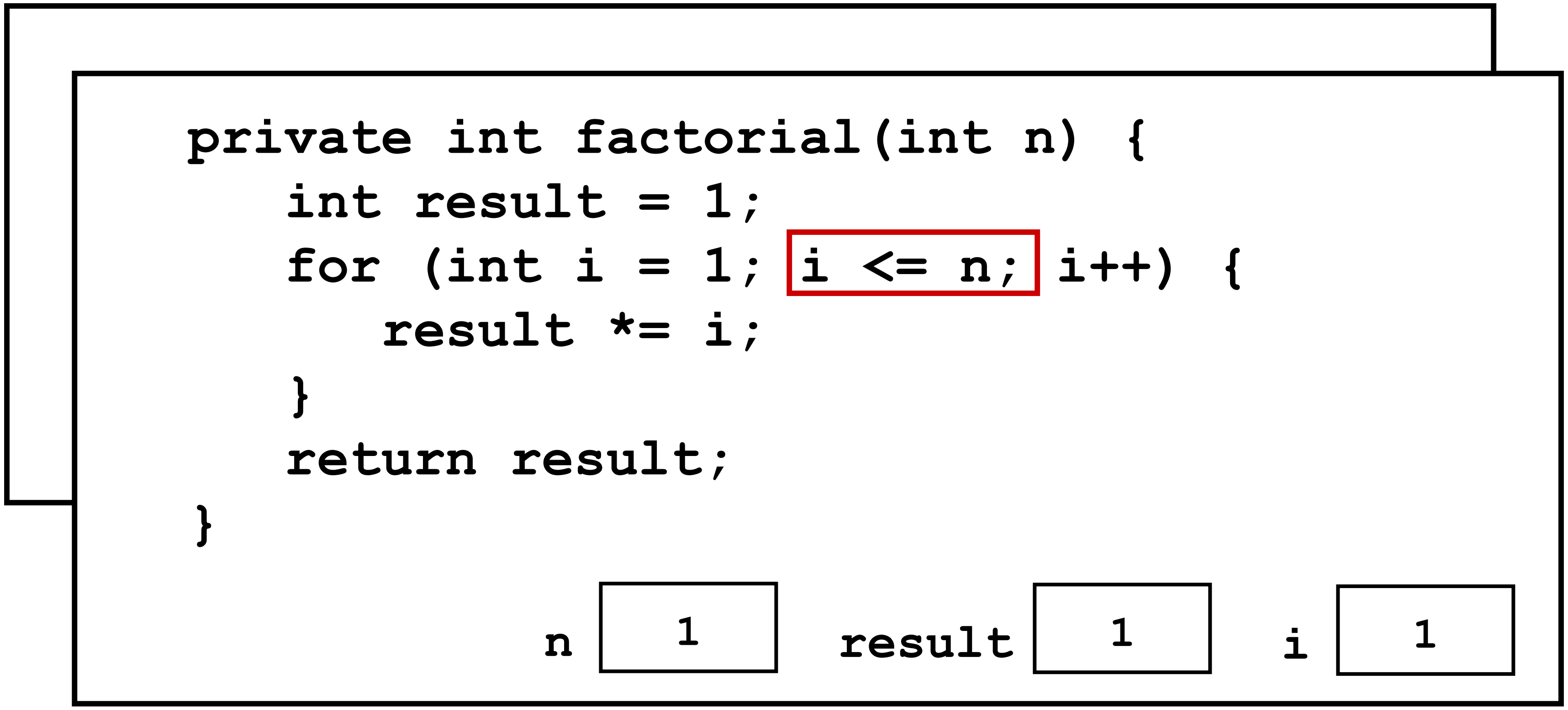
```
0! = 1
```



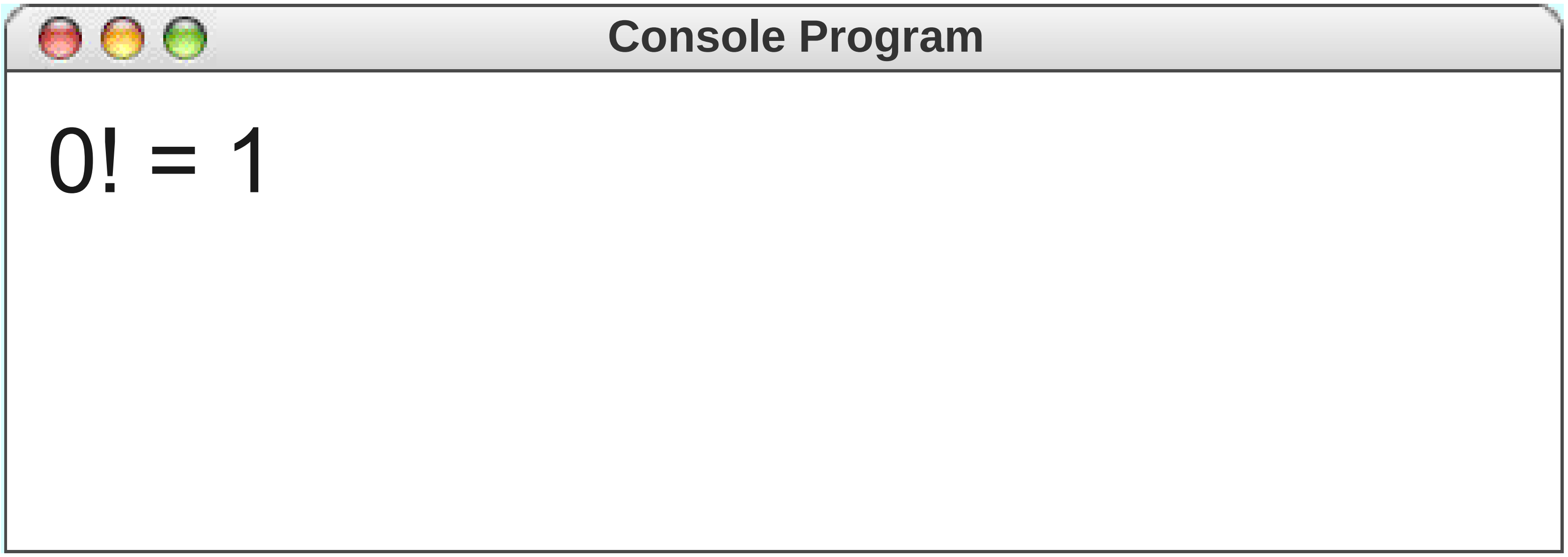
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



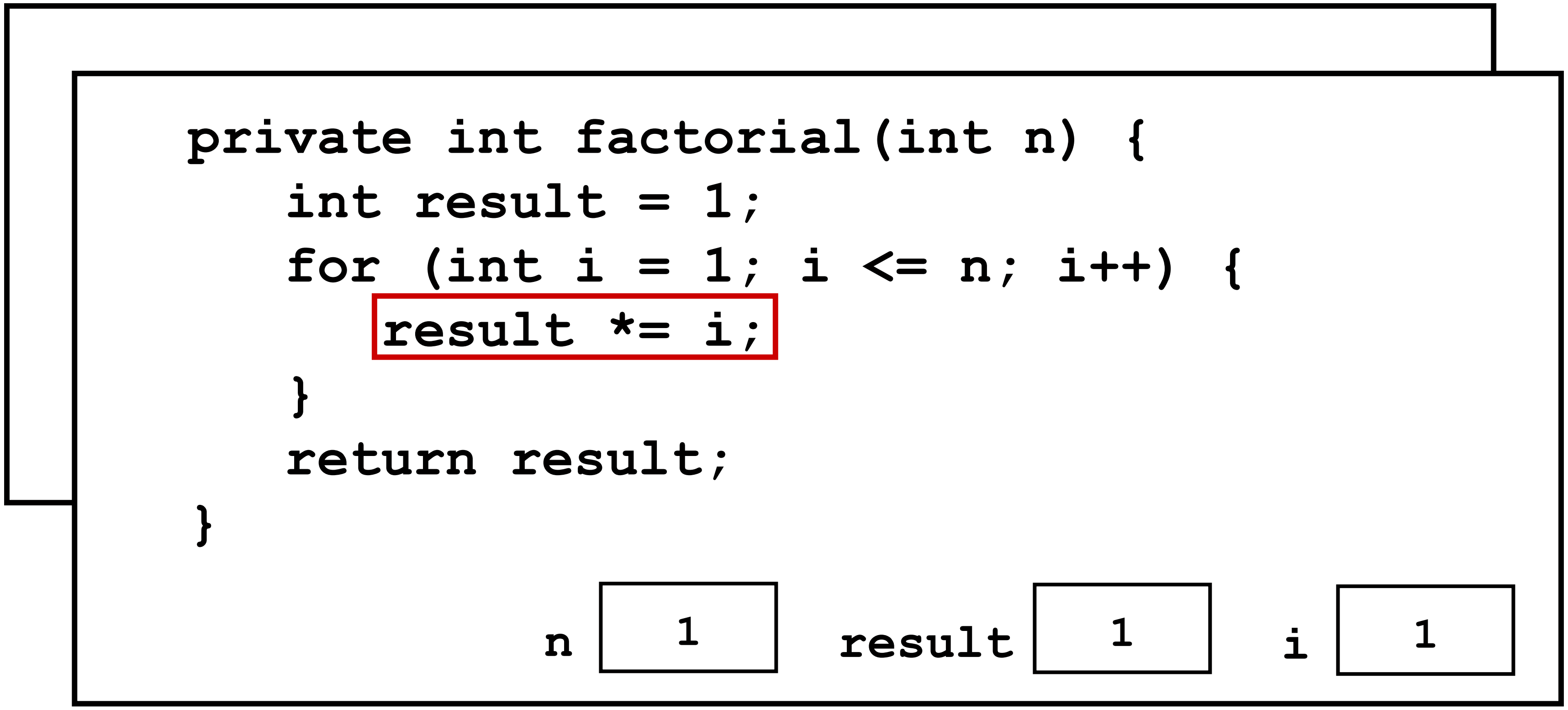
```
0! = 1
```



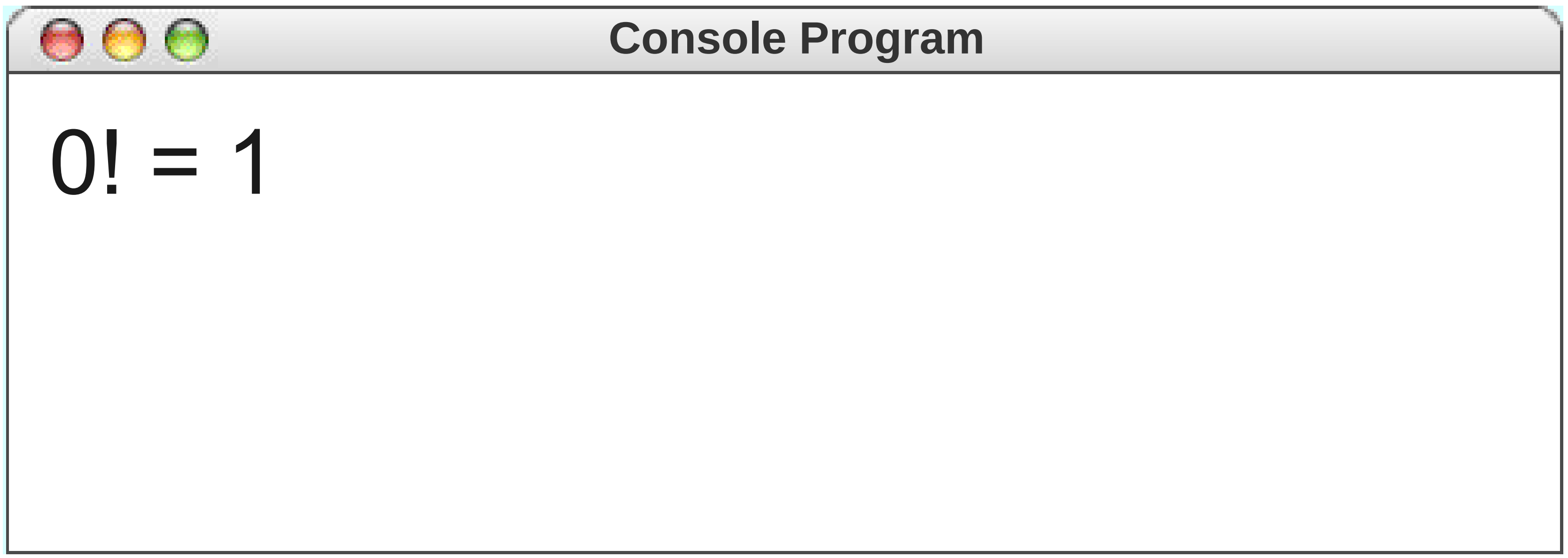
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



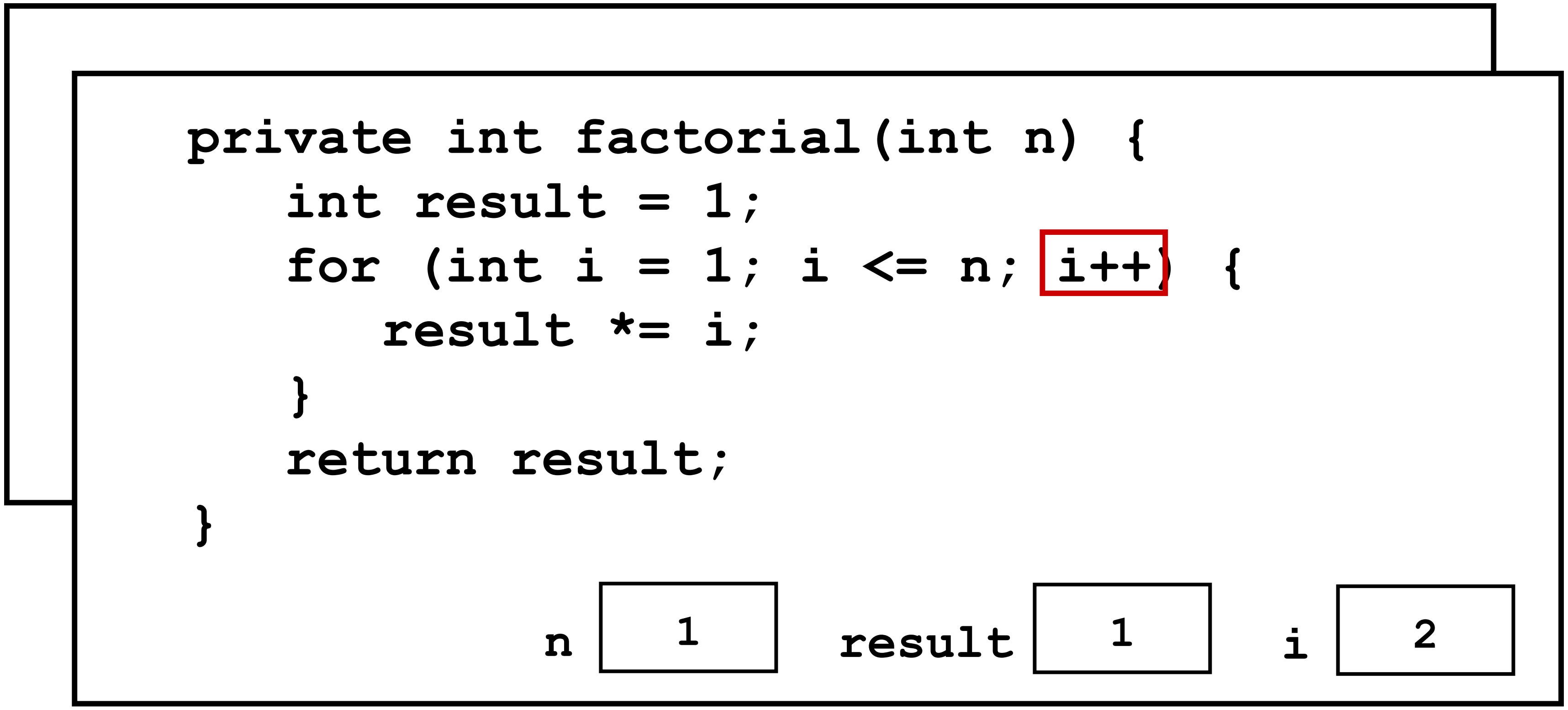
```
0! = 1
```



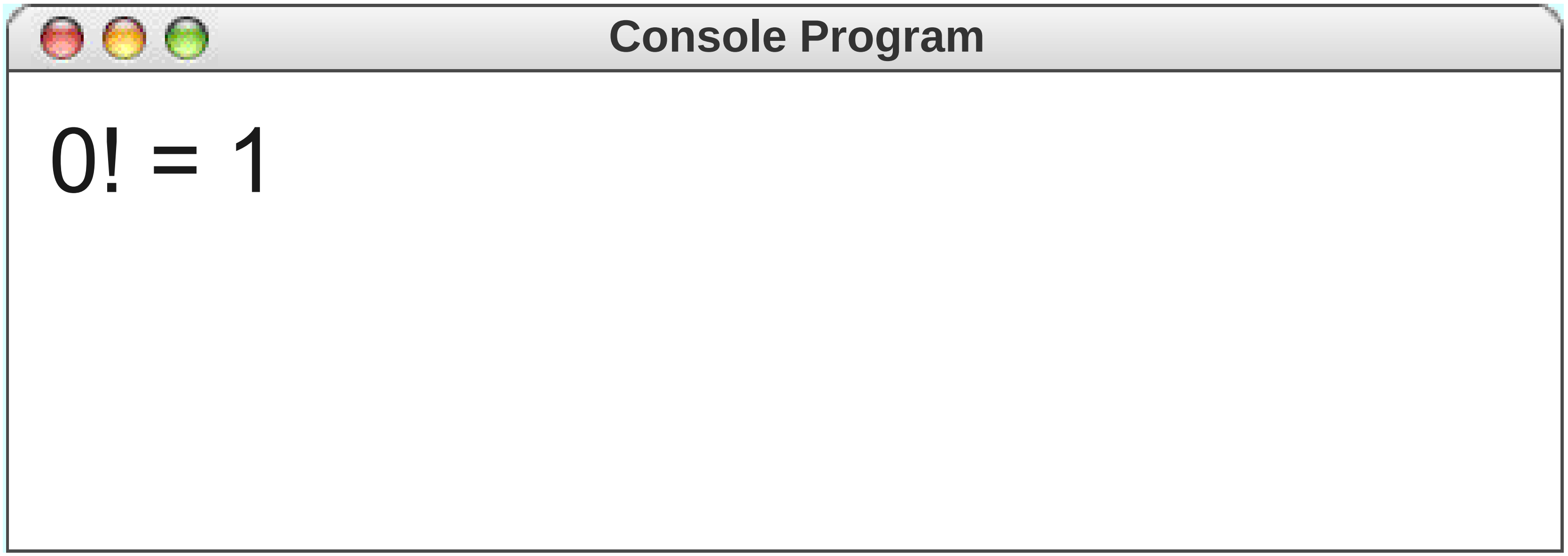
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



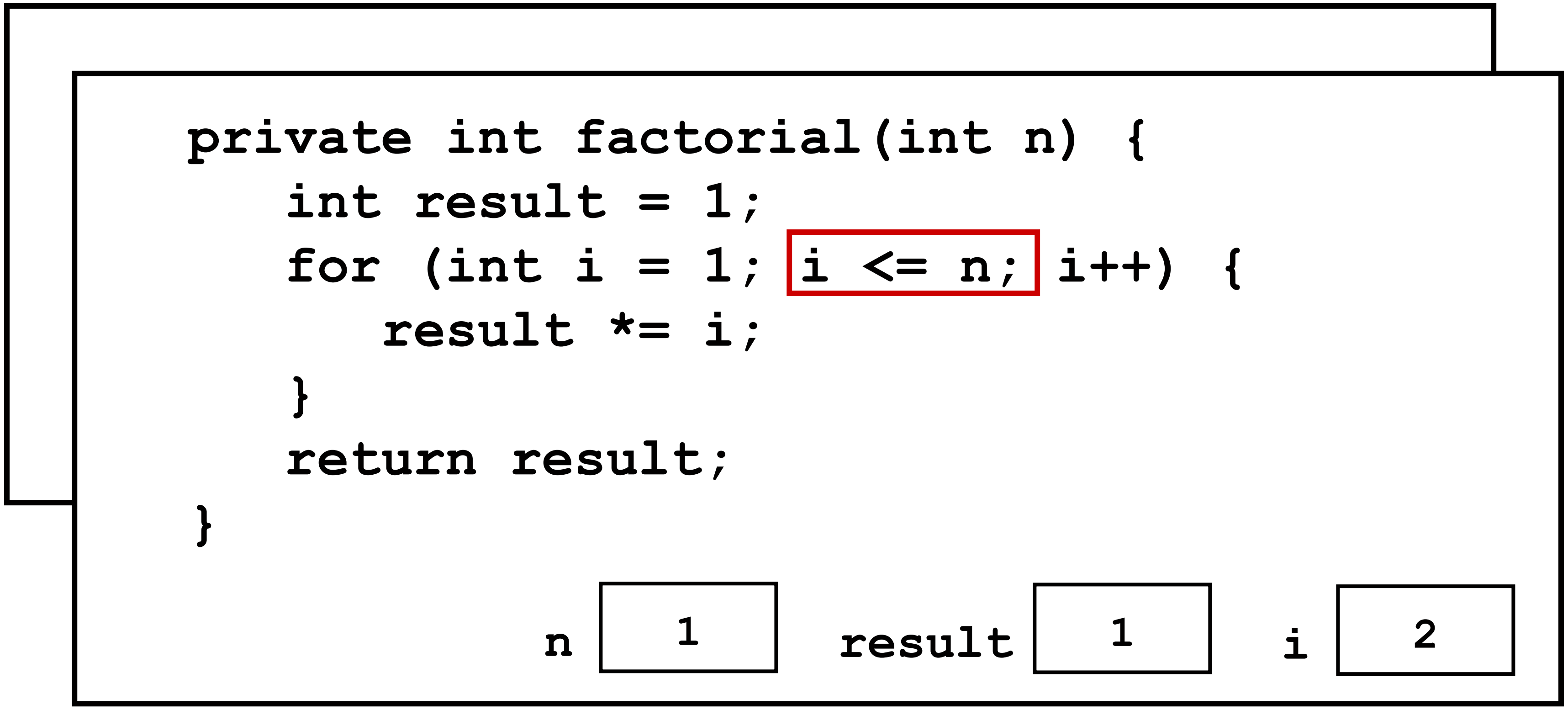
```
0! = 1
```



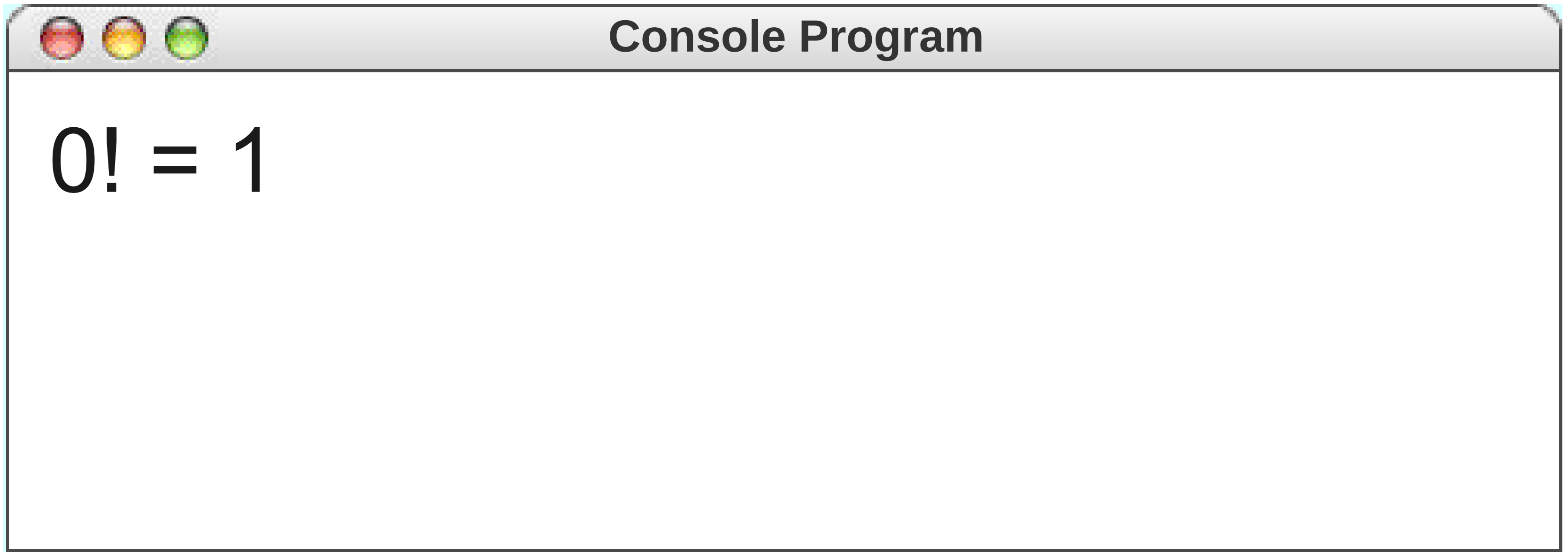
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



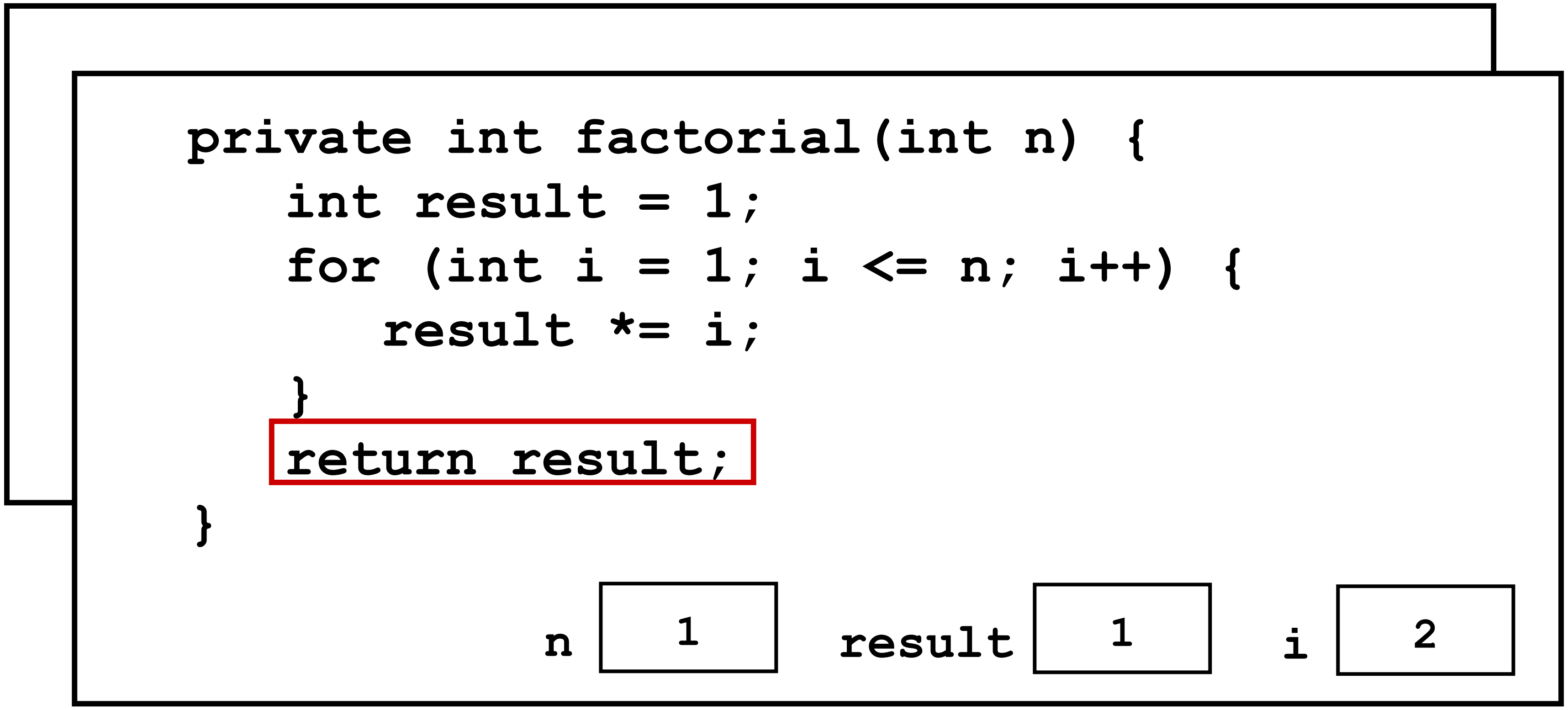
```
0! = 1
```



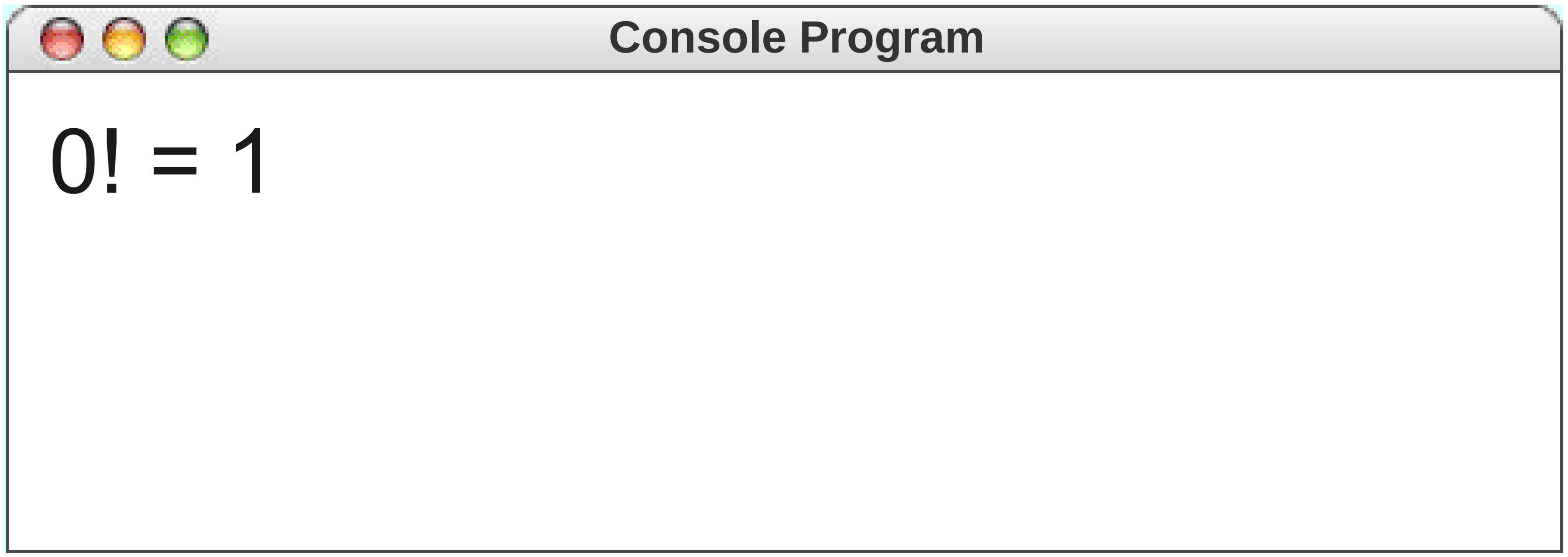
```
private int factorial(int n) {
   int result = 1;
   for (int i = 1; i <= n; i++) {
      result *= i;
   }
   return result;
}
```



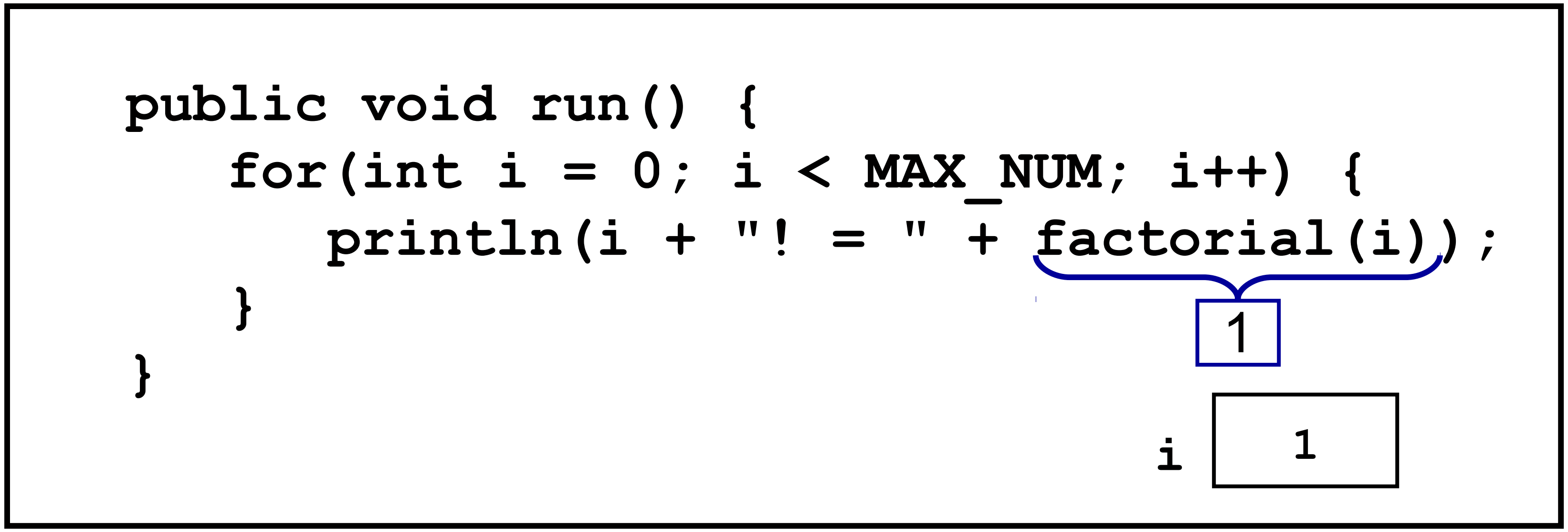
```
0! = 1
```



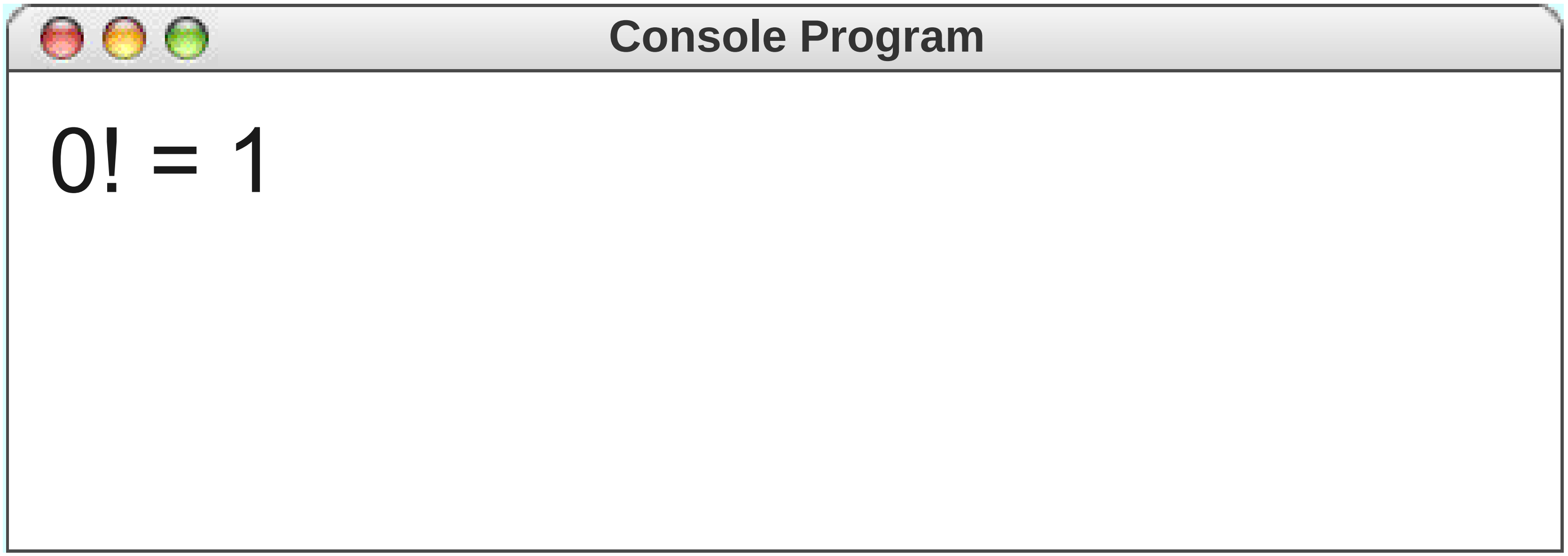
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```



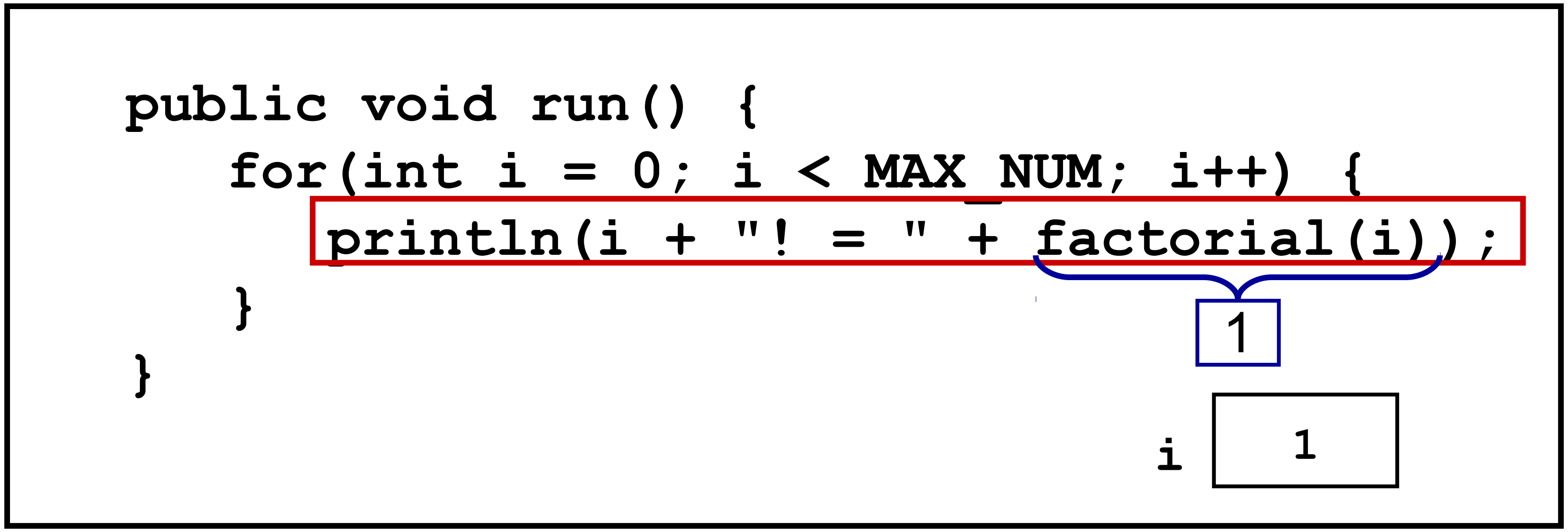
```
0! = 1
```



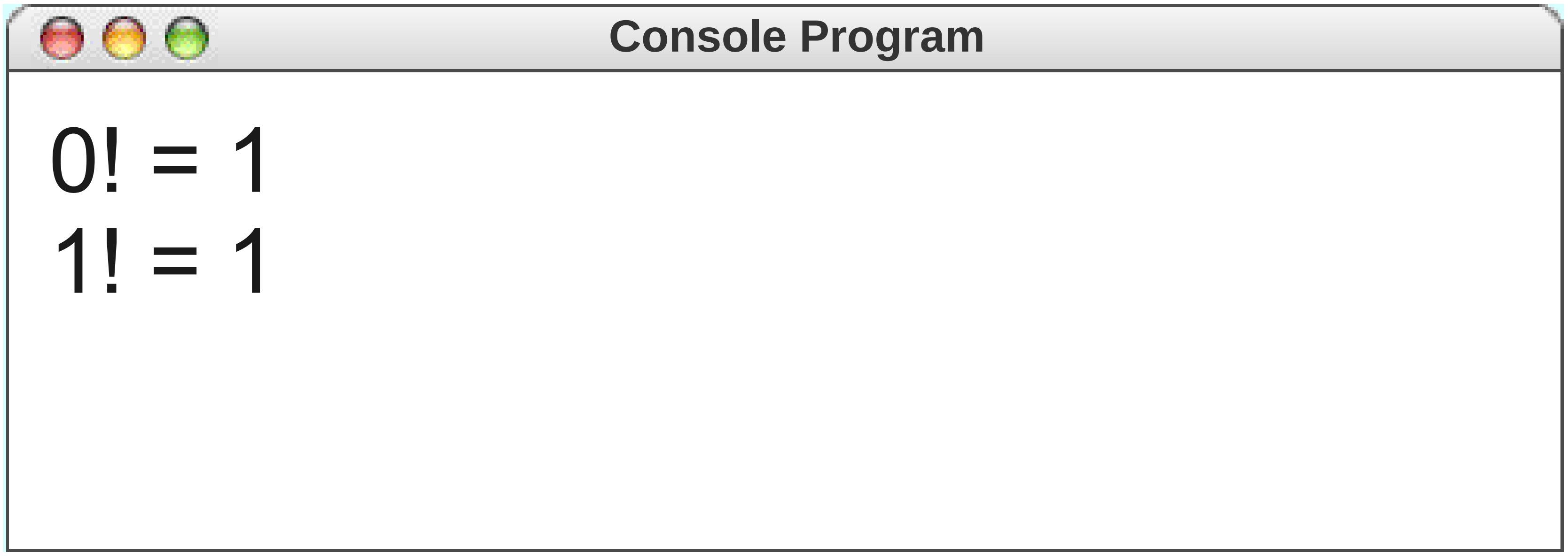
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```



```
0! = 1
```

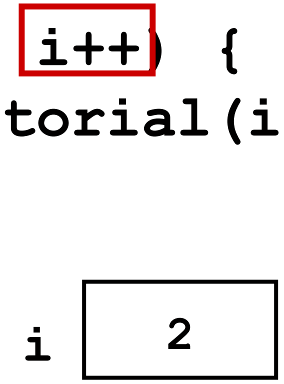


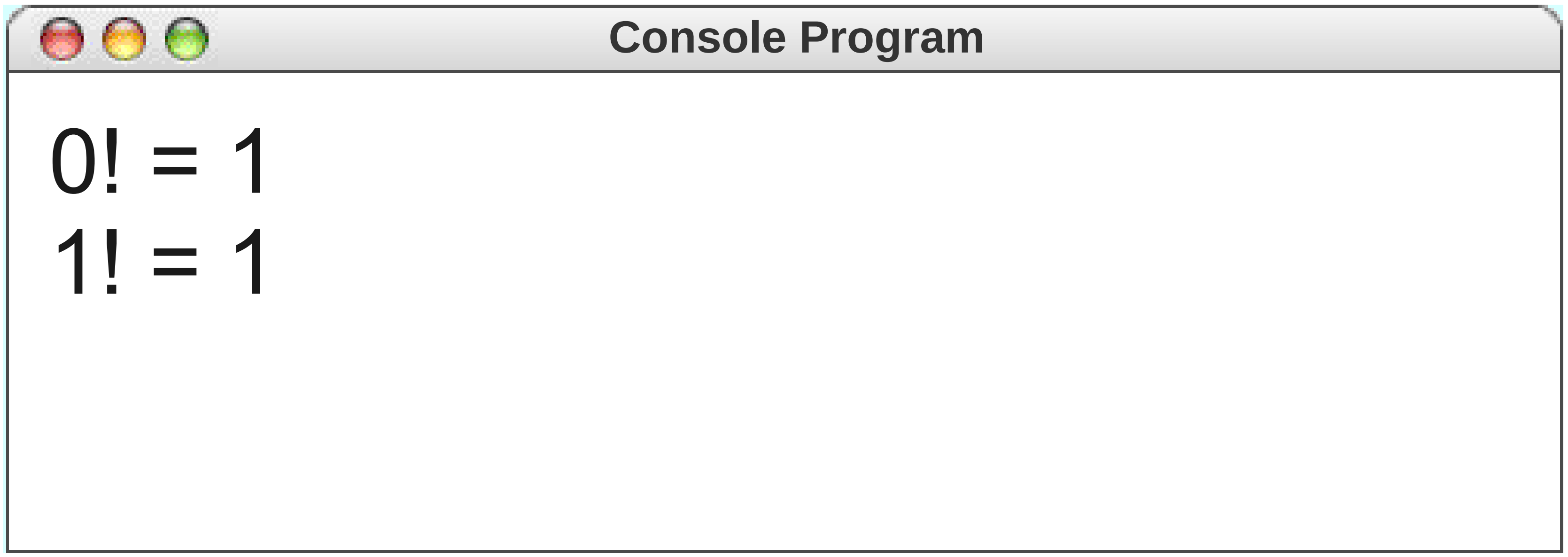
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```



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

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

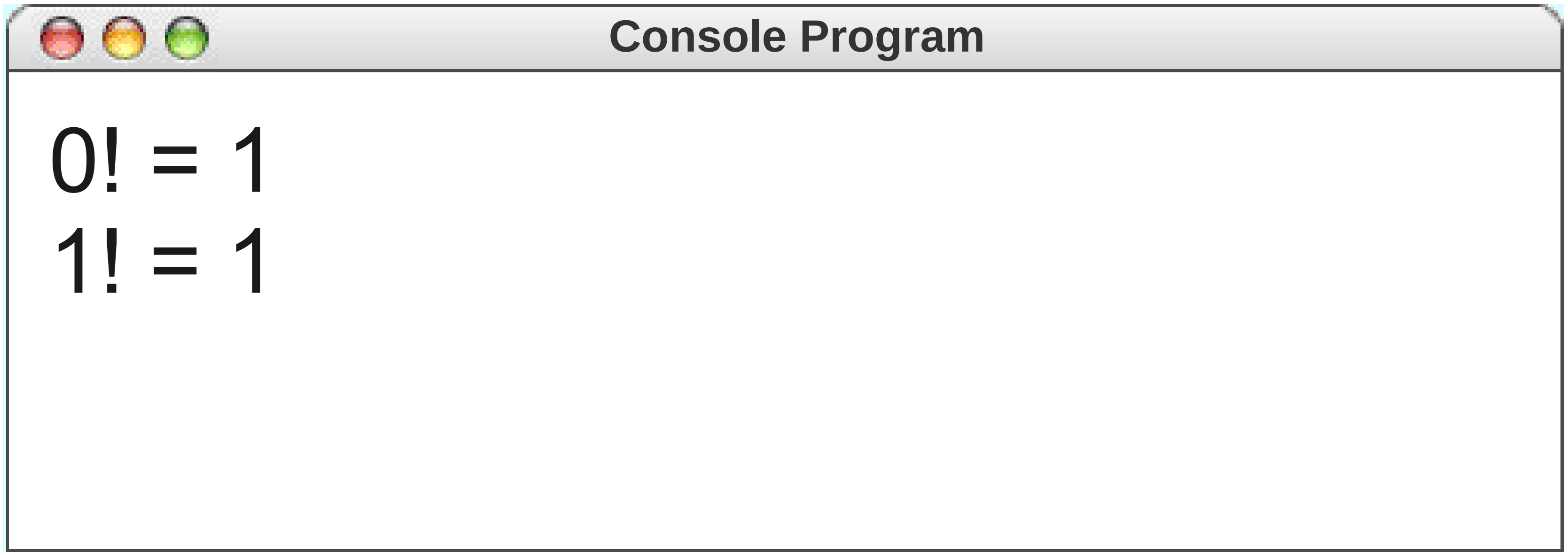




```
0! = 1
1! = 1
```

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

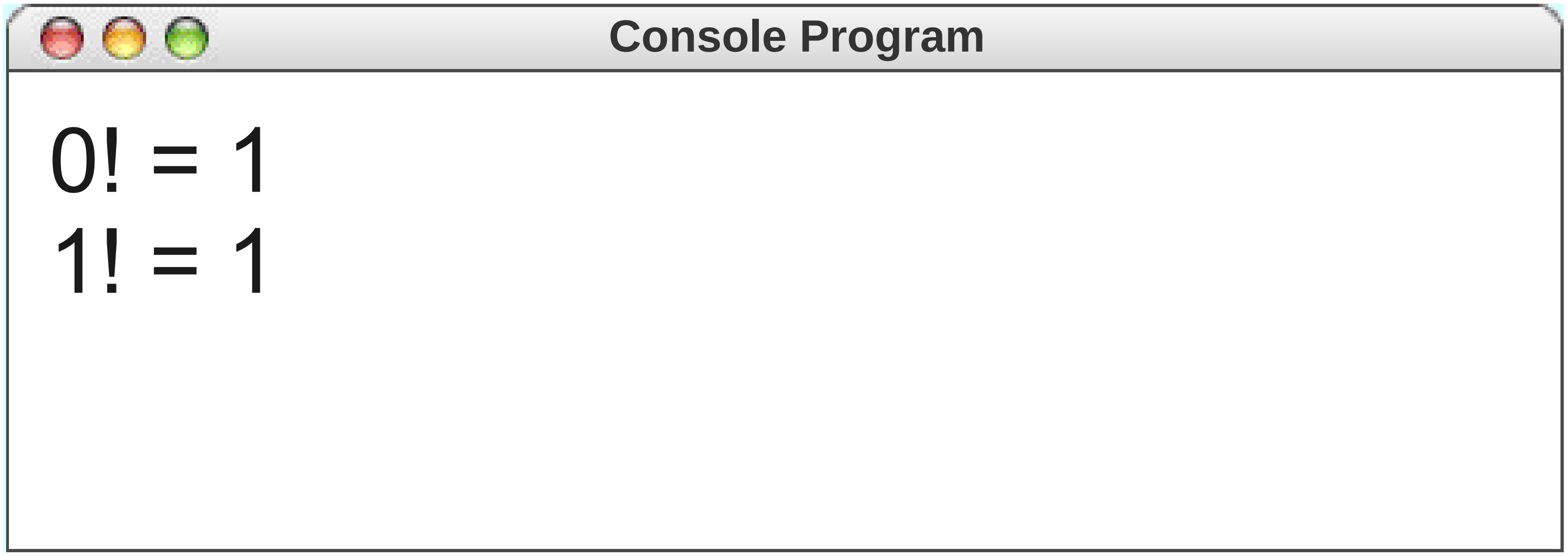
i $\boxed{2}$



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

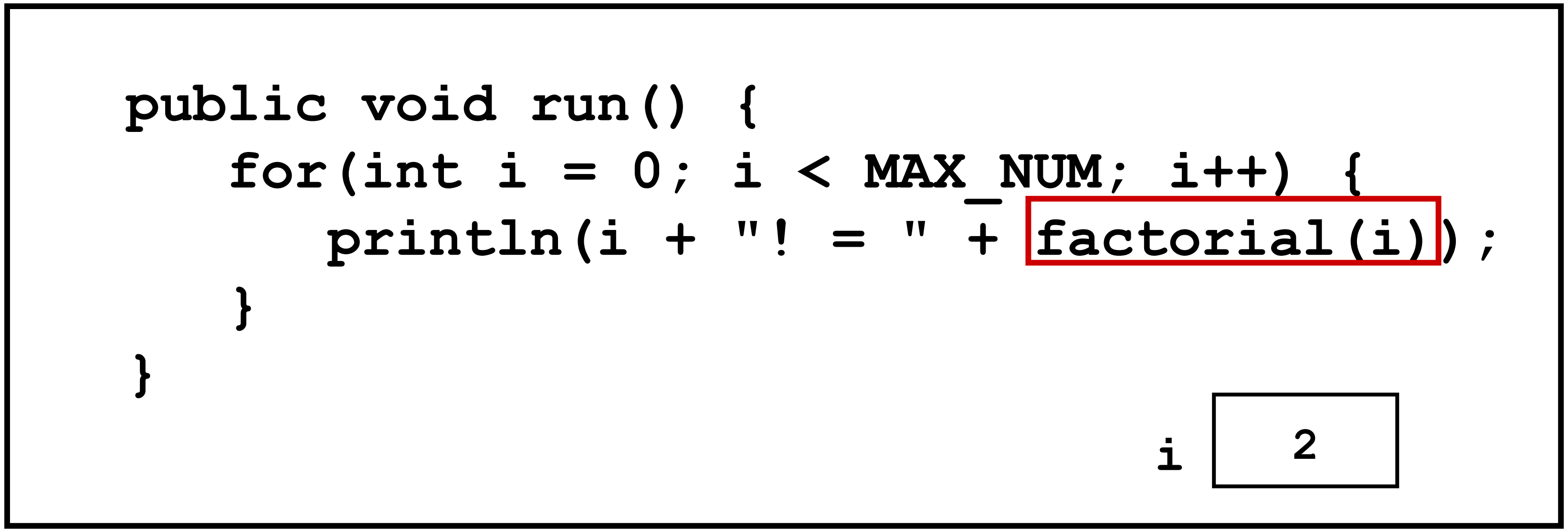
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

```
i  2
```

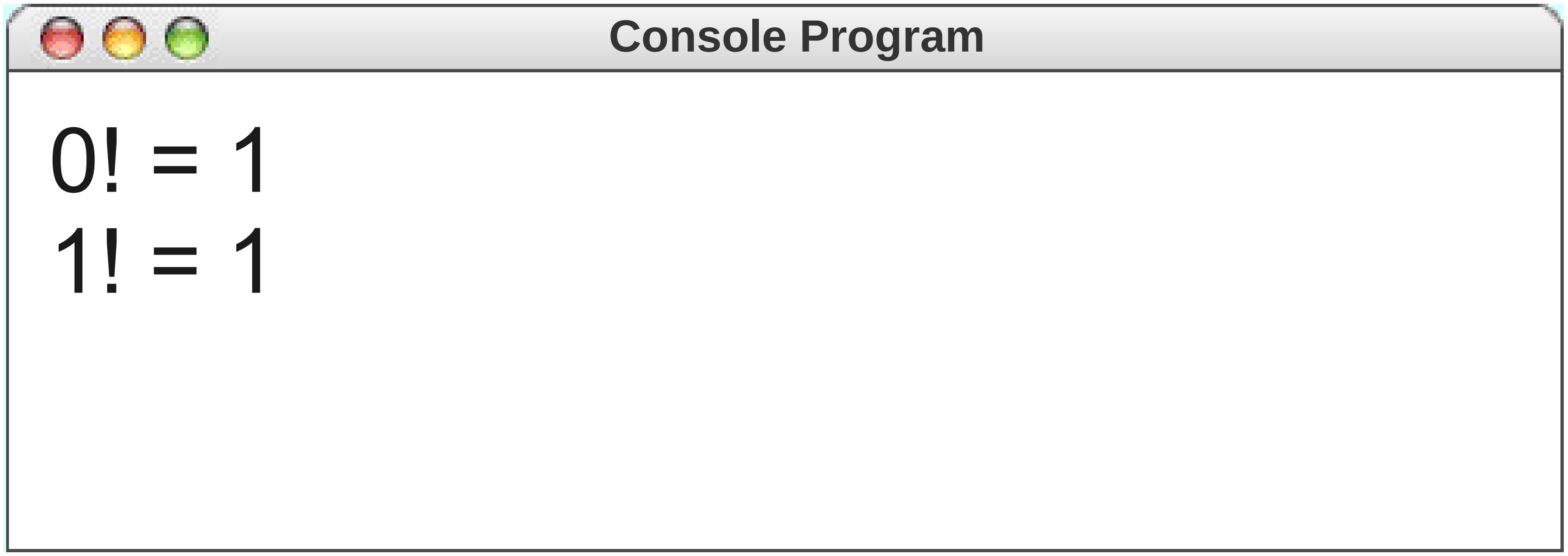


Console Program

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



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

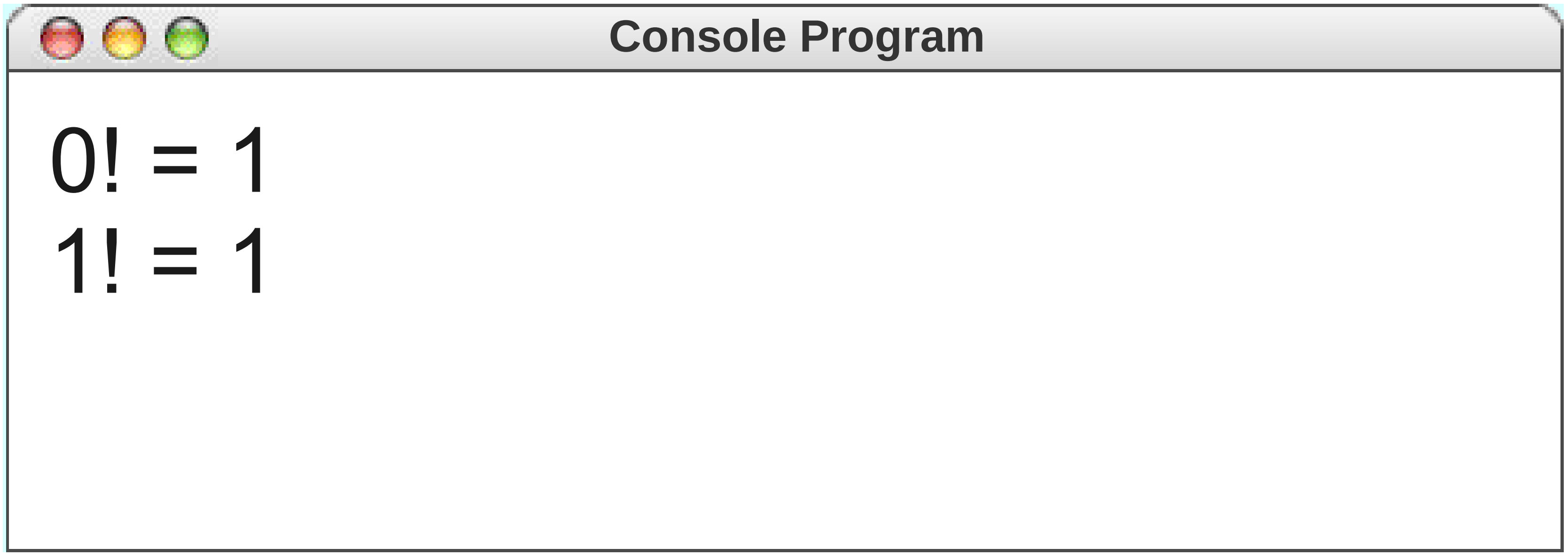


```
0! = 1
1! = 1
```

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

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

i $\boxed{2}$

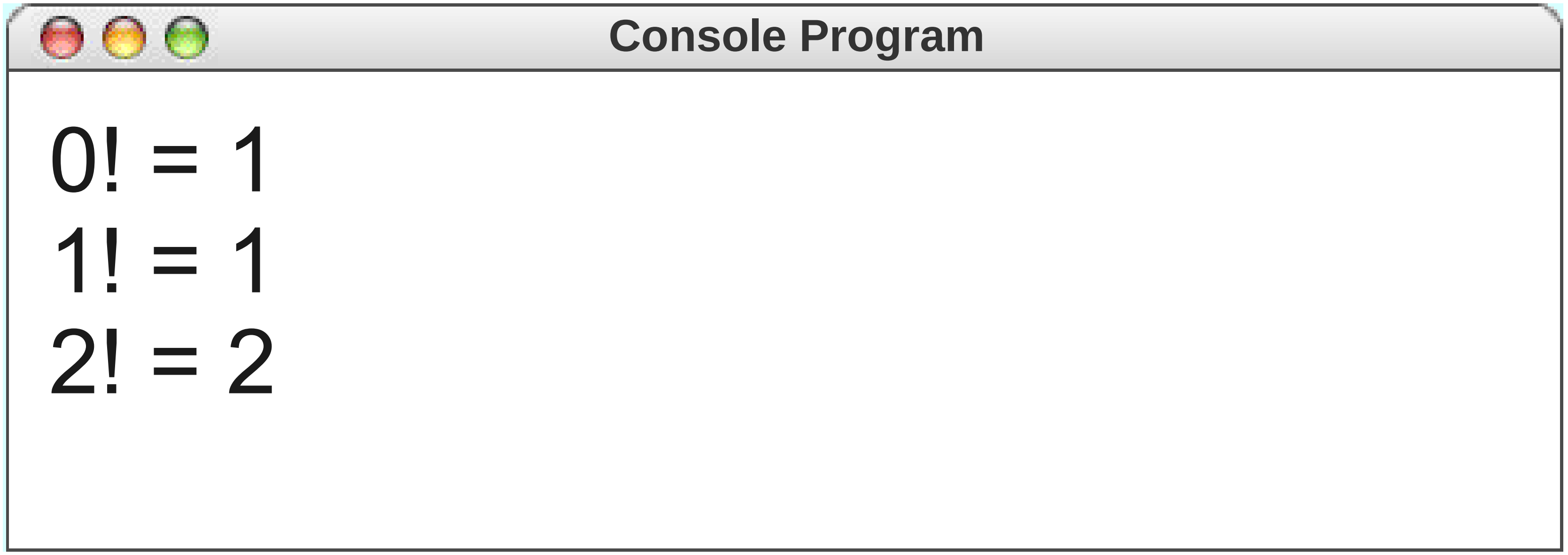


```
0! = 1
1! = 1
```

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

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

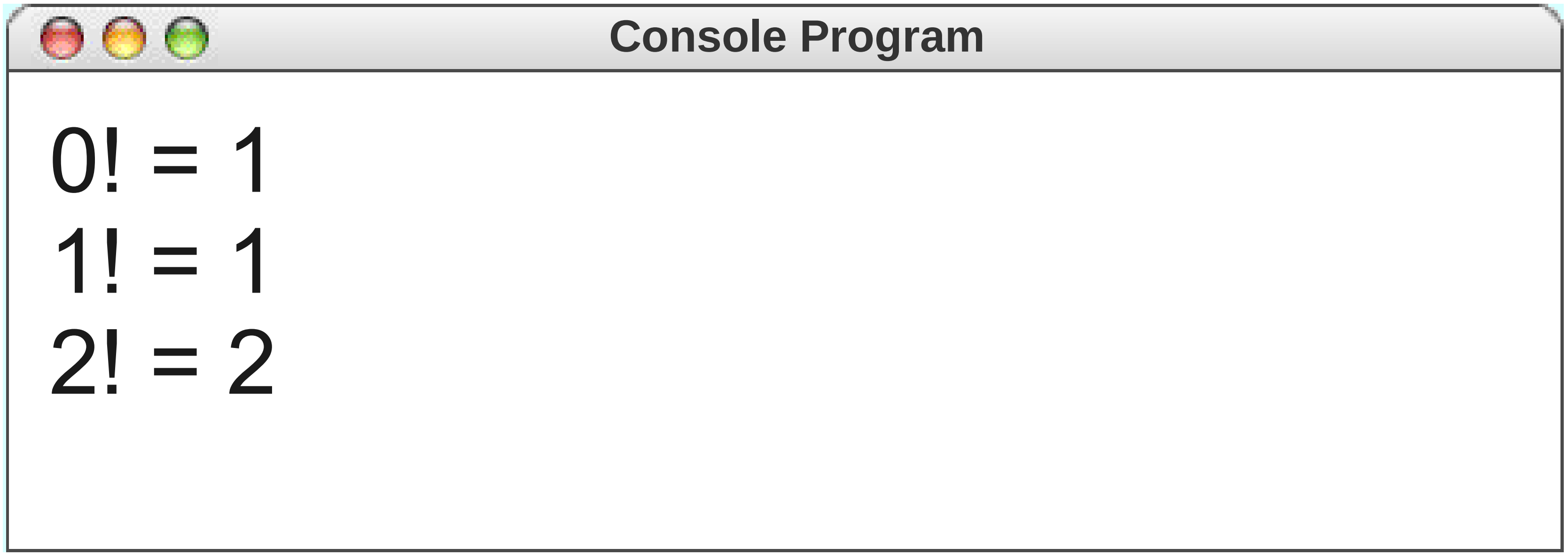
i $\boxed{2}$



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

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

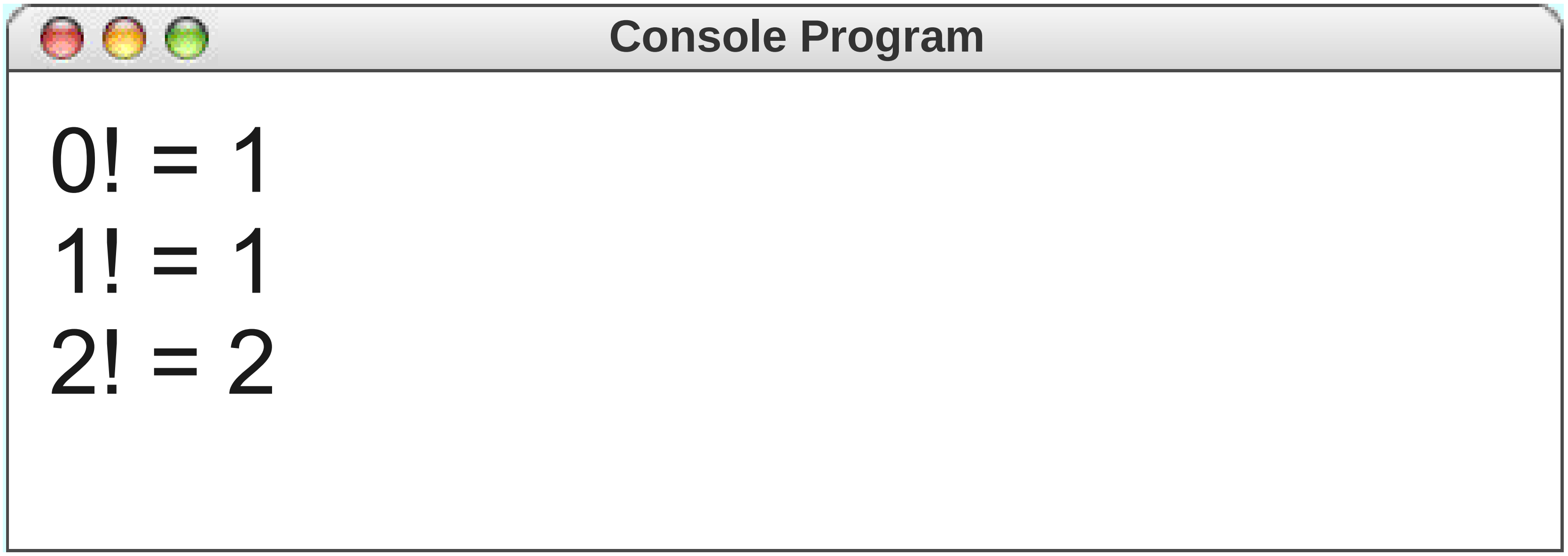
```
i  3
```



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

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

i $\boxed{3}$

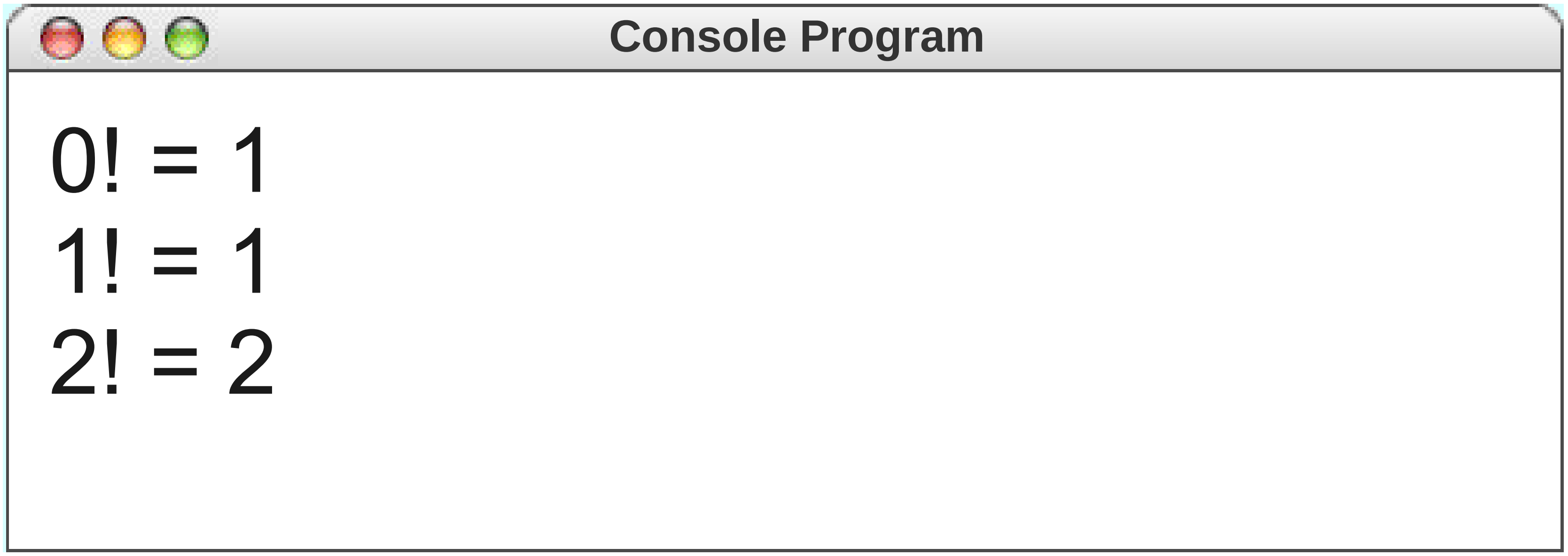


Console Program

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

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

i $\boxed{3}$



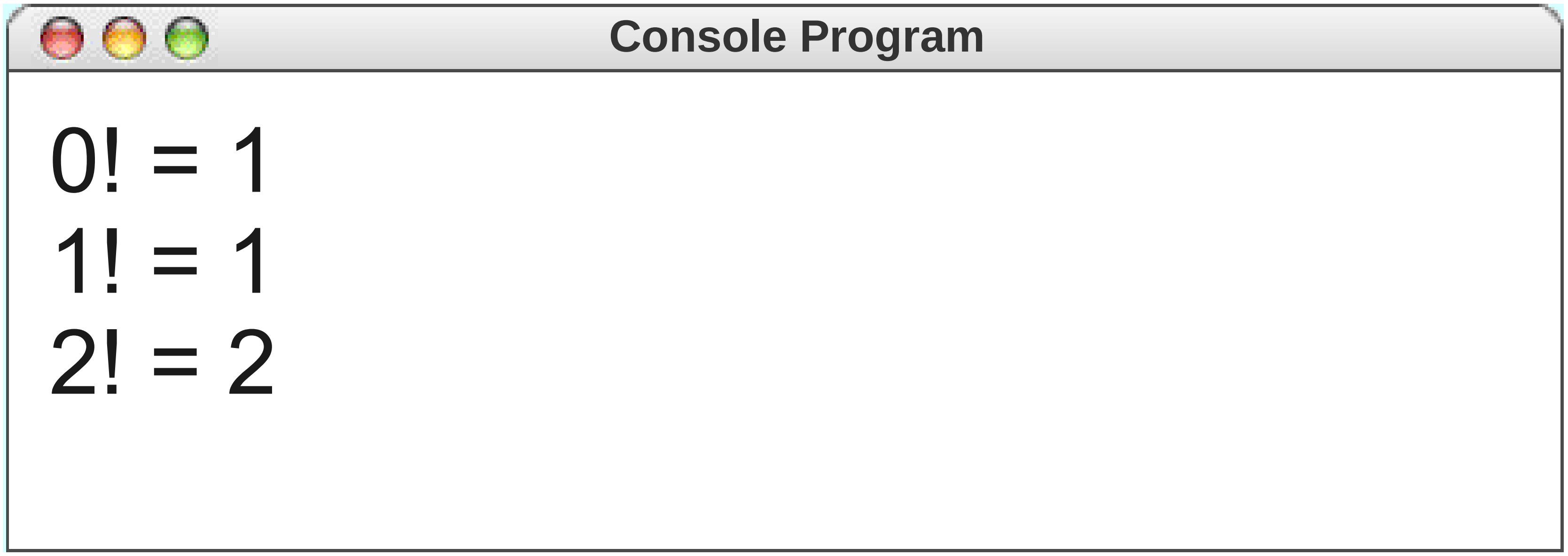
Console Program

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

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

$$\boxed{\texttt{factorial(i)}}$$

i $\boxed{3}$

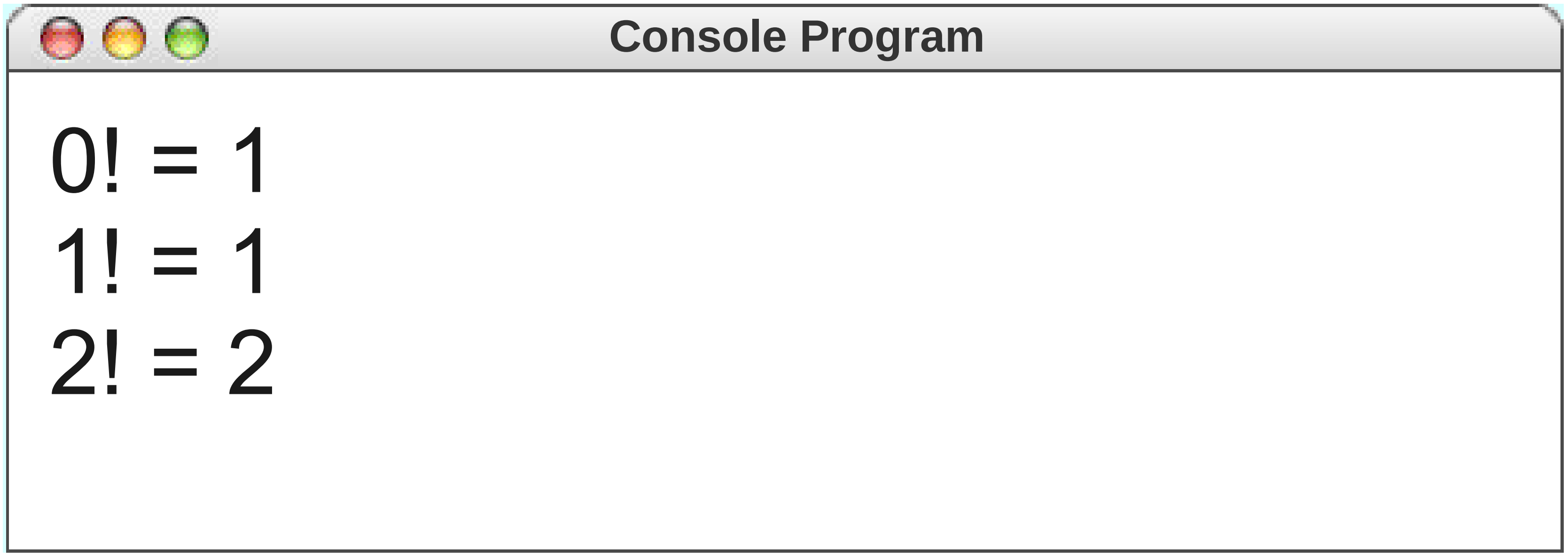

Console Program

```
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 $\boxed{3}$

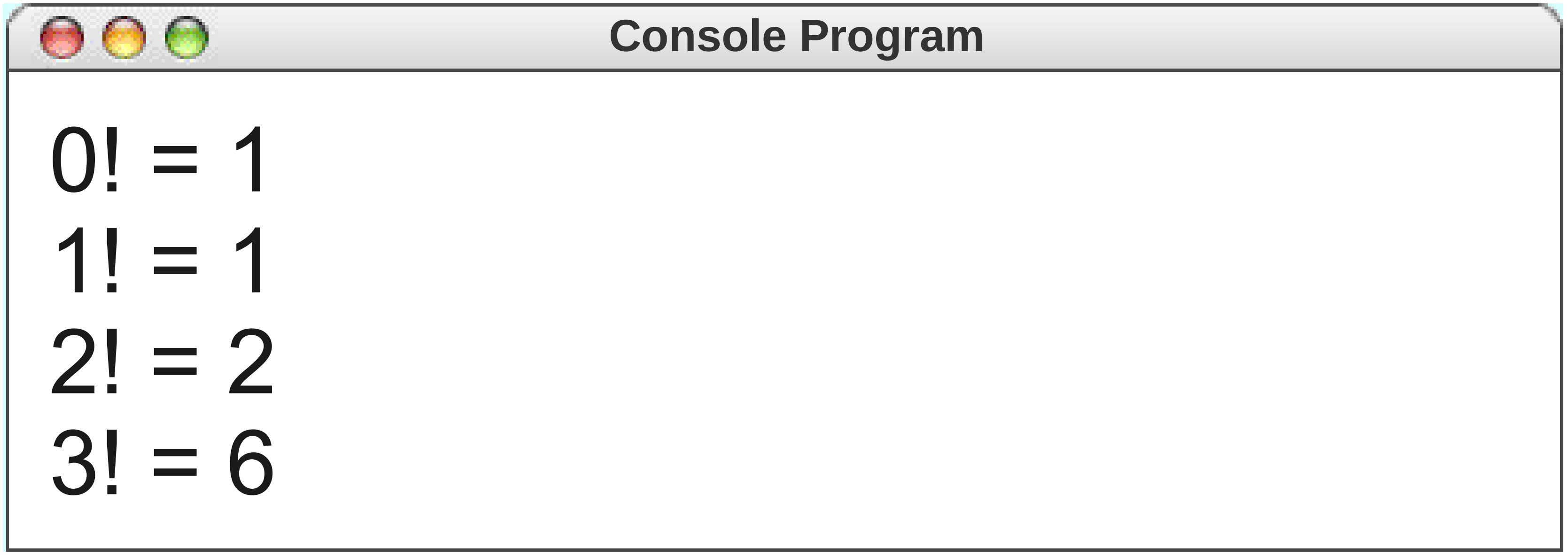


```
0! = 1
1! = 1
2! = 2
```

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

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

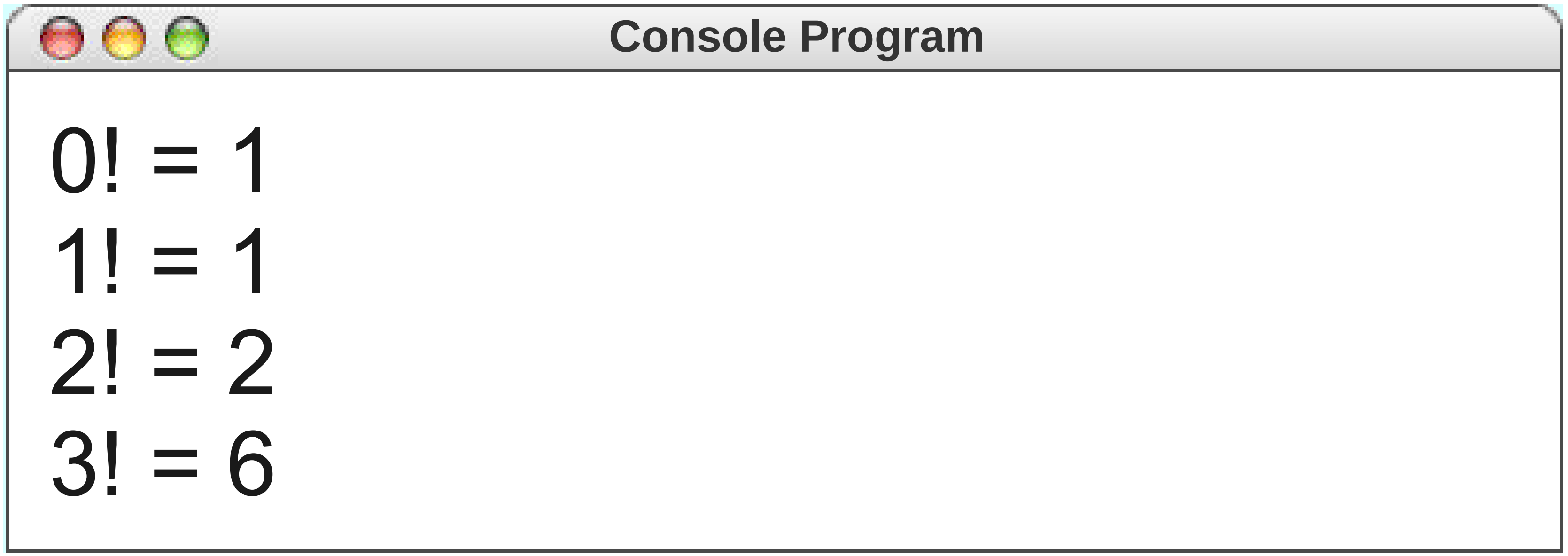
i $\boxed{3}$



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

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

```
i  4
```

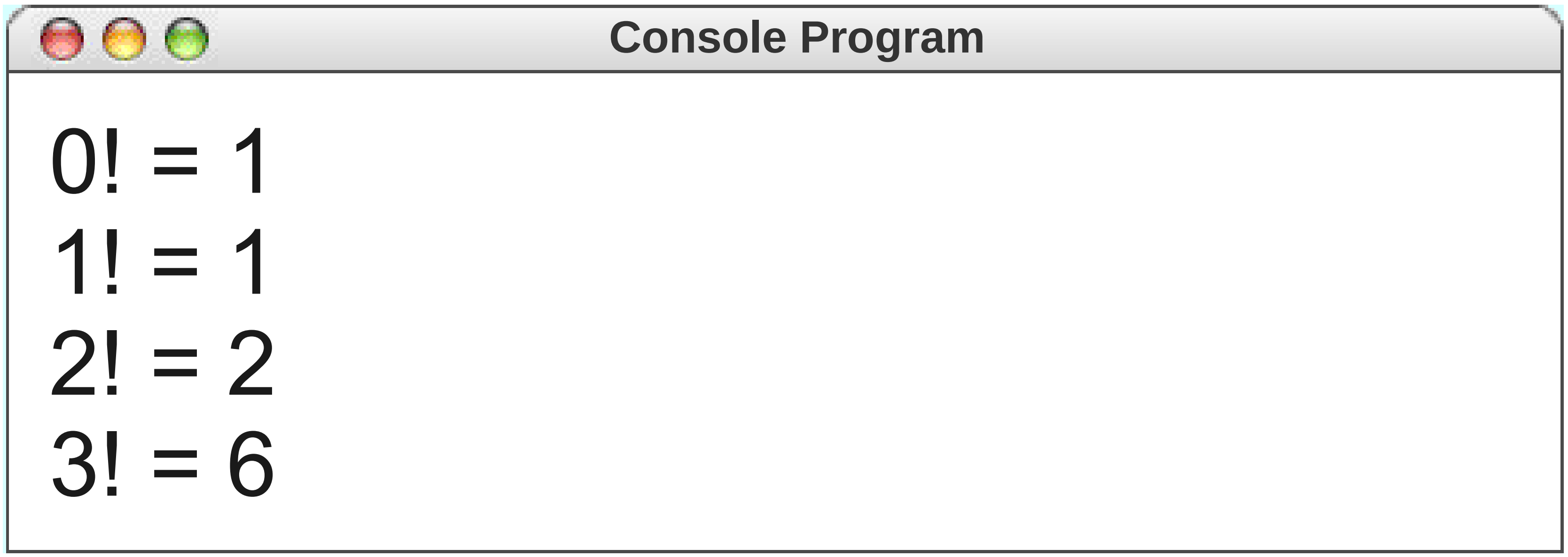

Console Program

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

Slides by Mehran Sahami

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

```
i   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`		

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:

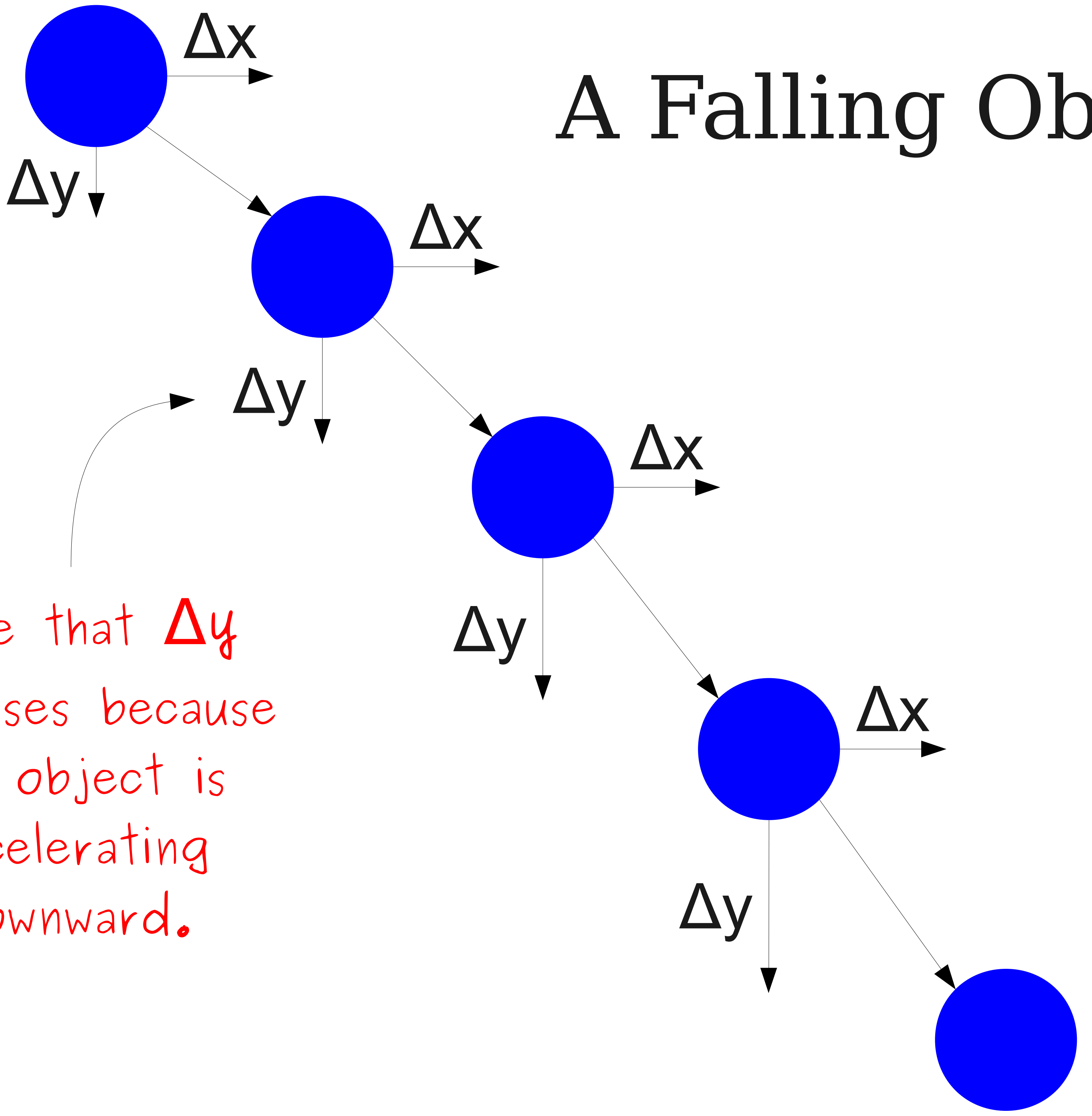
```
while (not-done-condition) {
    update graphics
    pause(pause-time);
}
```

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!