Model of Execution

Stack Examples

Lecture Task

- This is Lecture Task 6 from the Pale Blue Dot project -
- **Testing**: In the tests package, complete the test suite named Lecture Task6.
 - This suite will test a method in the PaleBlueDot object named "closestCity" that takes a String and a List of Doubles as parameters and returns a List of Strings
 - The inputs represent the filename for the cities file and a latitude/longitude location. This method outputs the country code, name, and region of the city closest to the input location. The upper/lower-case and formatting should match exactly as it appears in the file.
 - Example output format: List("cn", "longjiang", "08")
 - If no closest city is found, the method outputs the empty list: List()
- Functionality: Implement the functionality for closestCity.

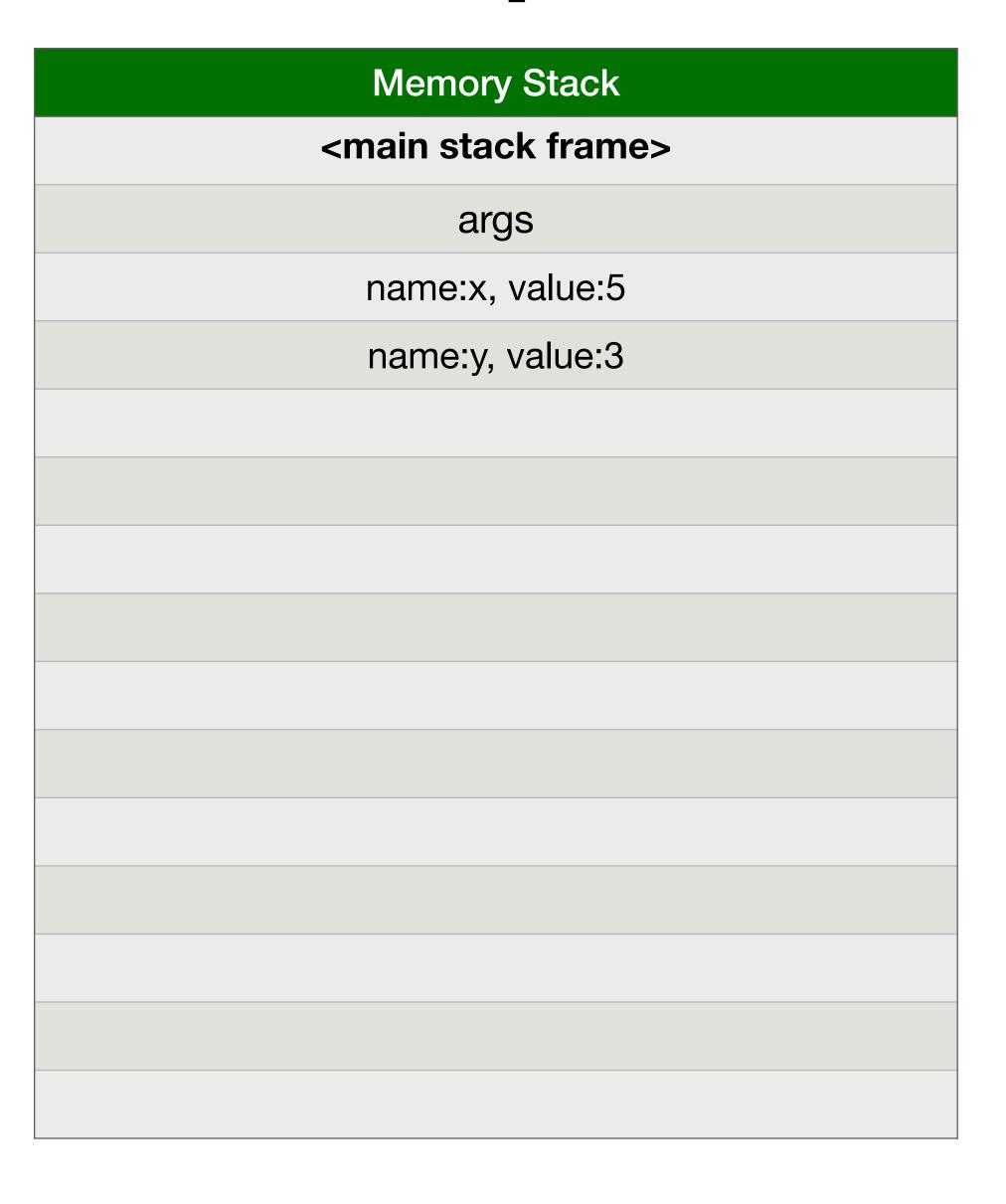
More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
       val x: Int = 1
        z += x
       } else {
       val x: Int = 1
        z -= x
       }
    }
    def main(args: Array[String]): Unit = {
       val x: Int = 5
       val y: Int = 3
       val z: Int = subtract(x, y)
       println(z)
    }
}</pre>
```

```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
       val x: Int = 1
        z += x
       } else {
       val x: Int = 1
        z -= x
       }
    }
    z
}

def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

- Start at the main method
- Create variables x, y





```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
         val x: Int = 1
         z += x
       } else {
         val x: Int = 1
         z -= x
       }
    }
    z
}

def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

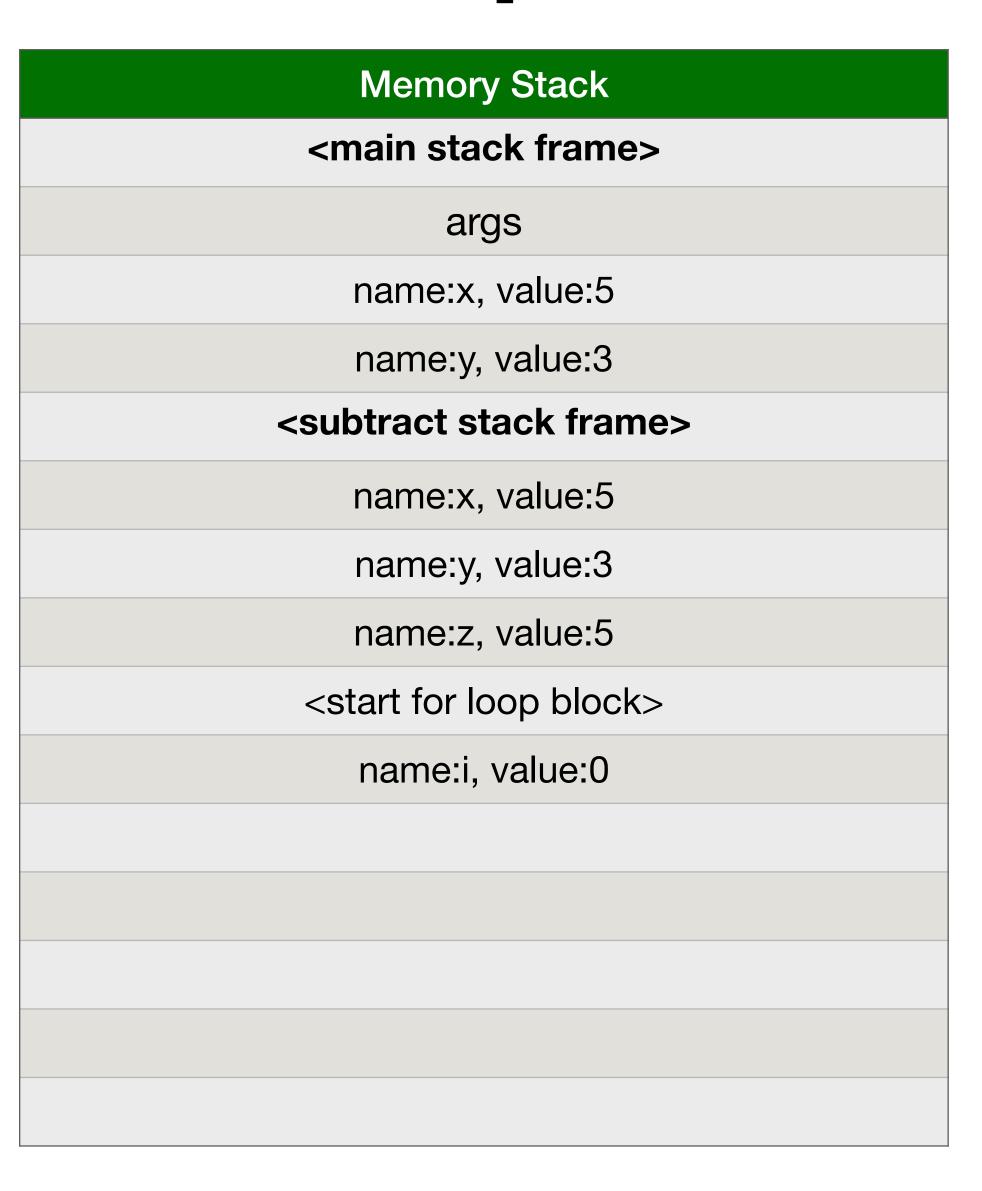
- Call the subtract method
- Add a new Stack Frame for the method call
- Add the parameter variables to the stack

Memory Stack
<main frame="" stack=""></main>
args
name:x, value:5
name:y, value:3
<subtract frame="" stack=""></subtract>
name:x, value:5
name:y, value:3

```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
        val x: Int = 20
        if (isNegative(y)) {
            val x: Int = 1
            z += x
        } else {
            val x: Int = 1
            z -= x
        }
    }
    z
}

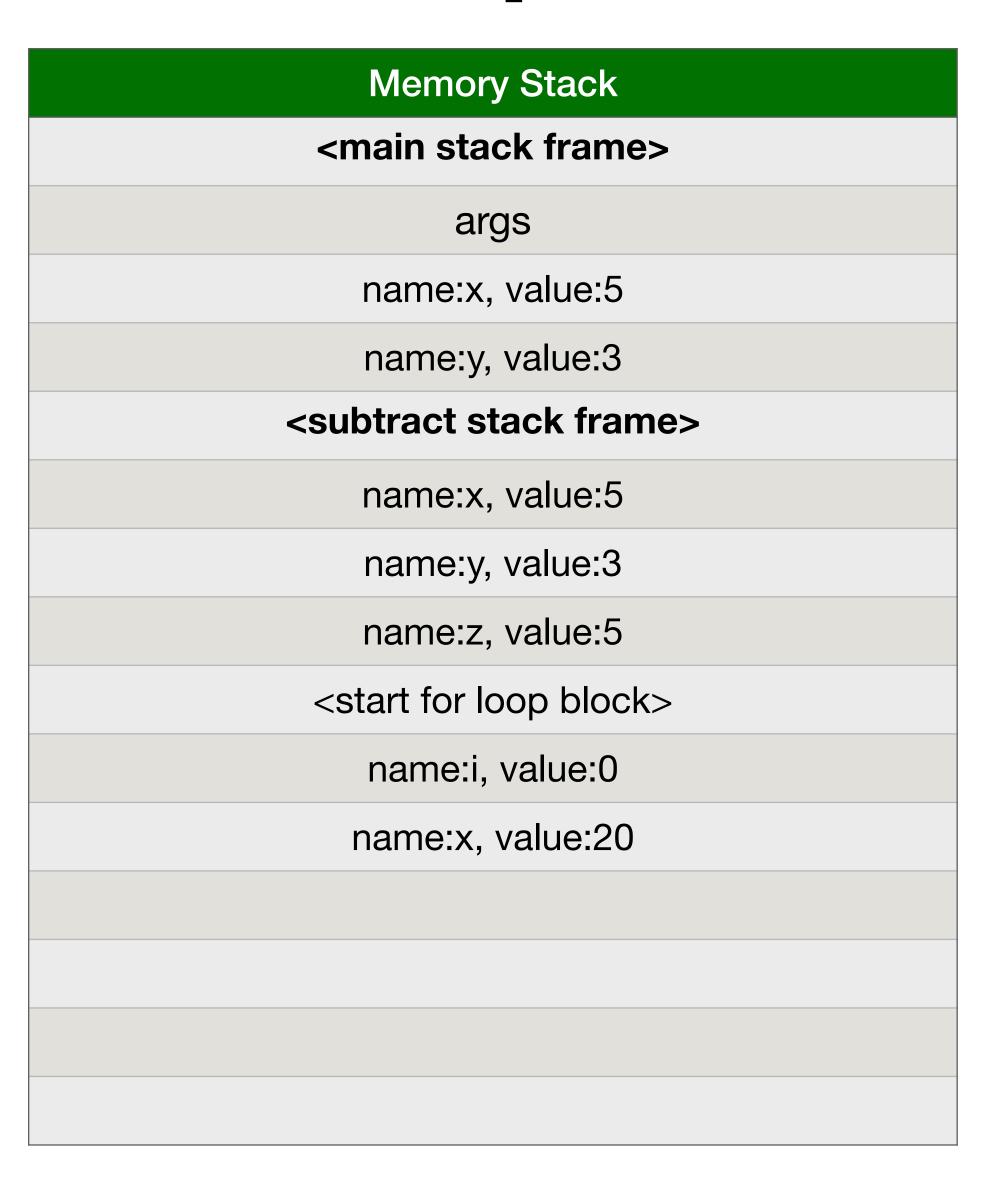
def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

- Add z to the stack
- Enter a new code block for the for loop
- Add i to the stack



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
       val x: Int = 1
        z += x
       } else {
       val x: Int = 1
        z -= x
       }
    }
    def main(args: Array[String]): Unit = {
       val x: Int = 5
       val y: Int = 3
       val z: Int = subtract(x, y)
       println(z)
}</pre>
```

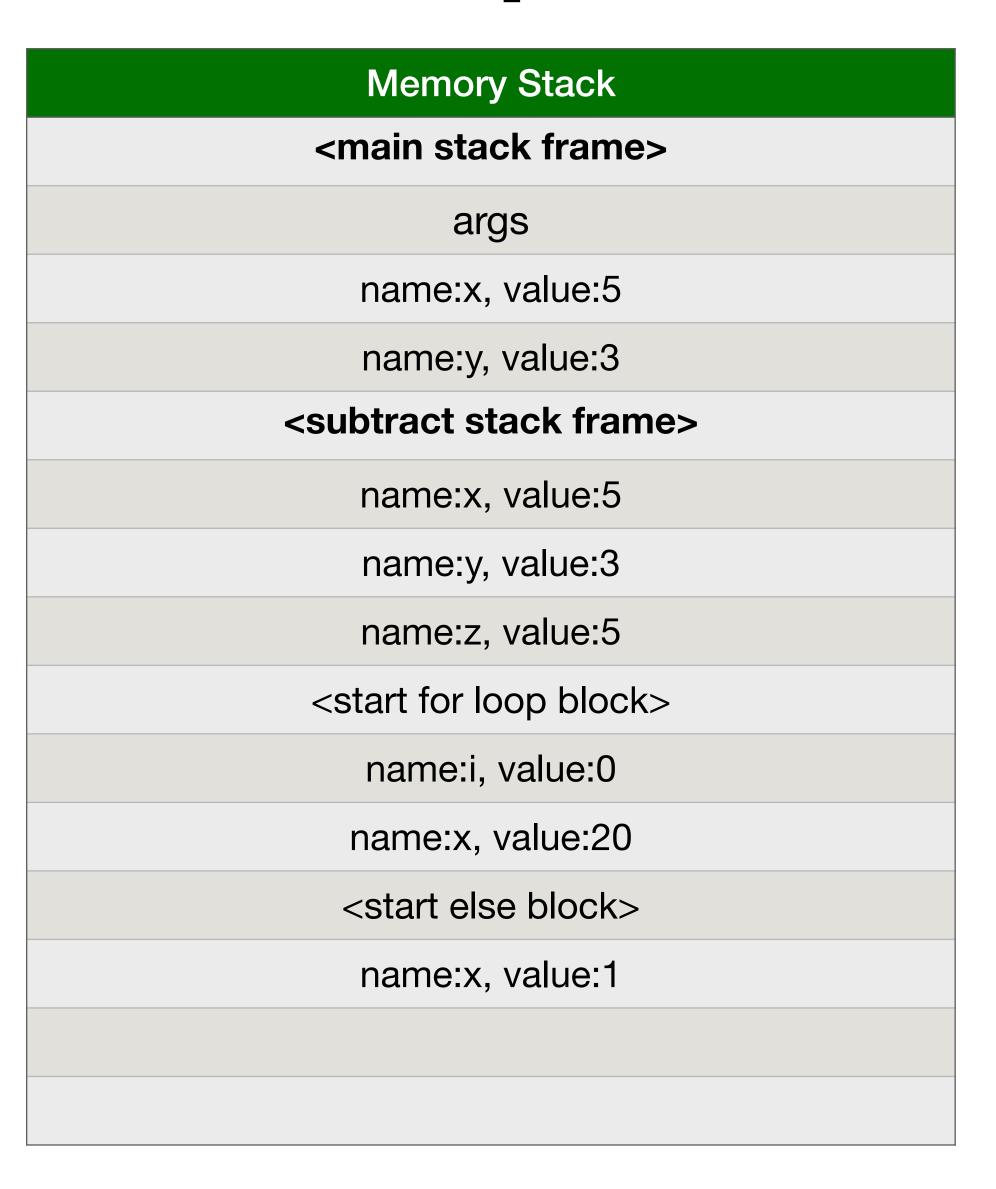
- Add x to the stack inside the for loop block
- There are 3 variables names x on the stack
- The one in the innermost block is "in scope" and will be used when we ask for the value of x



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
        val x: Int = 20
        if (isNegative(y)) {
            val x: Int = 1
                z += x
        } else {
            val x: Int = 1
                z -= x
        }
    }
    z
}

def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

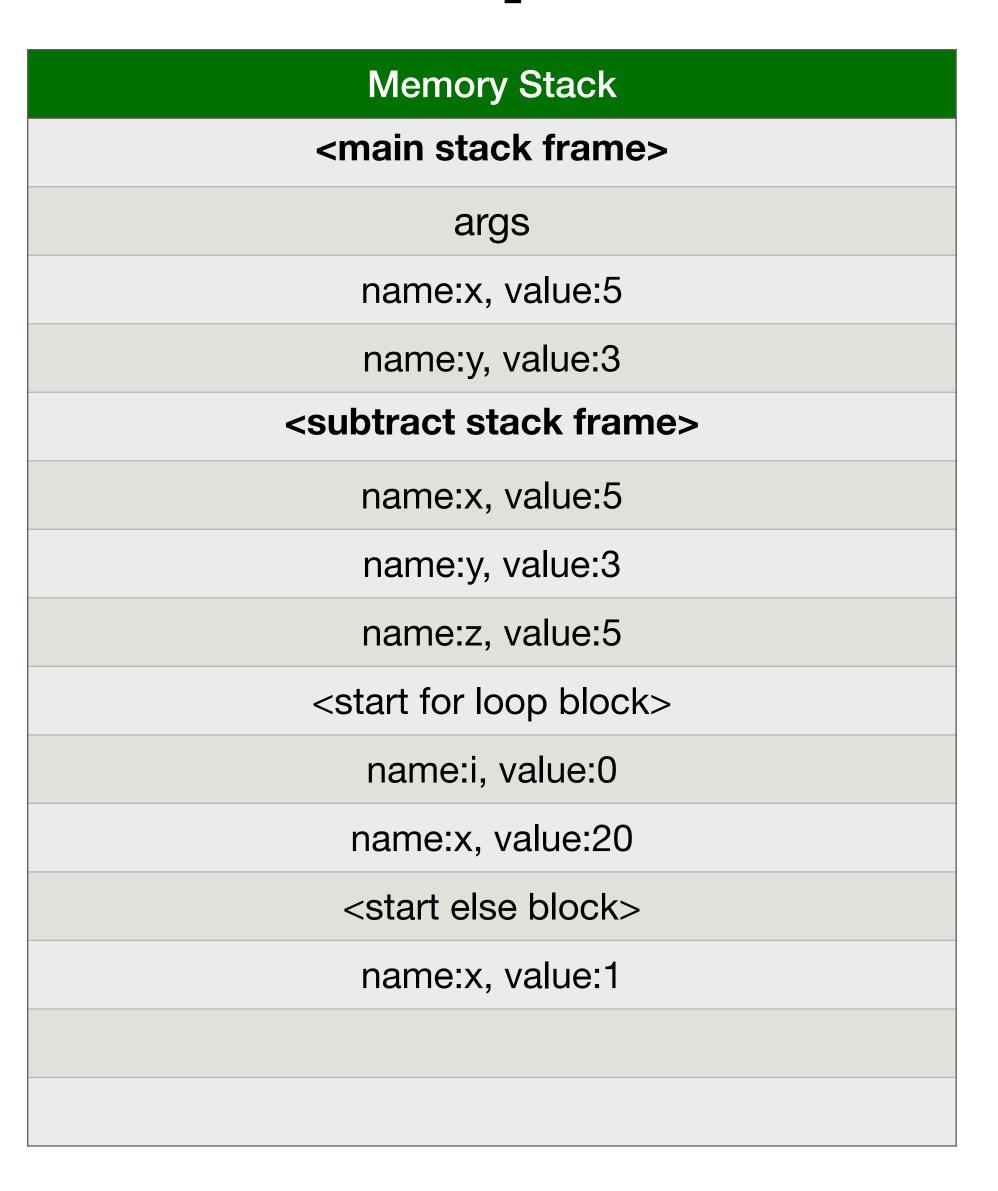
- The conditional is false
- Enter the else block
- Put another x on the stack



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
        val x: Int = 20
        if (isNegative(y)) {
            val x: Int = 1
                z += x
        } else {
            val x: Int = 1
                z -= x
        }
    }
    z
}

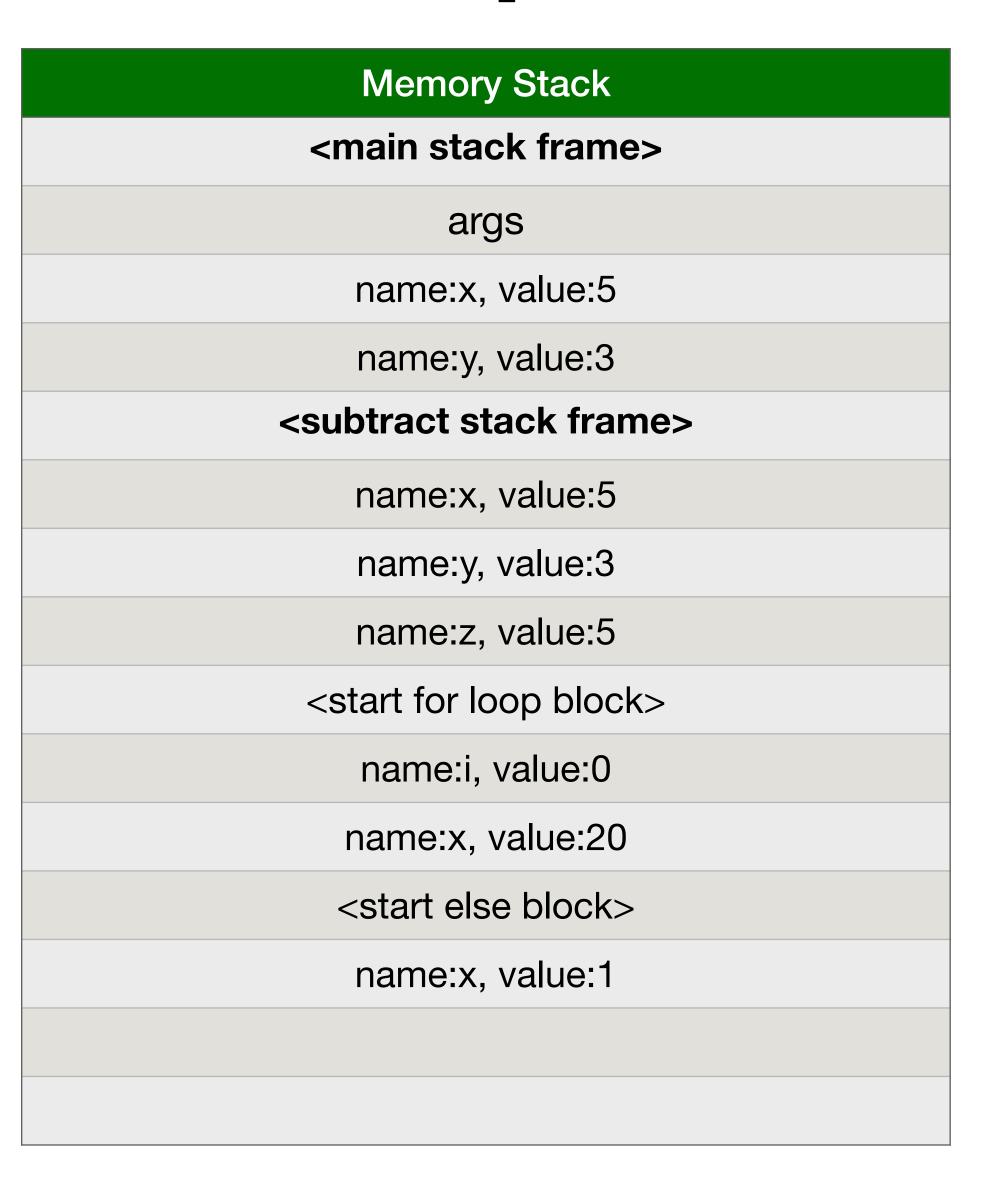
def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

- We need to compute z x
- But wait...
 - There are 4 different variables named x on the stack!
 - There's only 1 z, but it's not in this block!



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
        val x: Int = 20
        if (isNegative(y)) {
            val x: Int = 1
                z += x
        } else {
            val x: Int = 1
                z -= x
        }
    }
}
def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

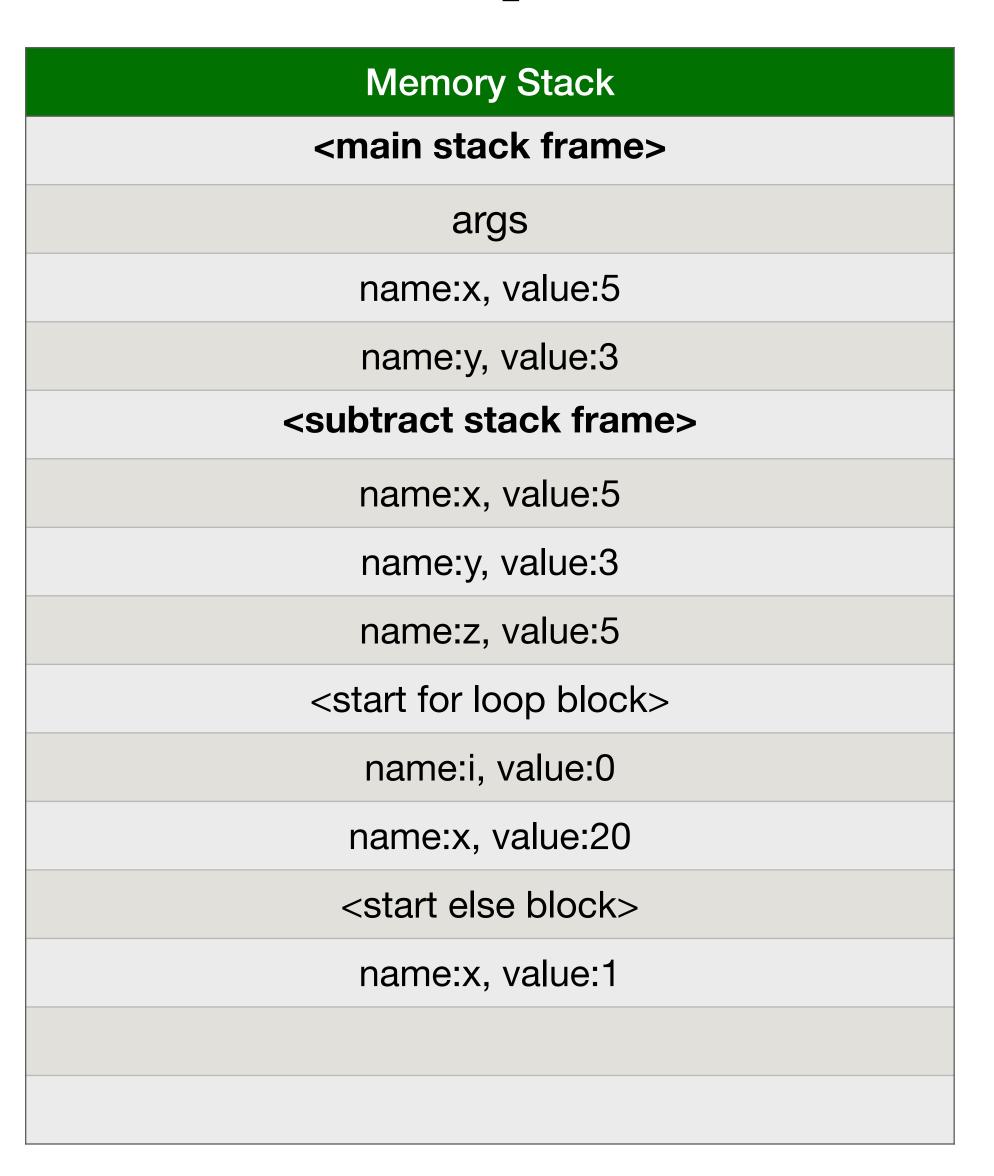
- Resolve x
 - Start looking in the current block for an x
 - We find one -> use it
 - x resolves to 1



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
        val x: Int = 1
          z += x
       } else {
        val x: Int = 1
          z -= x
       }
    }
}

def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

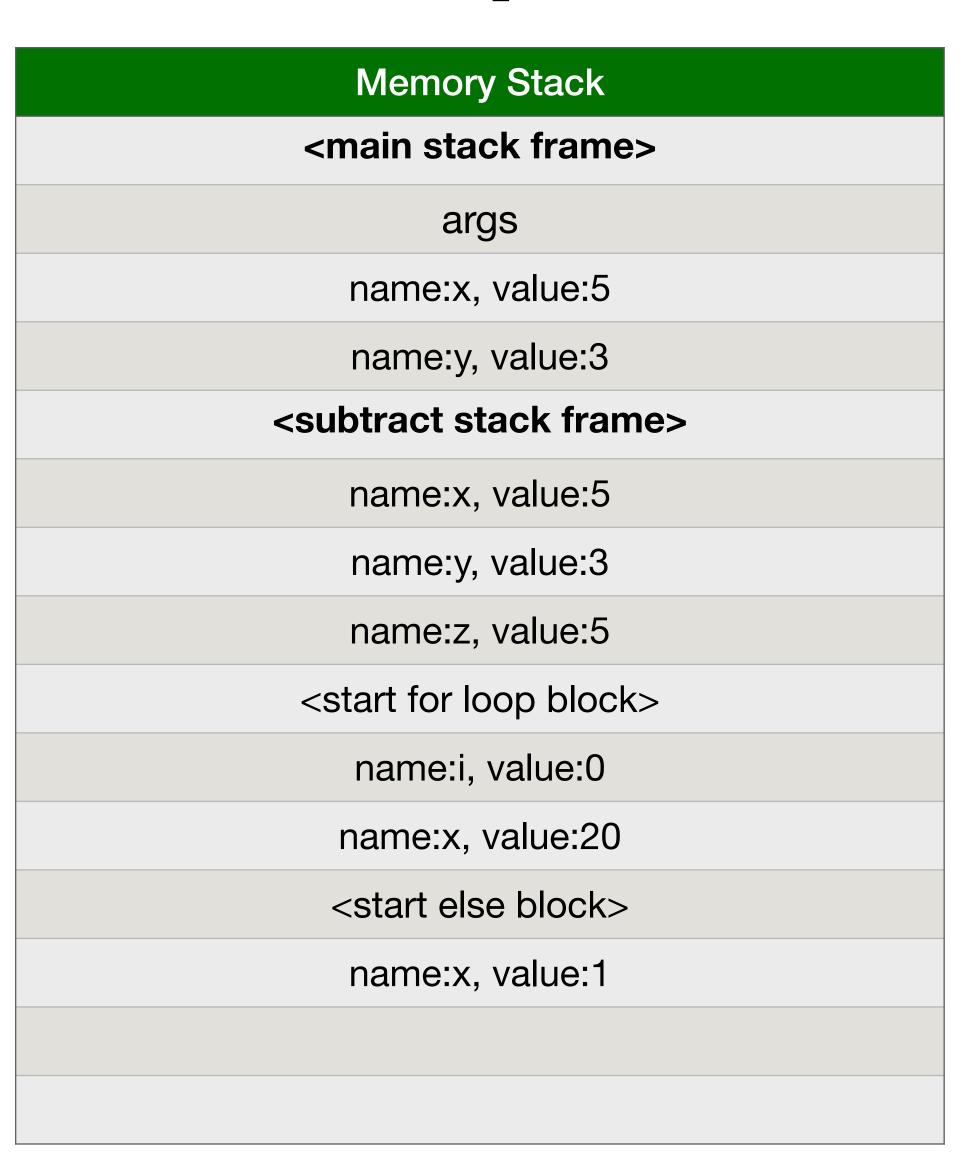
- Resolve z
 - Start looking in the current block for a z
 - We don't find one -> expand the search to the next block



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
        val x: Int = 20
        if (isNegative(y)) {
            val x: Int = 1
               z += x
        } else {
            val x: Int = 1
               z -= x
        }
    }
    z
}

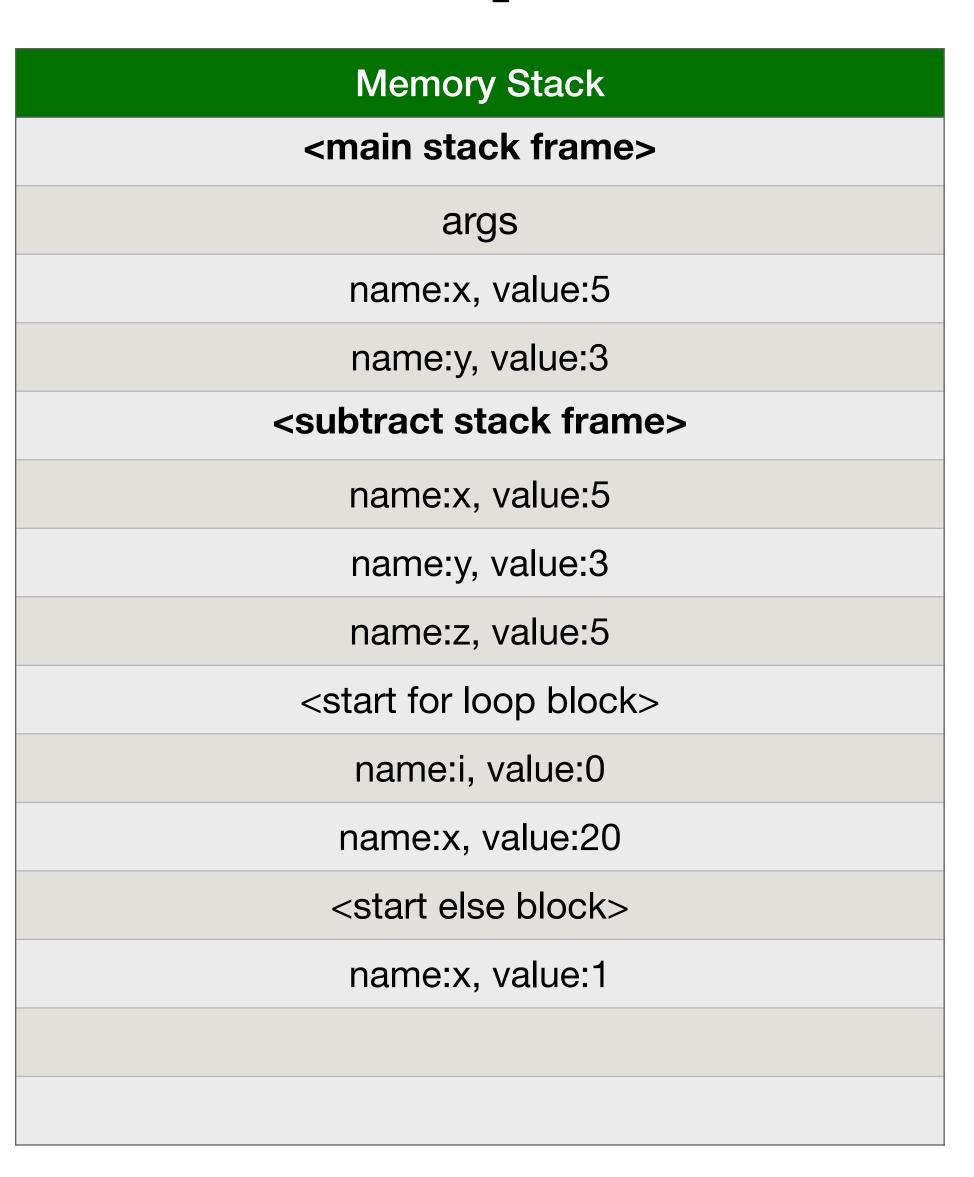
def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

- Resolve z cont'
 - We don't find z in the loop block either
 -> expand the search to the next block
 (The stack frame in this case)
 - We find z in the stack frame -> Resolves to 5



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
        val x: Int = 1
          z += x
       } else {
        val x: Int = 1
          z -= x
       }
    }
}
def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

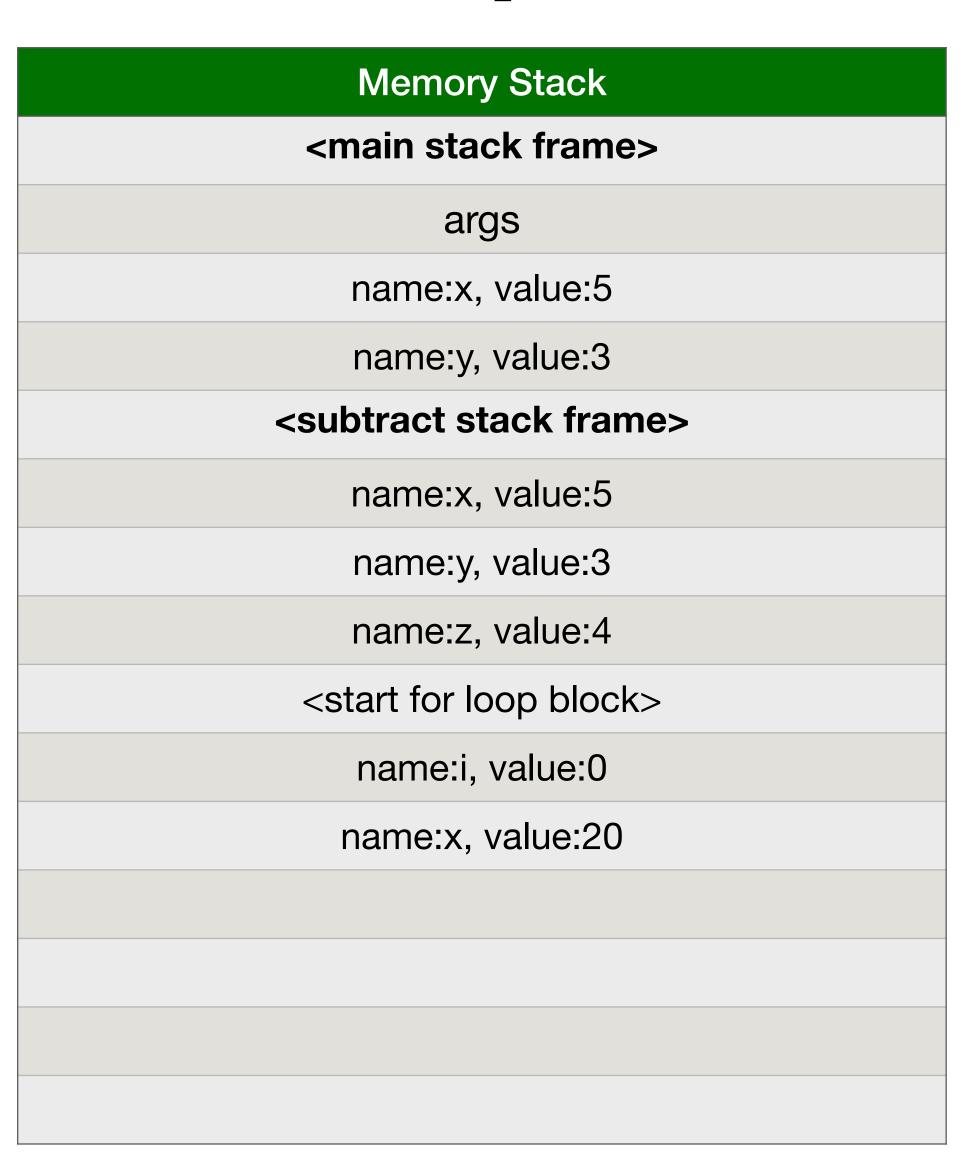
- Resolve z cont'
 - We don't find z in the loop block either
 -> expand the search to the next block
 (The stack frame in this case)
 - We find z in the stack frame -> Resolves to 5



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
    val x: Int = 1
        z += x
    } else {
    val x: Int = 1
        z -= x
    }
}

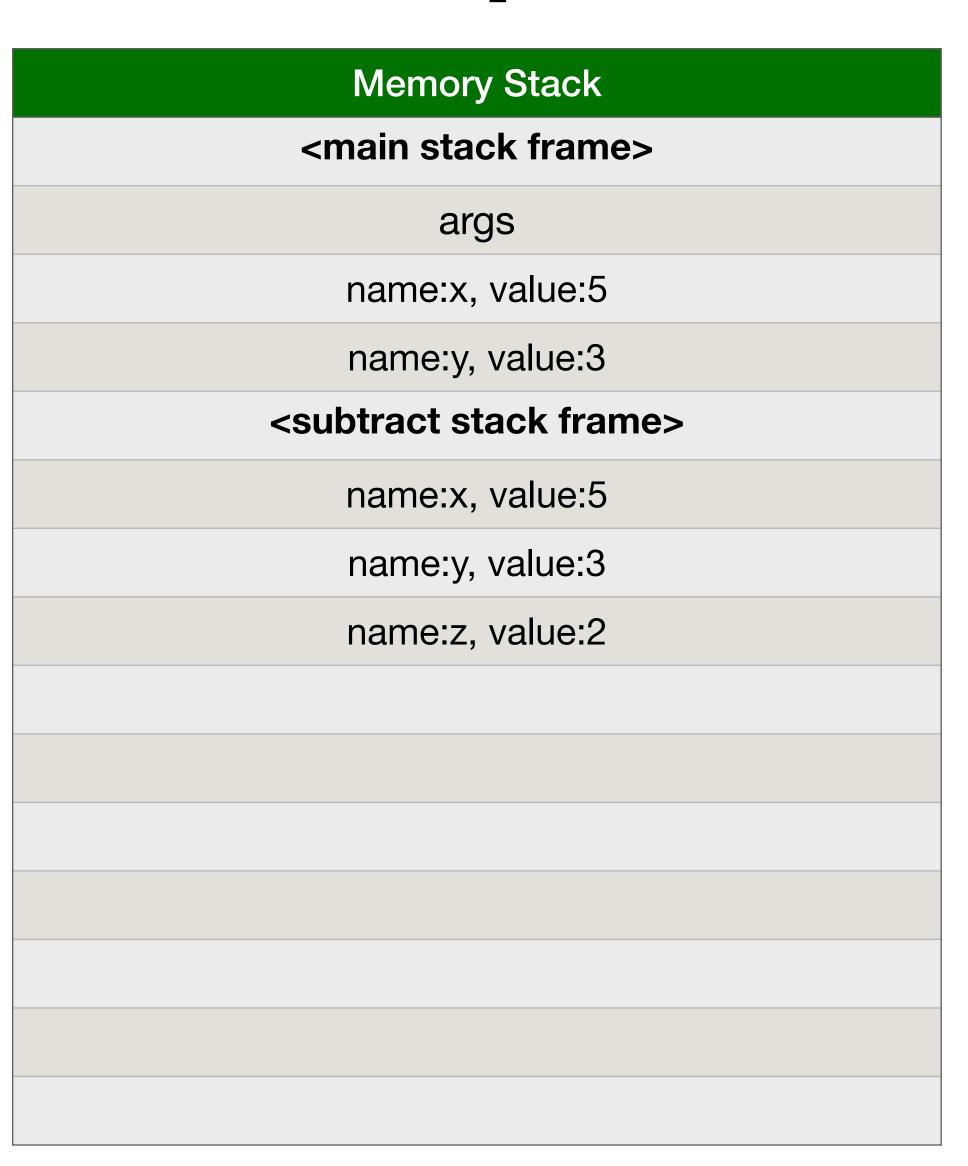
def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}</pre>
```

- Assign z to 5-1
- End of else block
 - Remove all variables declared in the else block



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
       val x: Int = 1
          z += x
       } else {
       val x: Int = 1
          z -= x
       }
    }
}
def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

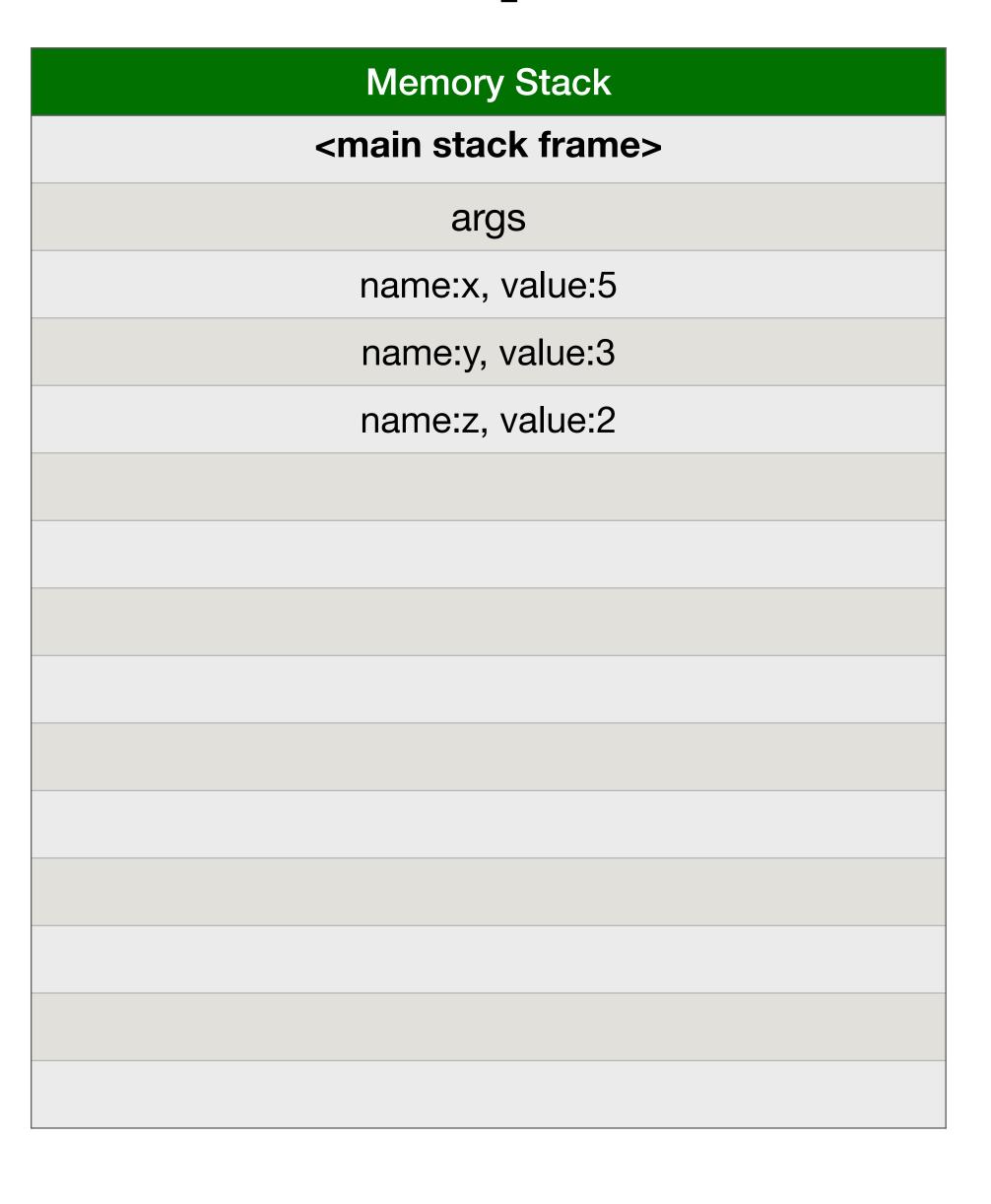
- Repeat until the loop ends
 - Remove it's block from the stack



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
       val x: Int = 1
          z += x
       } else {
       val x: Int = 1
          z -= x
       }
    }
    z
}

def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

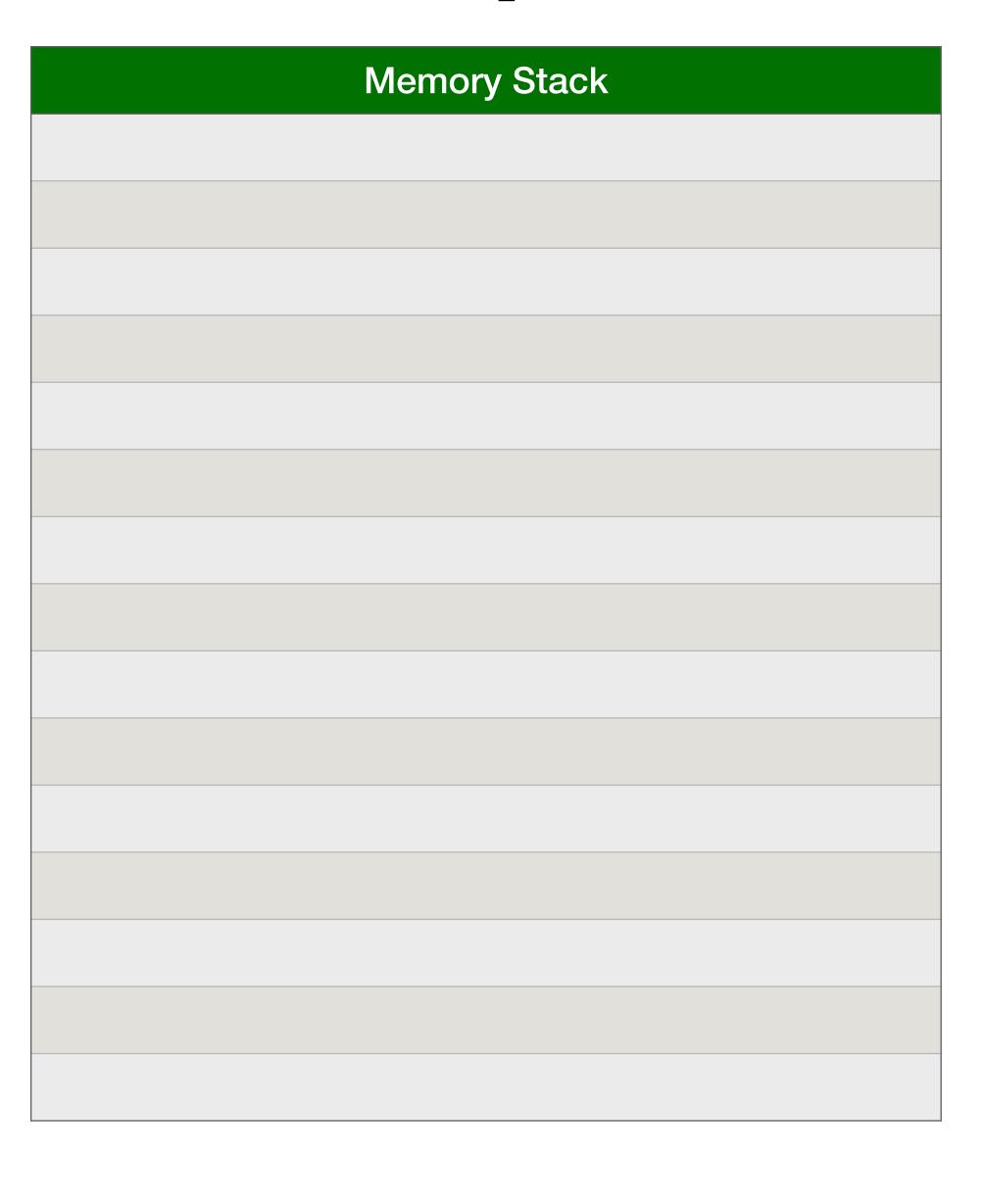
- Method returns
 - Remove its stack frame from the stack
 - subtract(x, y) resolves to 2
 - Add z to the stack and assign it the value 2



```
def subtract(x: Int, y: Int): Int = {
    var z: Int = x
    for (i <- 0 until Math.abs(y)) {
       val x: Int = 20
       if (isNegative(y)) {
         val x: Int = 1
         z += x
       } else {
         val x: Int = 1
         z -= x
       }
    }
  }
  Z
}

def main(args: Array[String]): Unit = {
    val x: Int = 5
    val y: Int = 3
    val z: Int = subtract(x, y)
    println(z)
}</pre>
```

- Main method ends
 - Remove it's frame from the stack
- Program ends



More Memory Examples

Multiple frames on the stack

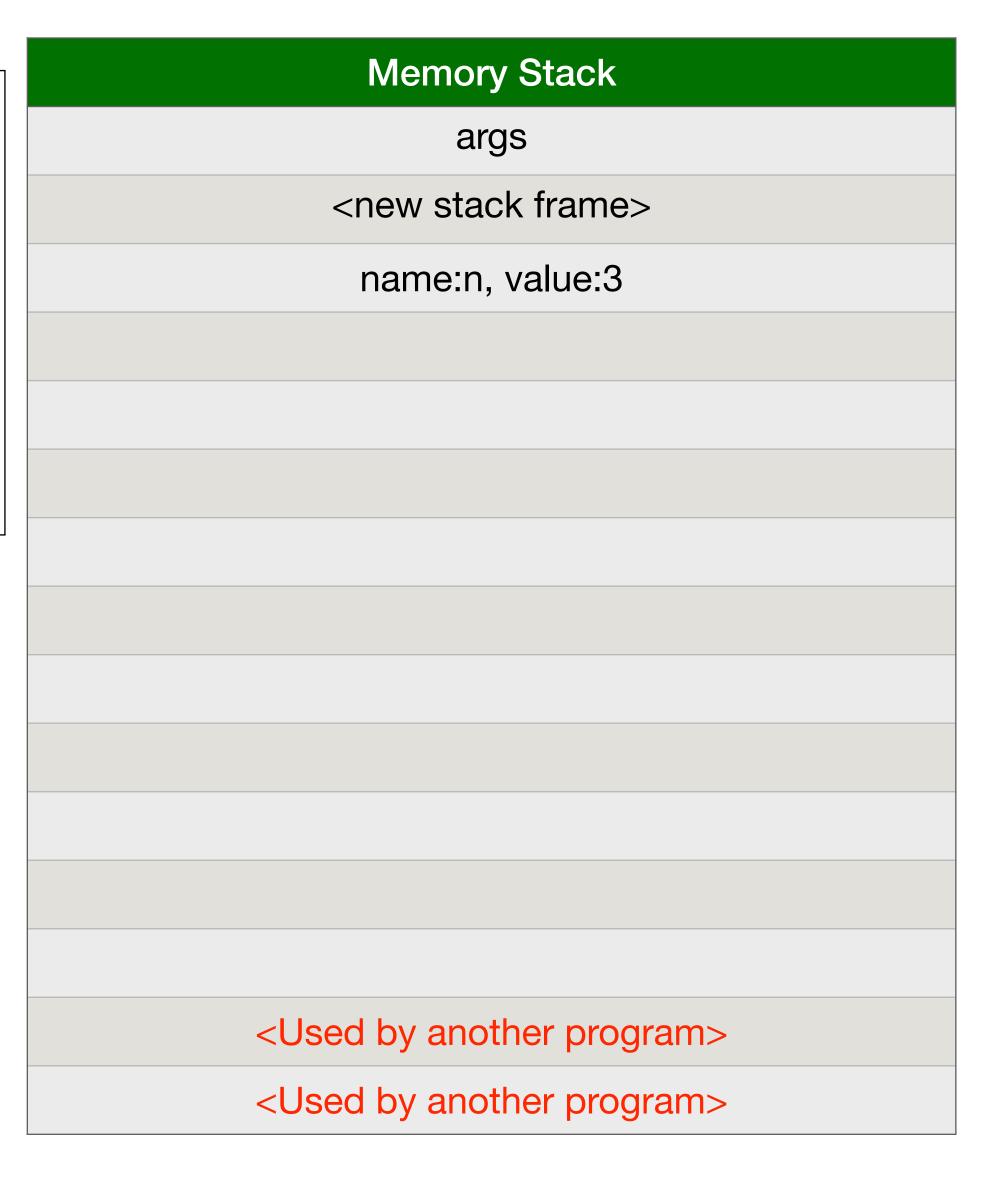
```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
    }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

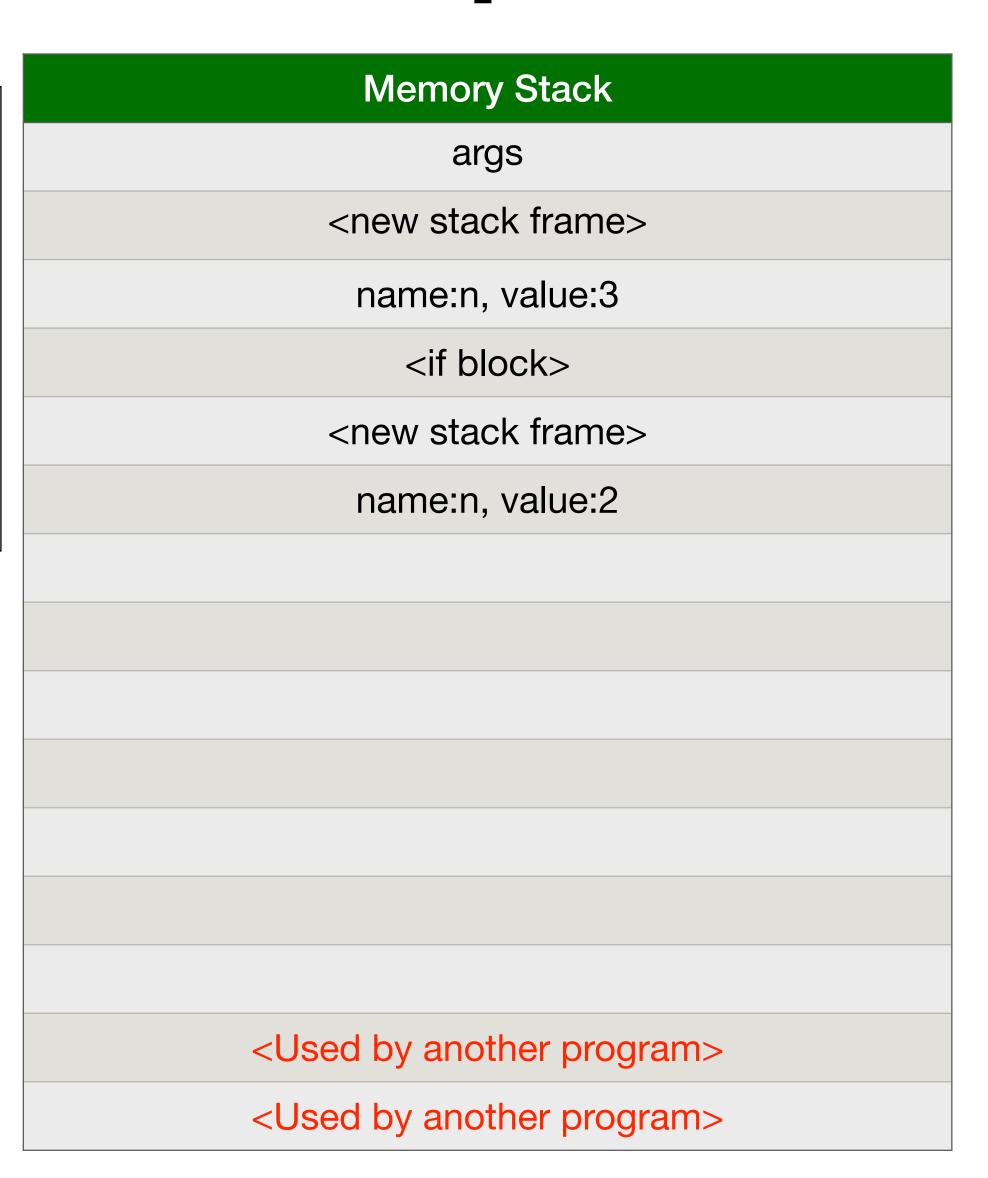
- Call function
- Create new stack frame



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

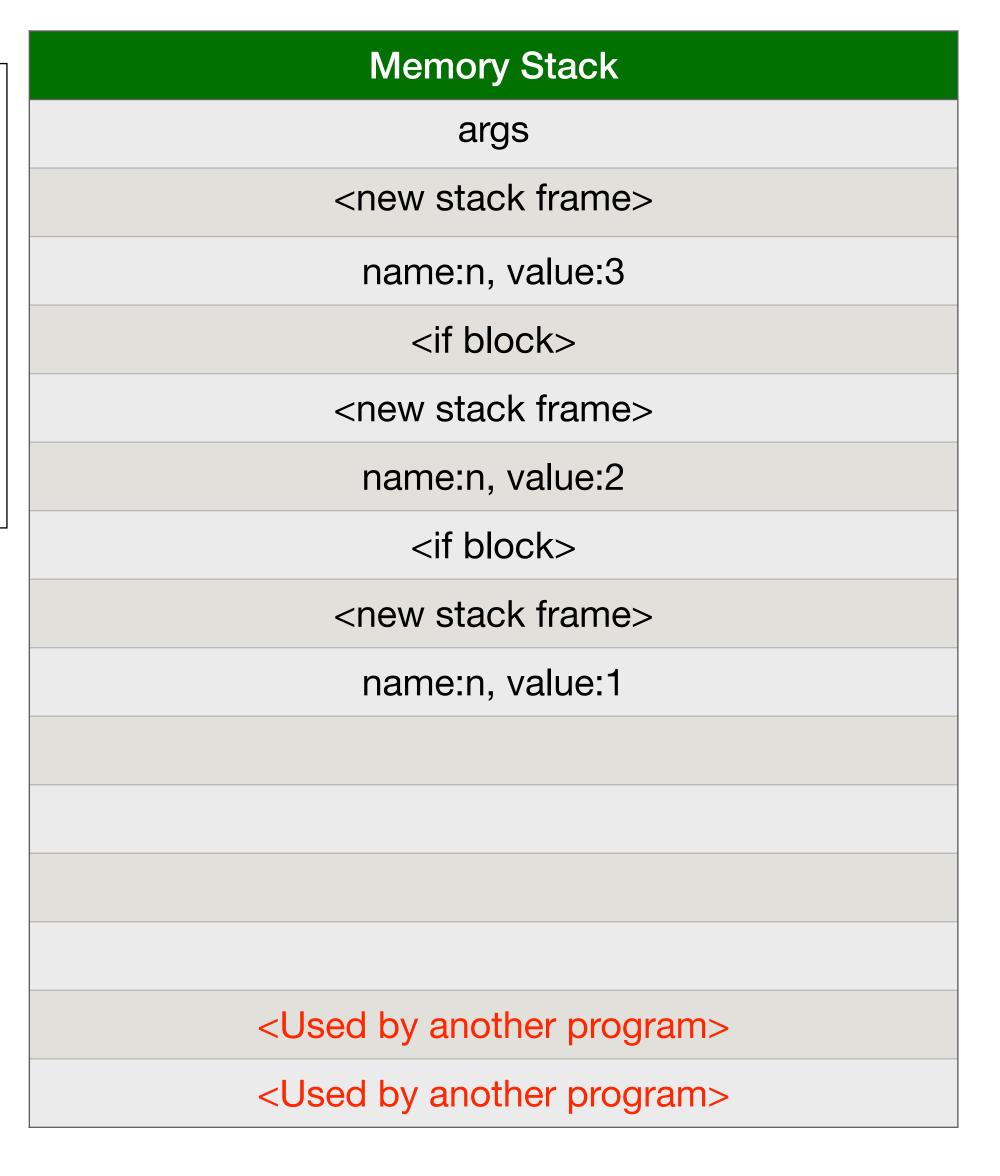
- Enter if block
- Call function again
- Create new stack frame



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

- In next function call, conditional true
- New if block
- New stack frame



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

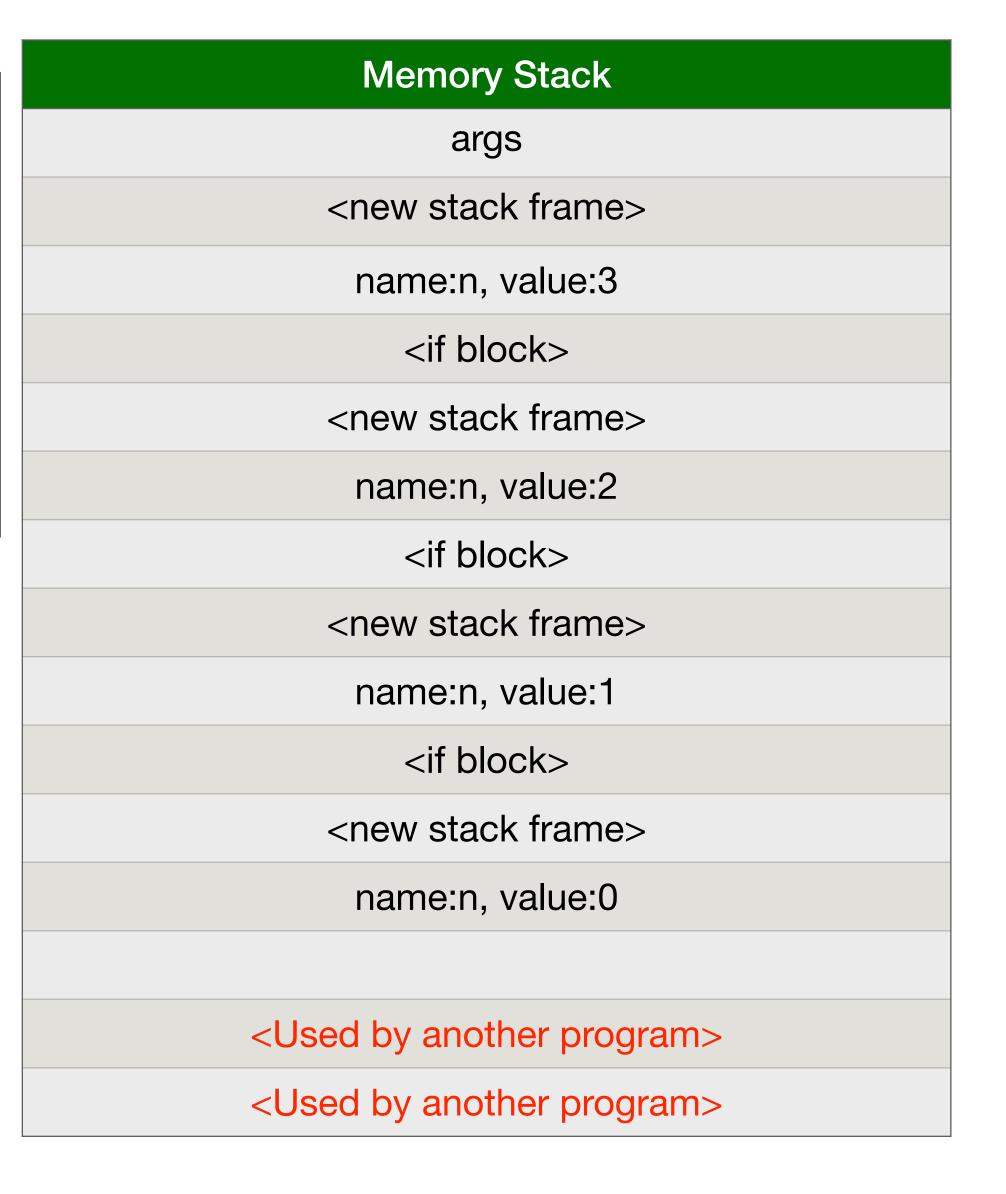
- Repeat, repeat
- Many variables named n on the stack
- Each is in different frame so it's ok

Memory Stack
args
<new frame="" stack=""></new>
name:n, value:3
<if block=""></if>
<new frame="" stack=""></new>
name:n, value:2
<if block=""></if>
<new frame="" stack=""></new>
name:n, value:1
<if block=""></if>
<new frame="" stack=""></new>
name:n, value:0
<used another="" by="" program=""></used>
<used another="" by="" program=""></used>

```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
}else{
   0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

- Conditional finally false
- return 0



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

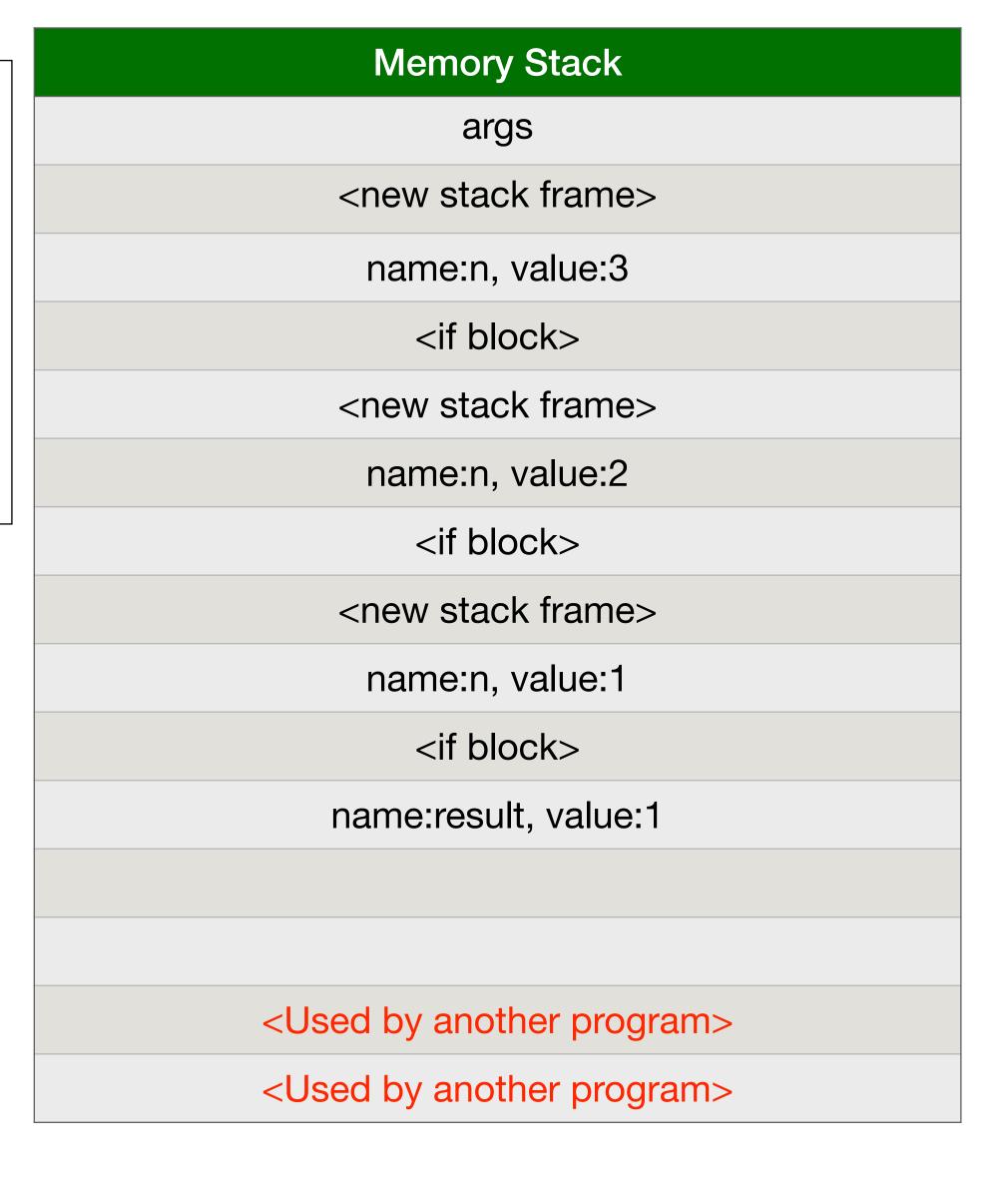
Assign return value to result

Memory Stack
args
<new frame="" stack=""></new>
name:n, value:3
<if block=""></if>
<new frame="" stack=""></new>
name:n, value:2
<if block=""></if>
<new frame="" stack=""></new>
name:n, value:1
<if block=""></if>
name:result, value:0
<used another="" by="" program=""></used>
<used another="" by="" program=""></used>

```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
} else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

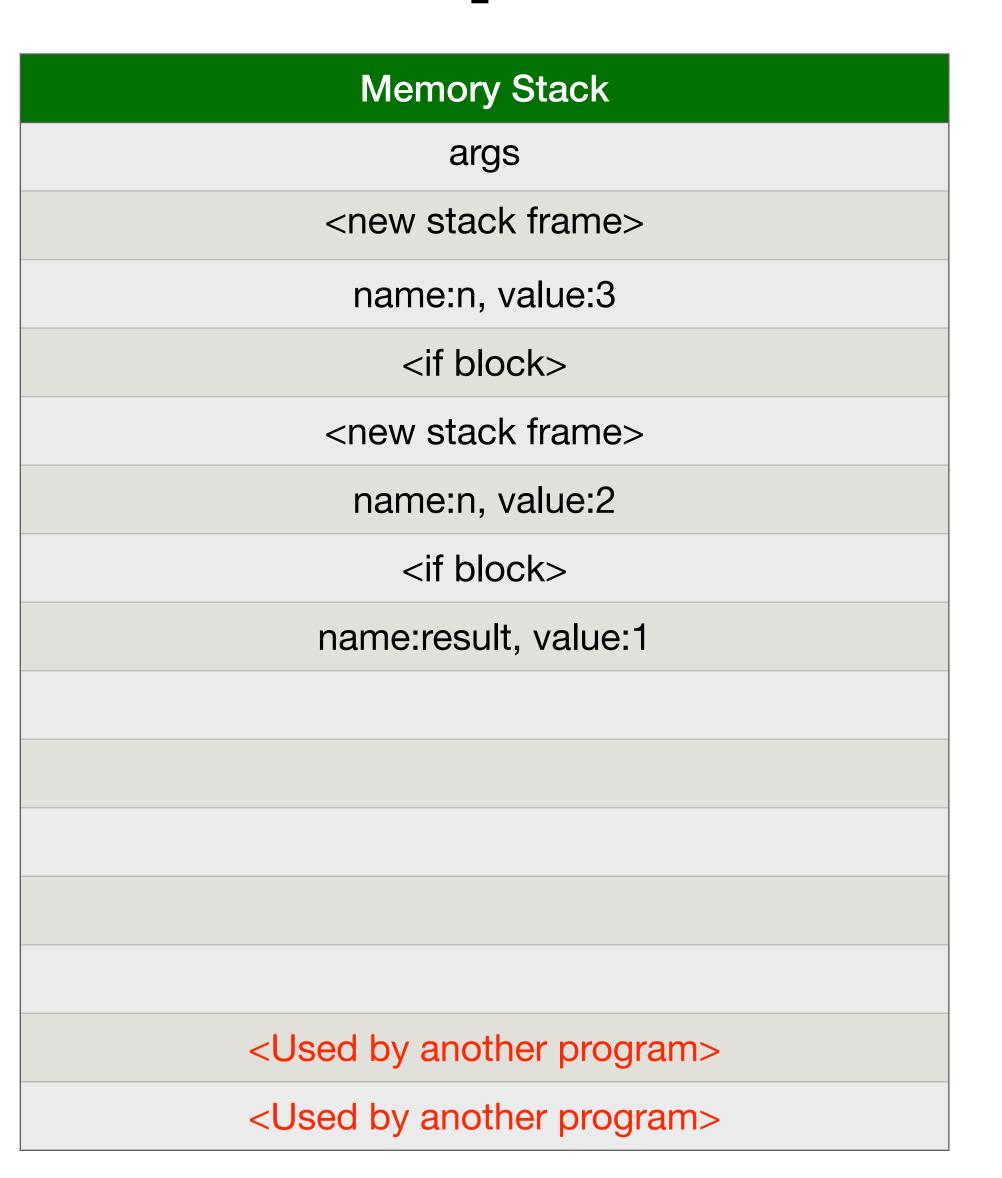
- Add value of the n in this stack frame to result
- result is the last expression and is returned



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

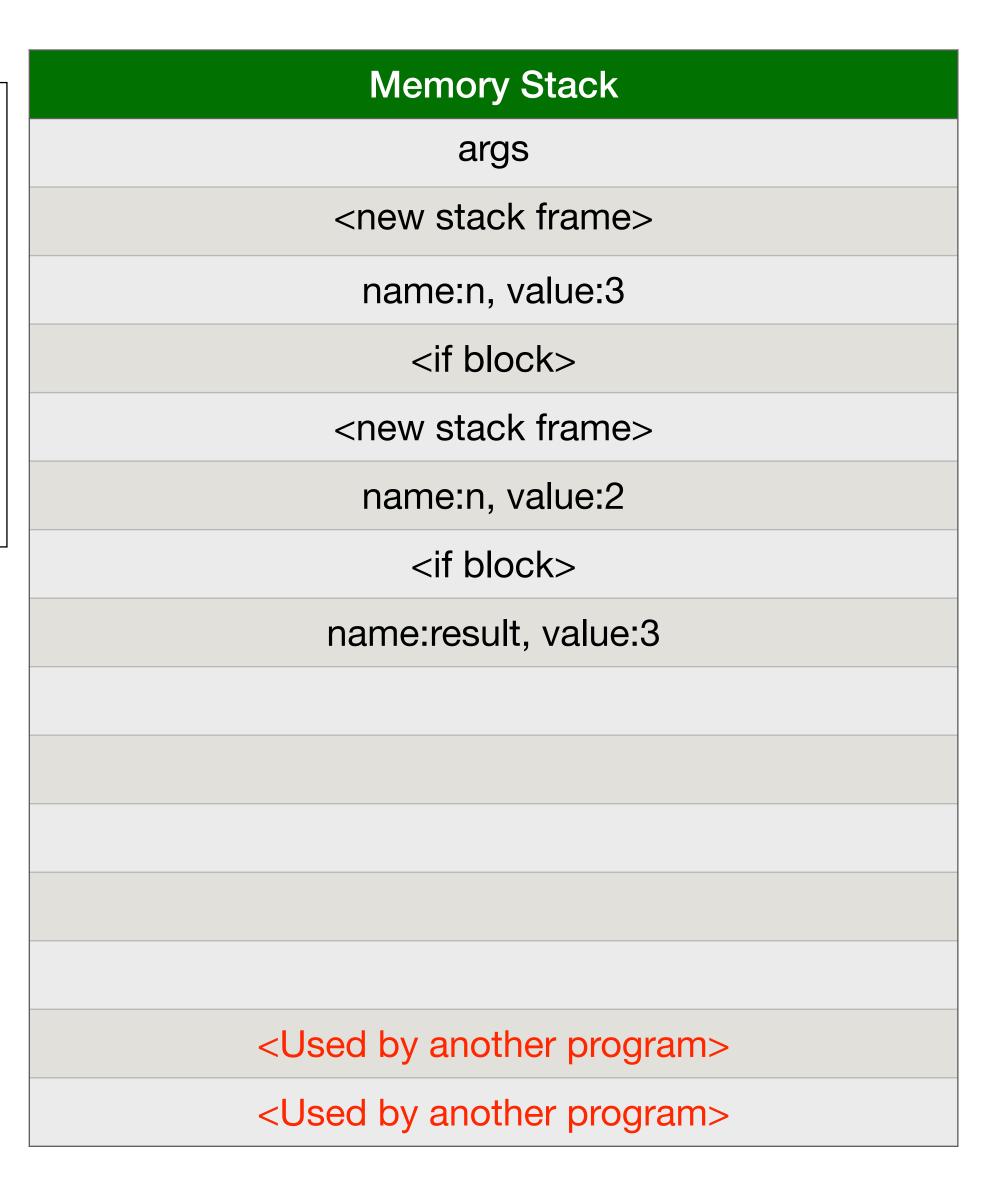
- Return to function call from previous frame
- Store return value in result



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
   result
}else{
    0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

- Add value of n from this frame..
- Repeat



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

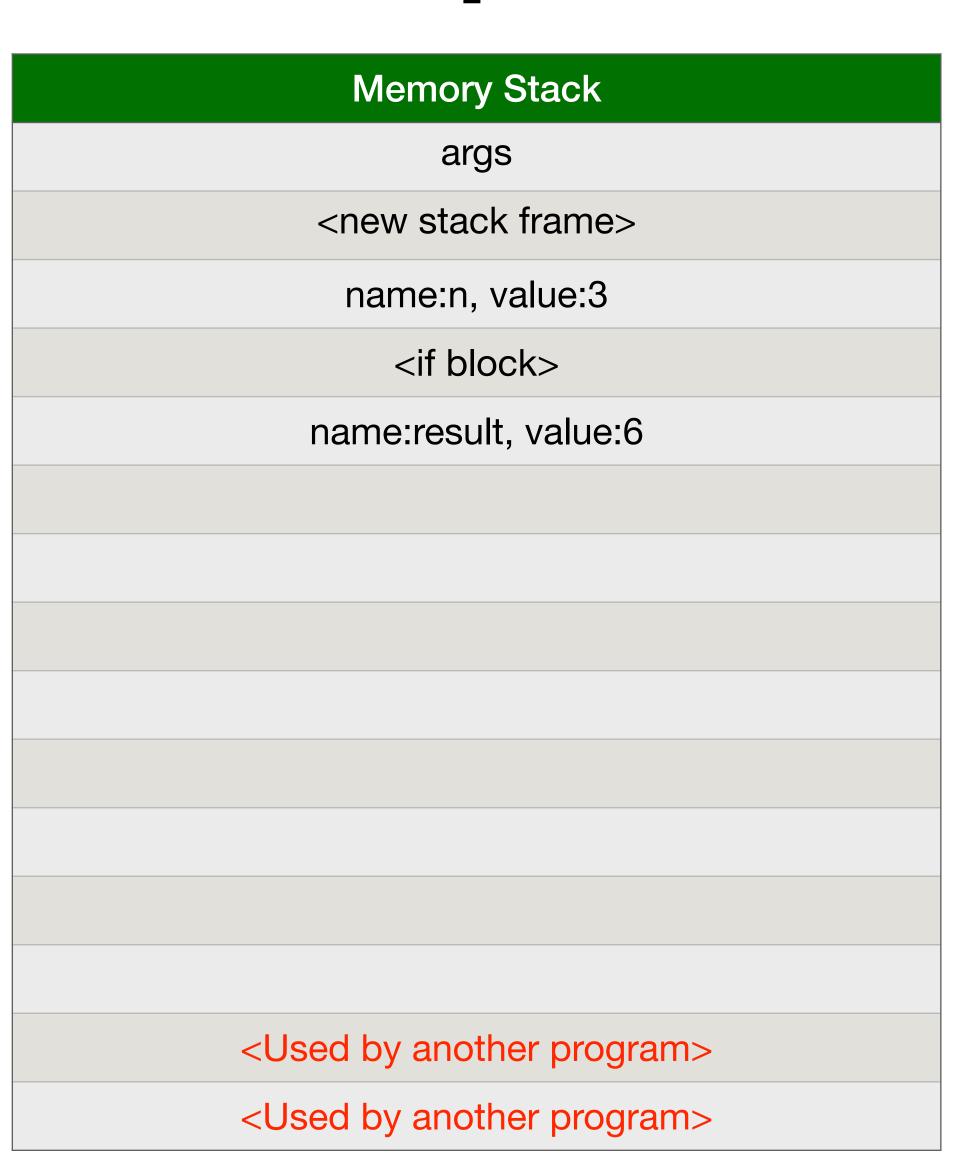
- Add value of n from this frame..
- Repeat



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

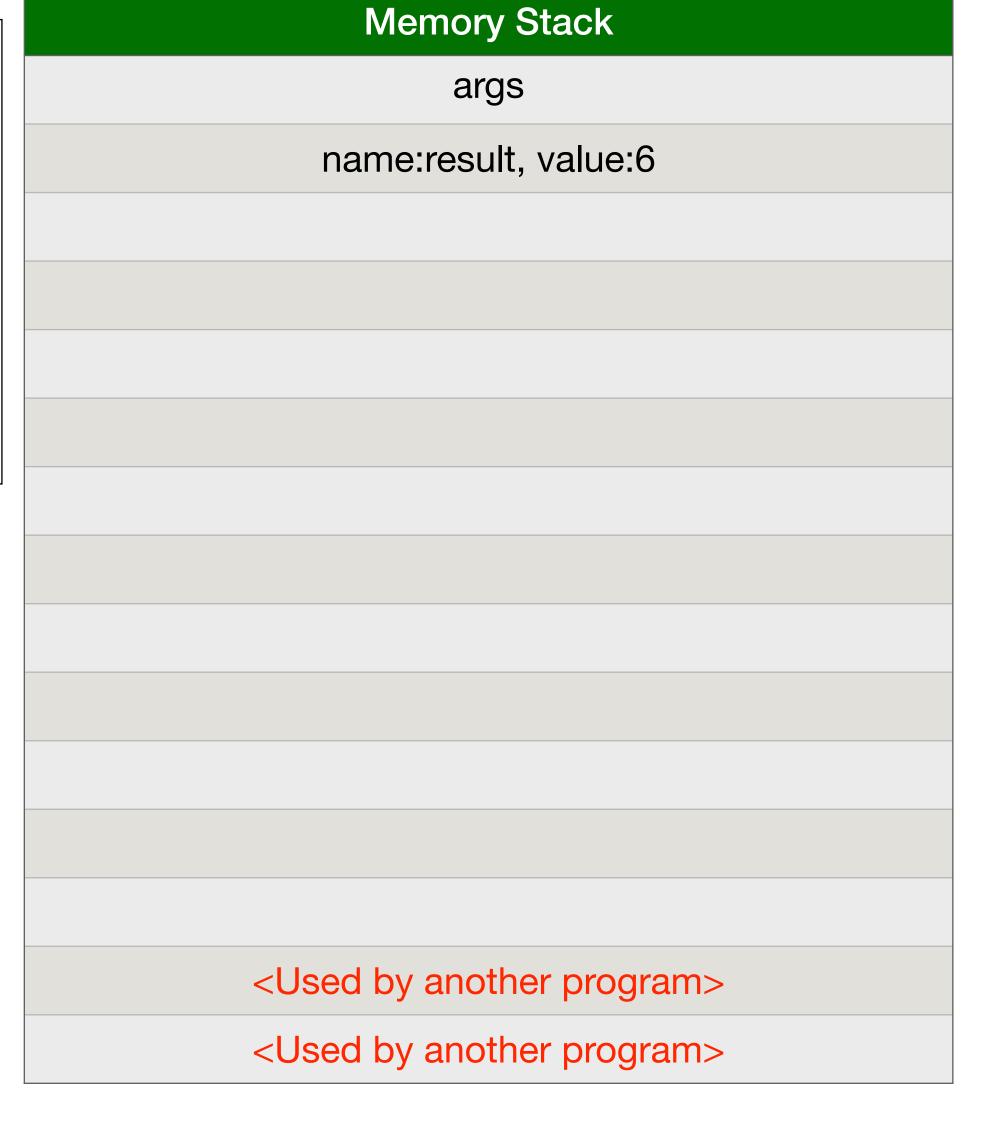
- And repeat...
- Imagine if the original input were 1000
 - This is why we use computers



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

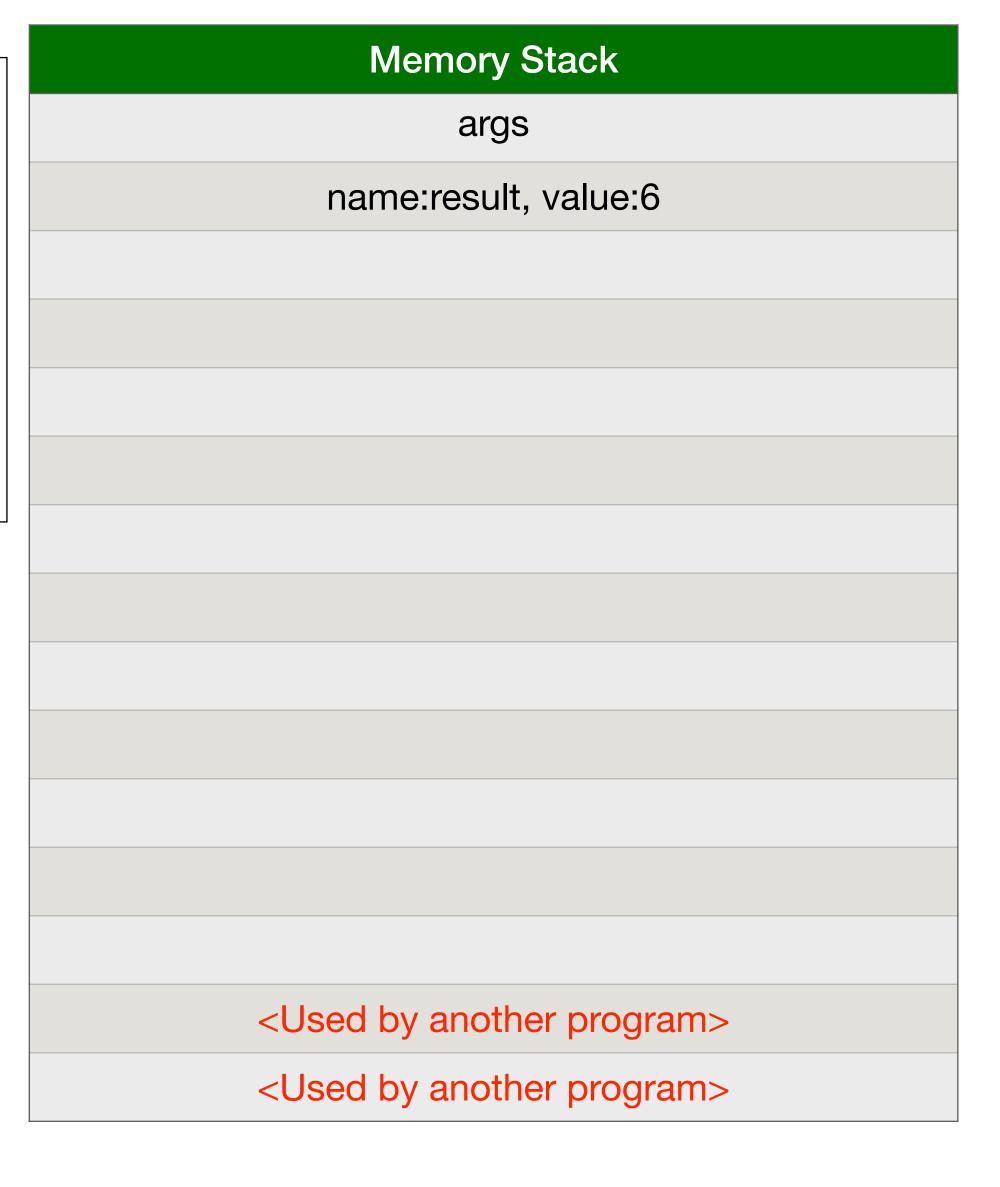
 Value result in main method gets the last return value



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

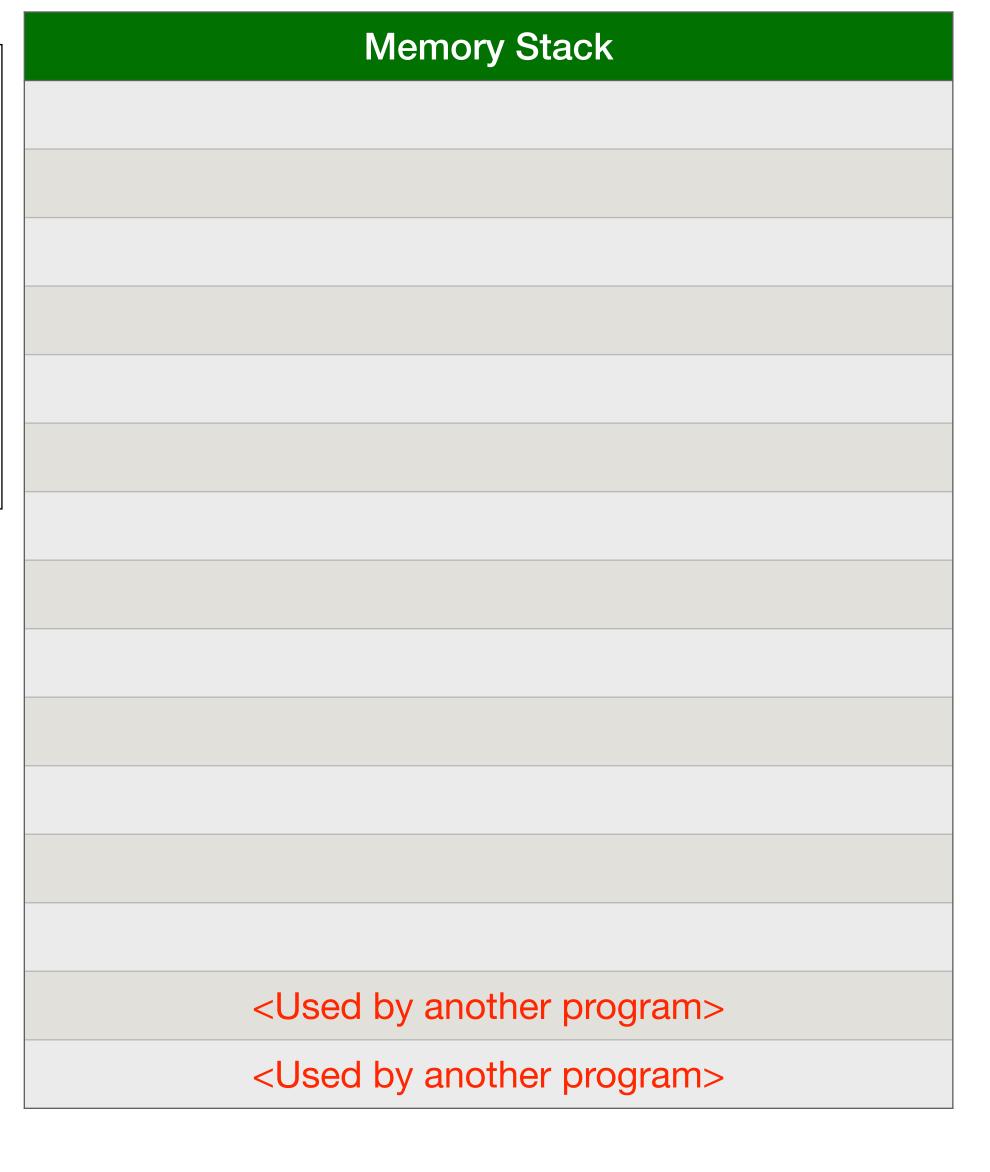
print 6



```
def computeGeometricSum(n: Int): Int ={
   if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
   }else{
      0
   }
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

Free memory



More Memory Examples

- We were close to the end of the stack on that example
 - In reality, the stack will be much larger than in this example
- What if this were our code?

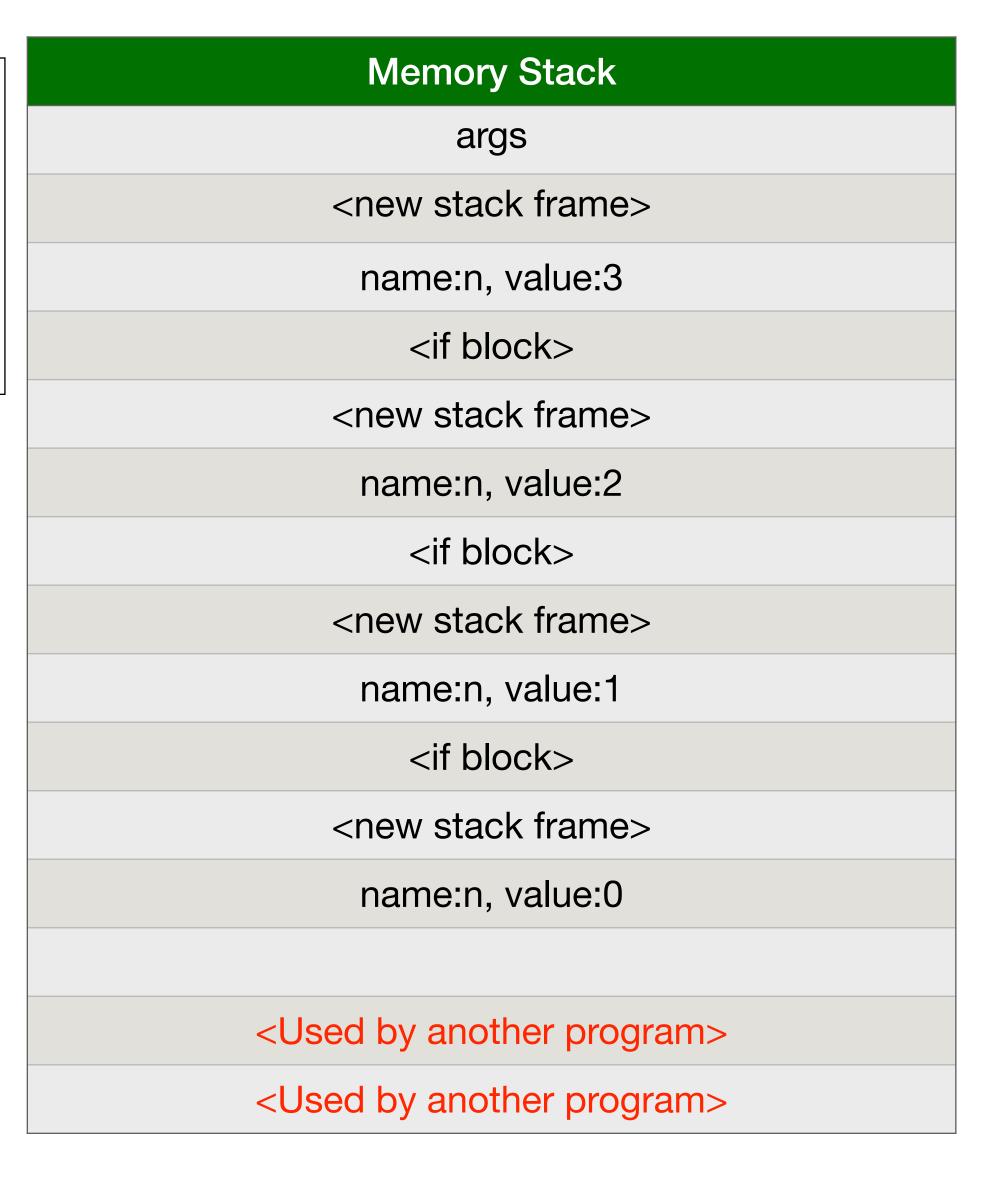
```
def computeGeometricSum(n: Int): Int ={
   var result: Int = computeGeometricSum(n - 1)
   result += n
   result
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

```
def computeGeometricSum(n: Int): Int ={
   var result: Int = computeGeometricSum(n - 1)
   result += n
   result
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

 At this point the other program was going to return 0 and return back up the stack



```
def computeGeometricSum(n: Int): Int ={
   var result: Int = computeGeometricSum(n - 1)
   result += n
   result
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```

 This program keeps adding frames to the stack

Memory Stack
args
<new frame="" stack=""></new>
name:n, value:3
<new frame="" stack=""></new>
name:n, value:2
<new frame="" stack=""></new>
name:n, value:1
<new frame="" stack=""></new>
name:n, value:0
<new frame="" stack=""></new>
name:n, value:-1
<new frame="" stack=""></new>
name:n, value:-2
<used another="" by="" program=""></used>
<used another="" by="" program=""></used>

```
def computeGeometricSum(n: Int): Int ={
   var result: Int = computeGeometricSum(n - 1)
   result += n
   result
}

def main(args: Array[String]): Unit = {
   val result: Int = computeGeometricSum(3)
   println(result)
}
```



- STACK OVERFLOW
- Program crashes



Memory Stack
args
<new frame="" stack=""></new>
name:n, value:3
<new frame="" stack=""></new>
name:n, value:2
<new frame="" stack=""></new>
name:n, value:1
<new frame="" stack=""></new>
name:n, value:0
<new frame="" stack=""></new>
name:n, value:-1
<new frame="" stack=""></new>
name:n, value:-2
<used anotknewstacknframe="" by=""></used>
<used anothame="" by="" ogrvatue:-3<="" td=""></used>

Debugger Demo

Lecture Task

- This is Lecture Task 6 from the Pale Blue Dot project -
- **Testing**: In the tests package, complete the test suite named Lecture Task6.
 - This suite will test a method in the PaleBlueDot object named "closestCity" that takes a String and a List of Doubles as parameters and returns a List of Strings
 - The inputs represent the filename for the cities file and a latitude/longitude location. This method outputs the country code, name, and region of the city closest to the input location. The upper/lower-case and formatting should match exactly as it appears in the file.
 - Example output format: List("cn", "longjiang", "08")
 - If no closest city is found, the method outputs the empty list: List()
- Functionality: Implement the functionality for closestCity.