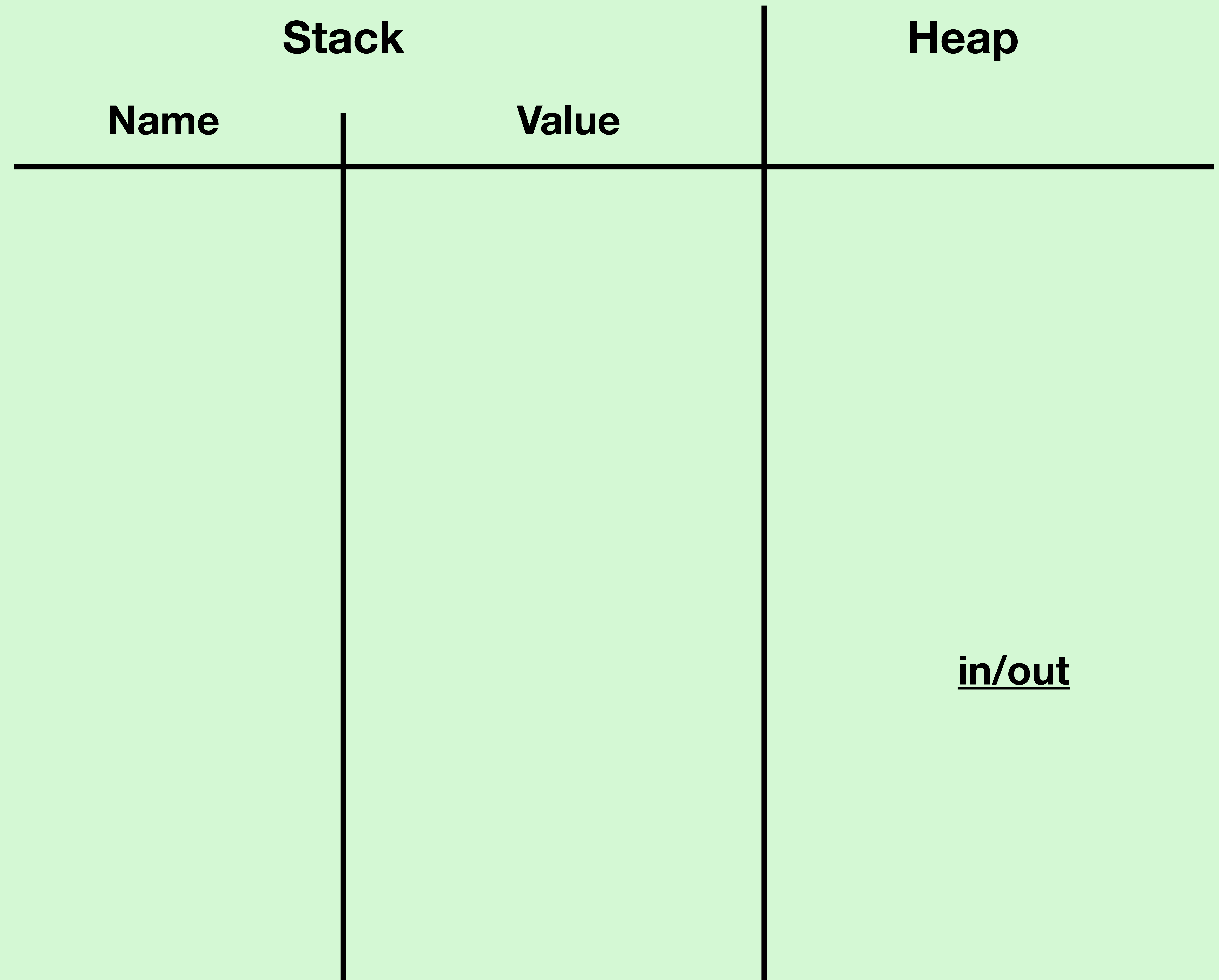
Memory Diagrams

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

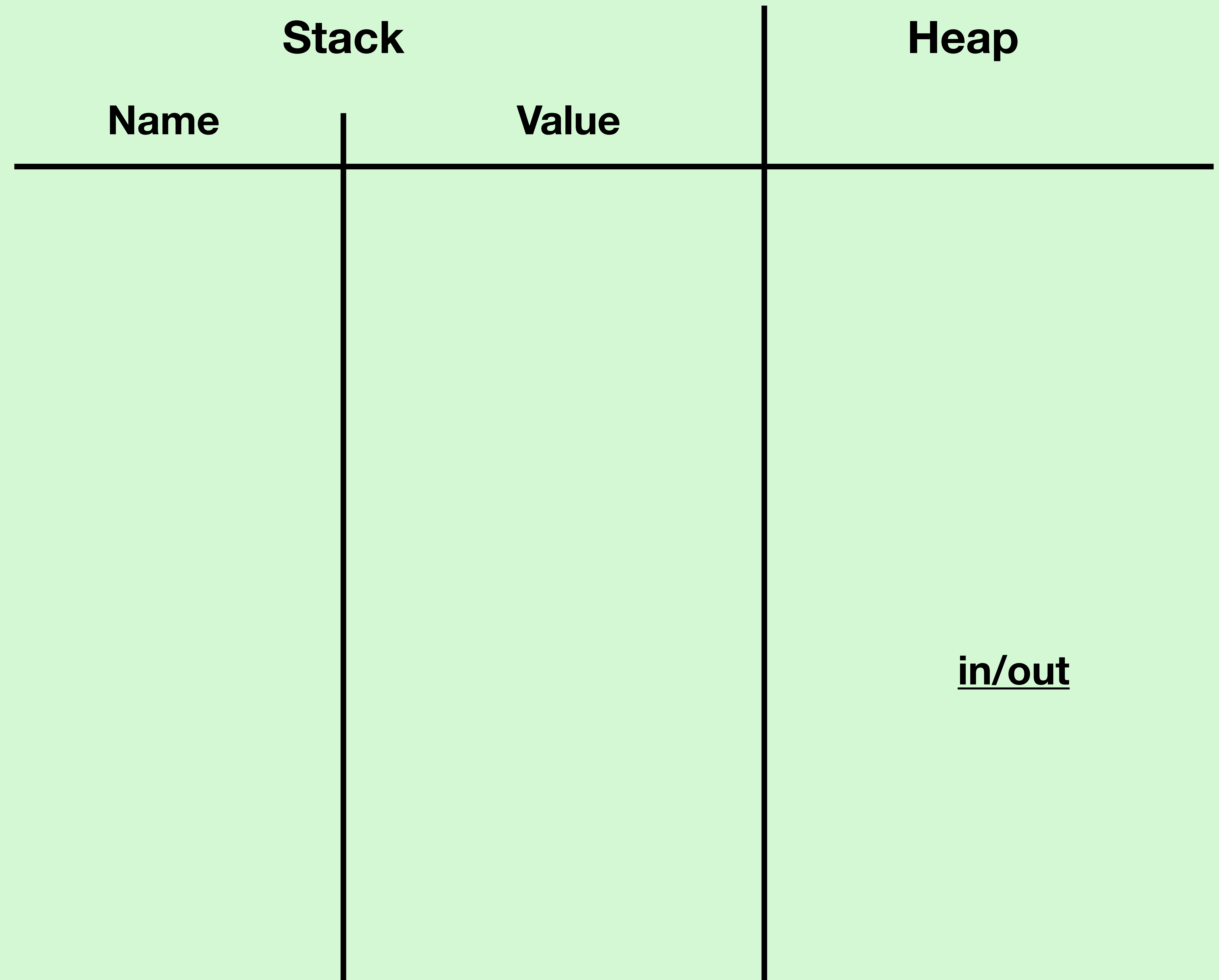


- Start by setting up the diagram
- Separate columns for stack and heap memory

Scope Example

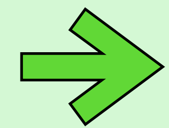
```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Separate the stack into name and value



Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



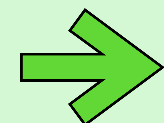
- Start tracing the program from main
- First 2 lines add variables to the stack

Stack		Heap
Name	**Value**	
x	5	
y	2	

in/out

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



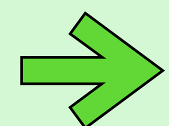
Stack		Heap
Name	**Value**	
x	5	
y	2	
z		

in/out

- We call a method named subtract
  - Add the return variable to the stack

Scope Example

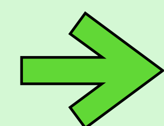
```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- Draw a solid box for the **Stack Frame**
- Arrow to the return variable and name of method being called

Scope Example

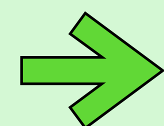
```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- The stack frame cannot be crossed
- Can't access variables across the solid box

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



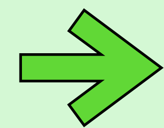
- We say variables outside the current stack frame are **out of scope**

Stack		Heap
Name	Value	
x	5	
y	2	
z		

subtract

in/out

Scope Example

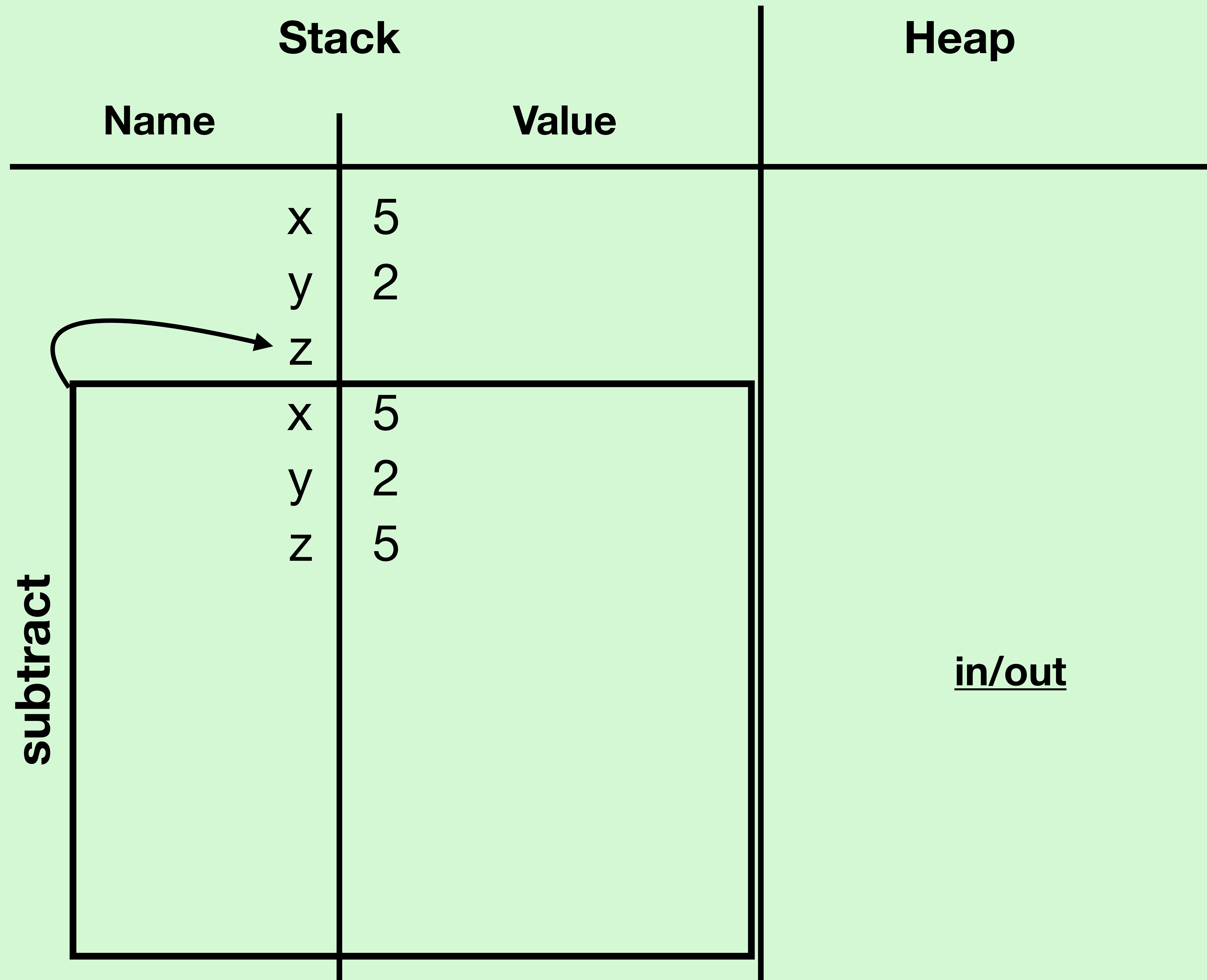


```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Add the names of the parameters with the values of the arguments inside the stack frame

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

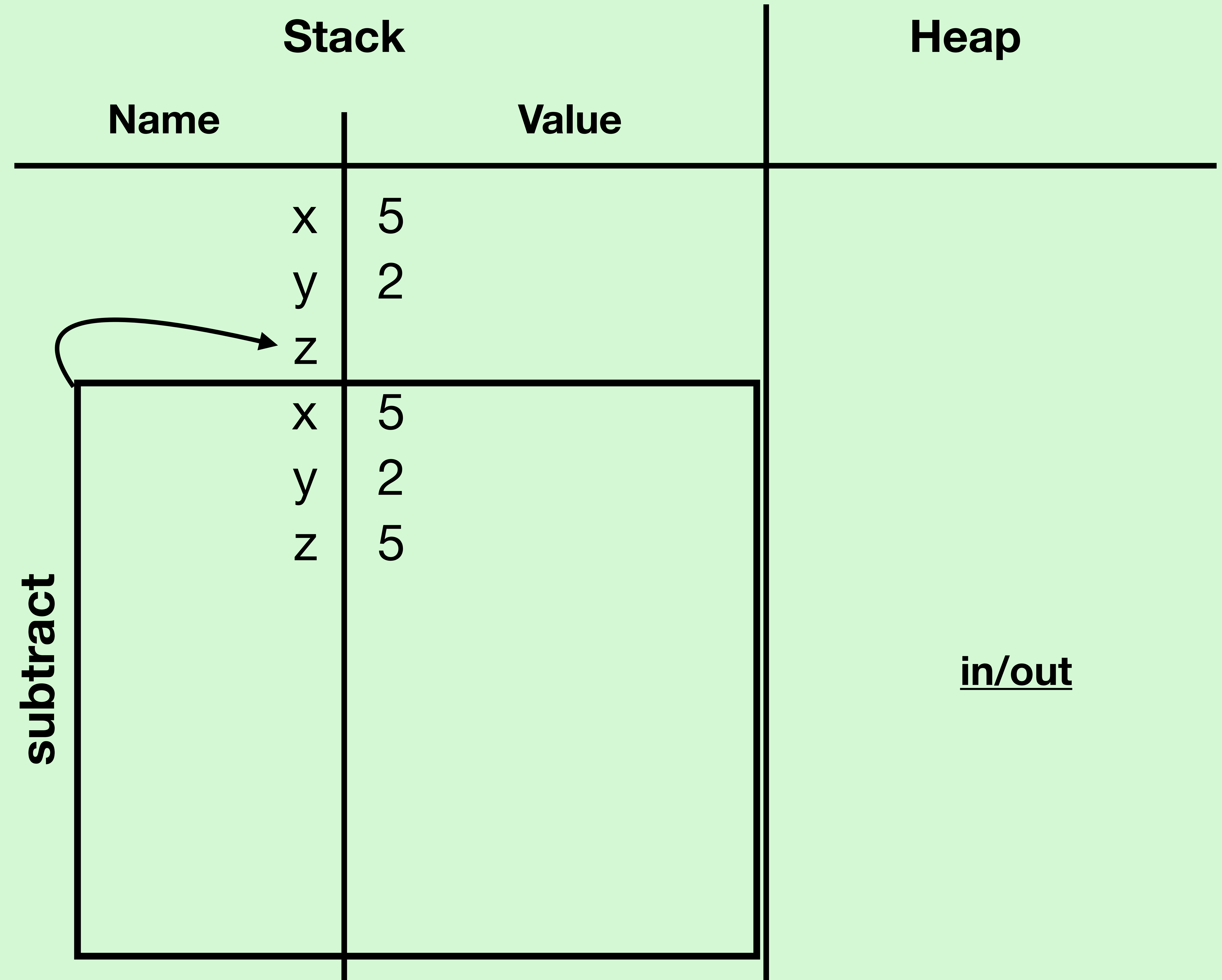


- Add z equal to the value of x
- Two x's on the stack
  - Only 1 inside this stack frame!

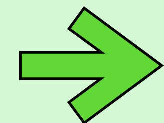
Scope Example

```scala
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Can reuse variable names in different stack frames



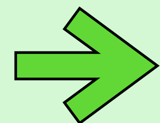
Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Start of a code block for a loop
- New code block whenever there are { } that do not define a method

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Draw a dotted box for code blocks
- Dotted lines can *sometimes* be crossed

Stack

Name	Value
x	5
y	2
z	
x	5
y	2
z	5
i	0

subtract

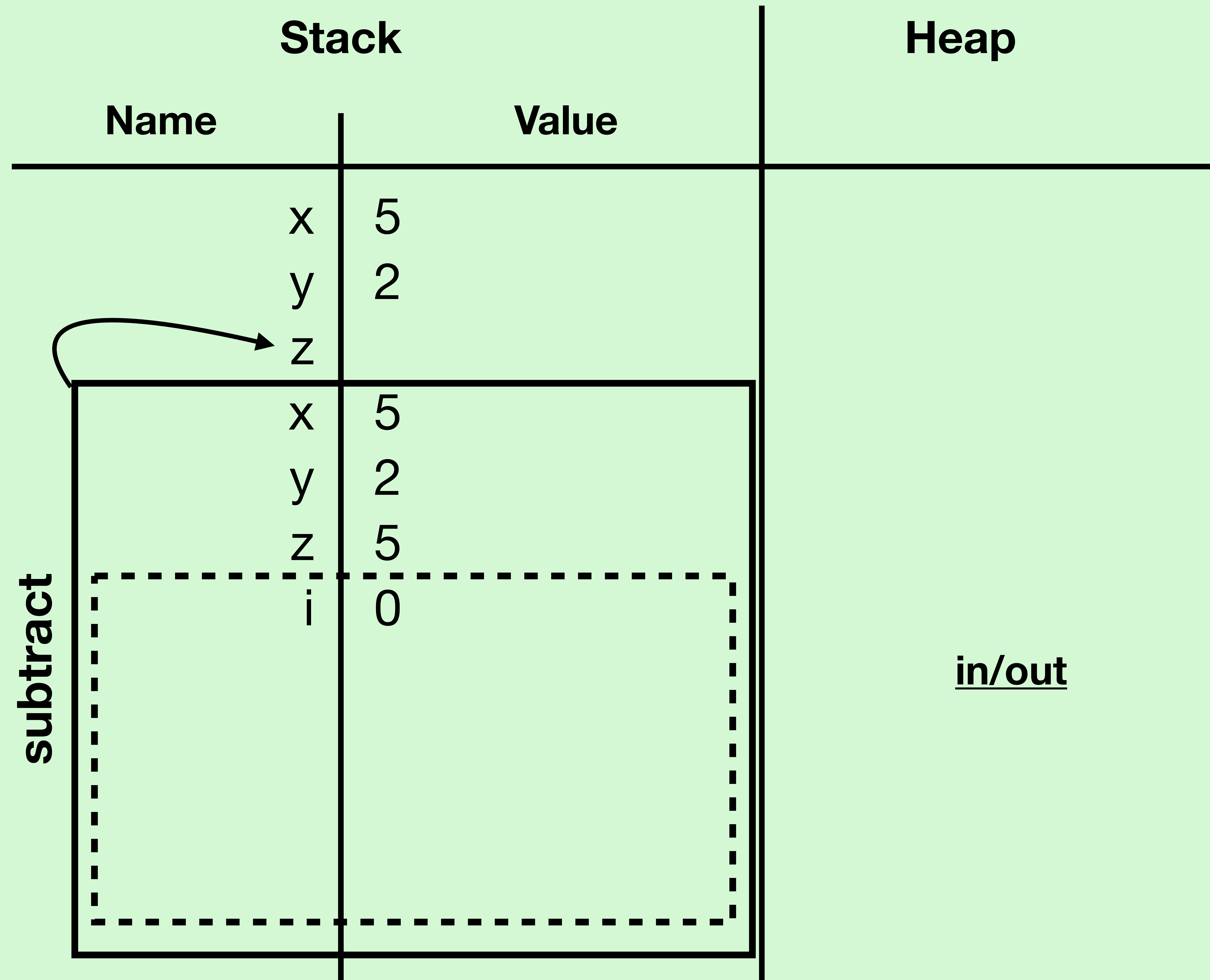
Heap

in/out

Scope Example

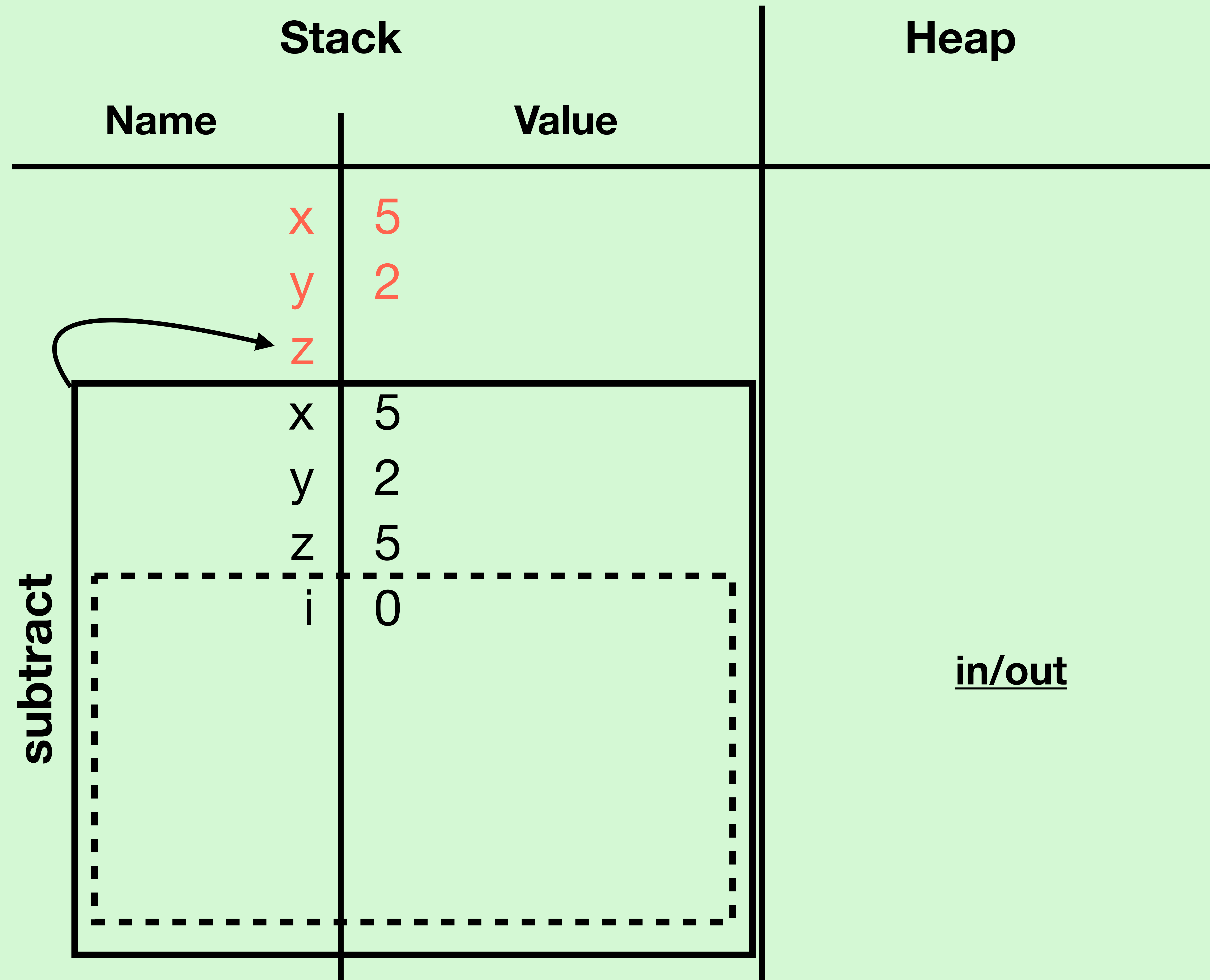
```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Code blocks affect **Variable Scope**
- Variable scope determines which variables can be accessed



Scope Example

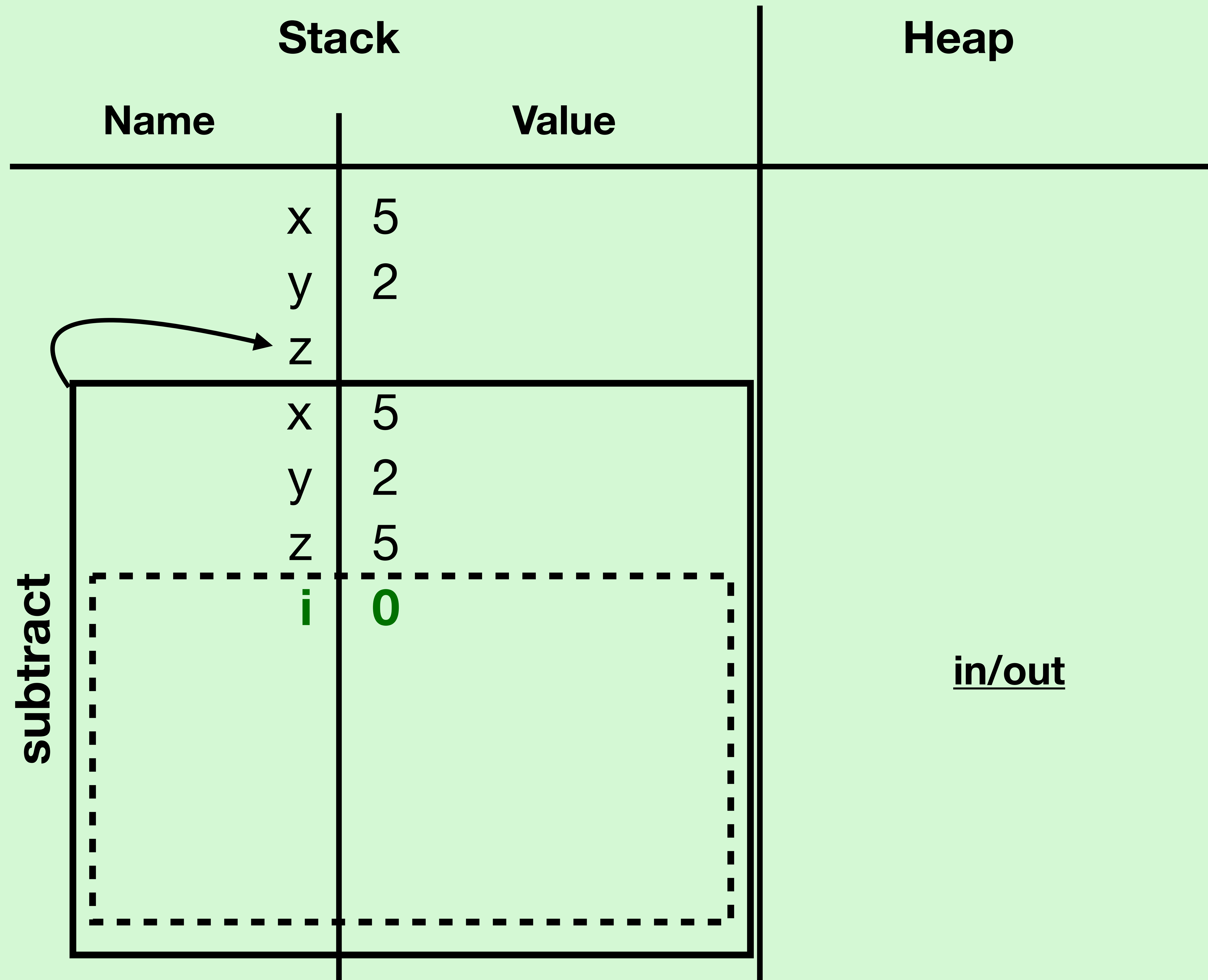
```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- Variables in another stack frame are always **out of scope**
- Cannot access variables across stack frames

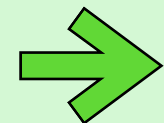
Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- Variables in the current code block are always **in scope**
- Can always be accessed

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Variable in the current stack frame, but not in the current code block, are *usually* **in scope**
  - Except when another variable with the same name is already in scope

Stack

Name	Value
x	5
y	2
z	
x	5
y	2
z	5
i	0

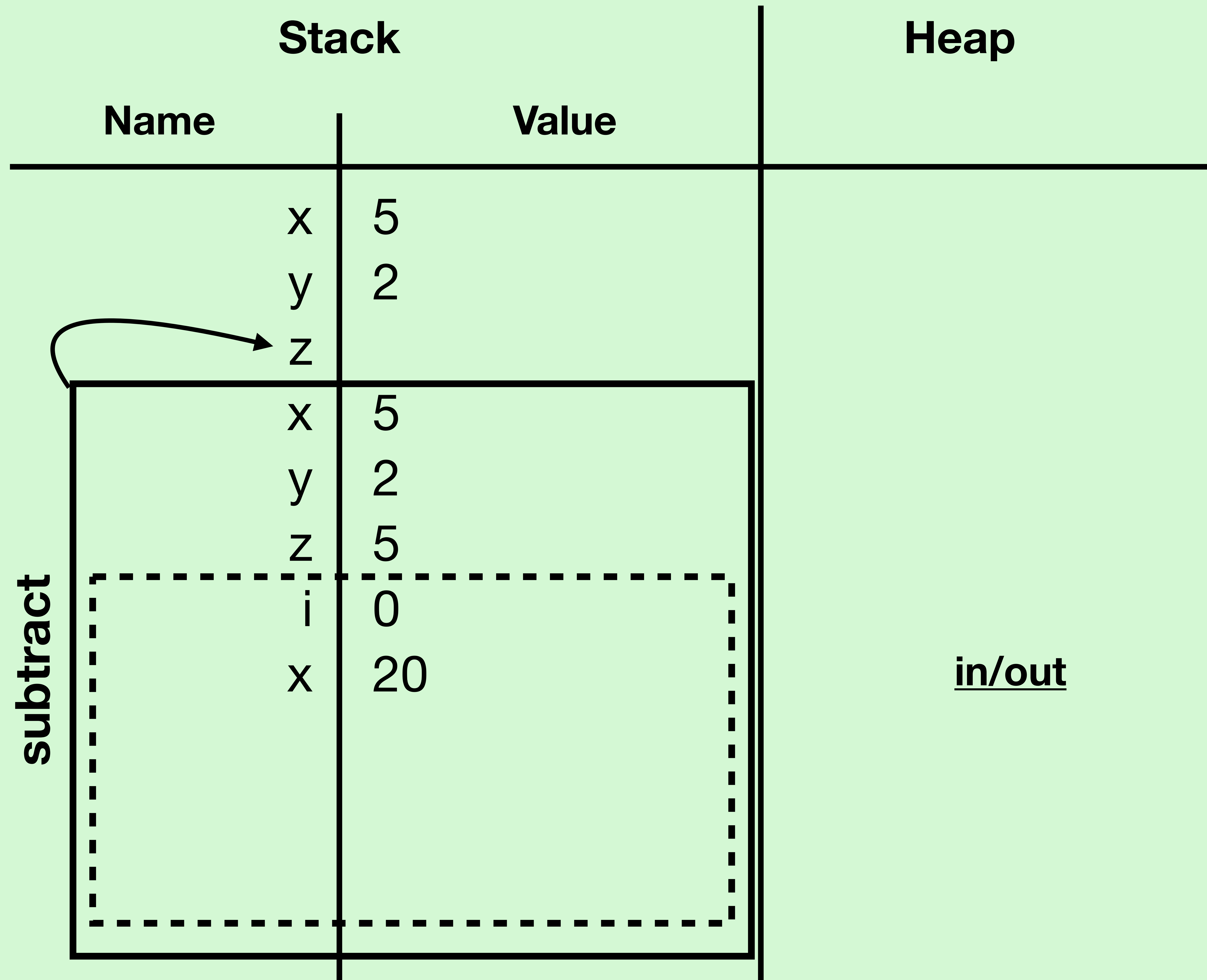
subtract

Heap

in/out

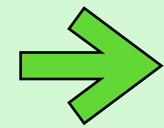
Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- Whenever val or var are used, a new variable is being added to the stack
- We can have multiple variables with the same name in the same stack frame!

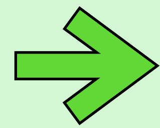
Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Declare a new variable x inside the code block

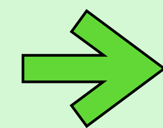
Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Since variables in the current code block are always in scope, the x with 20 will be accessed if x is used
- The other 2 x's cannot be accessed from inside this block

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Add another x to the stack in a new code block
- Now this is the only x in scope

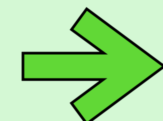
Stack | **Heap**

Name	Value
x	5
y	2
z	
x	5
y	2
z	5
i	0
x	20
x	1

subtract

in/out

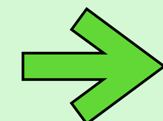
Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```


- z -= x
- But we have 4 x's on the stack!
- Which one is used??

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

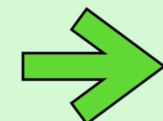
- Use the one in the inner-most code block
- The current code block has an x, use that one

Stack		Heap
Name	Value	
x	5	
y	2	
z		
x	5	
y	2	
z	5	
i	0	
x	20	
x	1	

subtract

in/out

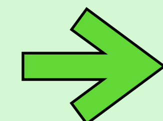
Scope Example



```scala
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```


- We're also accessing z
- Current code block does not contain a z

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

Stack

Name	Value
x	5
y	2
z	
x	5
y	2
z	5
i	0
x	20
x	1

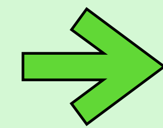
subtract

Heap

in/out

- Continue searching in the next code block
- Still no z...

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

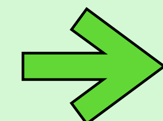
- Keep searching code blocks until we reach the stack frame
- Found a z! Use it.

Stack		Heap
Name	**Value**	
x	5	
y	2	
z		
x	5	
y	2	
z	**5**	
i	0	
x	20	
x	1	

subtract

in/out

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Subtract 1 from z
- When a value changes, cross out the old value and write the new one

Stack

Name	Value
x	5
y	2
z	
x	5
y	2
z	~~5~~ 4
i	0
x	20
x	1

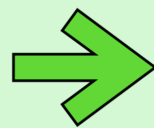
subtract

Heap

in/out

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



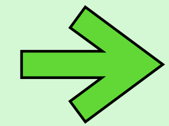
- End of the if/else code block
  - Cross it out
- The x with value 1 is no longer on the stack and the x with 20 is back in scope

Stack		Heap
Name	**Value**	
x	5	
y	2	
z		
x	5	
y	2	
z	~~5~~ 4	
i	0	
x	20	
~~x~~	~~1~~	

subtract

in/out

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Advance the iteration variable
- Run the loop body again

Stack

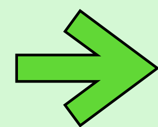
Name	Value
x	5
y	2
z	
x	5
y	2
z	~~5~~ 4
i	~~0~~ 1
x	20
~~x~~	~~1~~

subtract

Heap

in/out

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

Stack | **Heap**

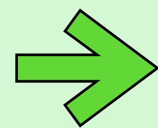
Name	Value
x	5
y	2
z	
x	5
y	2
z	~~5~~ 4
i	~~0~~ 1
x	~~20~~ 20
~~x~~	~~1~~

subtract

in/out

- Declare a new value x
- When inside a loop, use the same line and update the value

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Add a new code block for the conditional

Stack

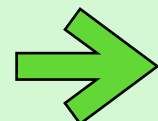
Name	Value
x	5
y	2
z	
x	5
y	2
z	~~5~~ 4
i	~~0~~ 1
x	~~20~~ 20
~~x~~	~~1~~
x	1

subtract

Heap

in/out

Scope Example



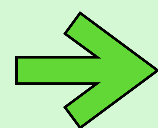
```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

Stack		Heap
Name	Value	
x	5	
y	2	
z		
x	5	
y	2	
z	~~5~~ 4 3	
i	~~0~~ 1	
x	~~20~~ 20	in/out
~~x~~	~~1~~	
x	1	

subtract

- Update z
- Found an x with value 1 in the inner-most block

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

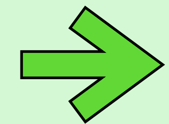
- End of the if/else block
- Cross it out to show that x with value 1 is no longer in memory

Stack		Heap
Name	**Value**	
x	5	
y	2	
z		
x	5	
y	2	
z	~~5~~ 4 3	
i	~~0~~ 1	
x	~~20~~ 20	
~~x~~	~~1~~	
~~x~~	~~1~~	

subtract

in/out

Scope Example



```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- End of the loop block
  - i is no longer on the stack
  - x with value 5 is back in scope

Stack | **Heap**

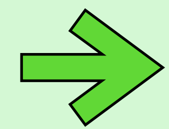
Name	Value
x	5
y	2
z	3
x	5
y	2
z	~~5~~ 4 3
i	~~0~~ ~~1~~
x	~~20~~ 20
x	1
x	1

subtract

in/out

Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      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 = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- Print 3 to the screen
- End of program

Another One

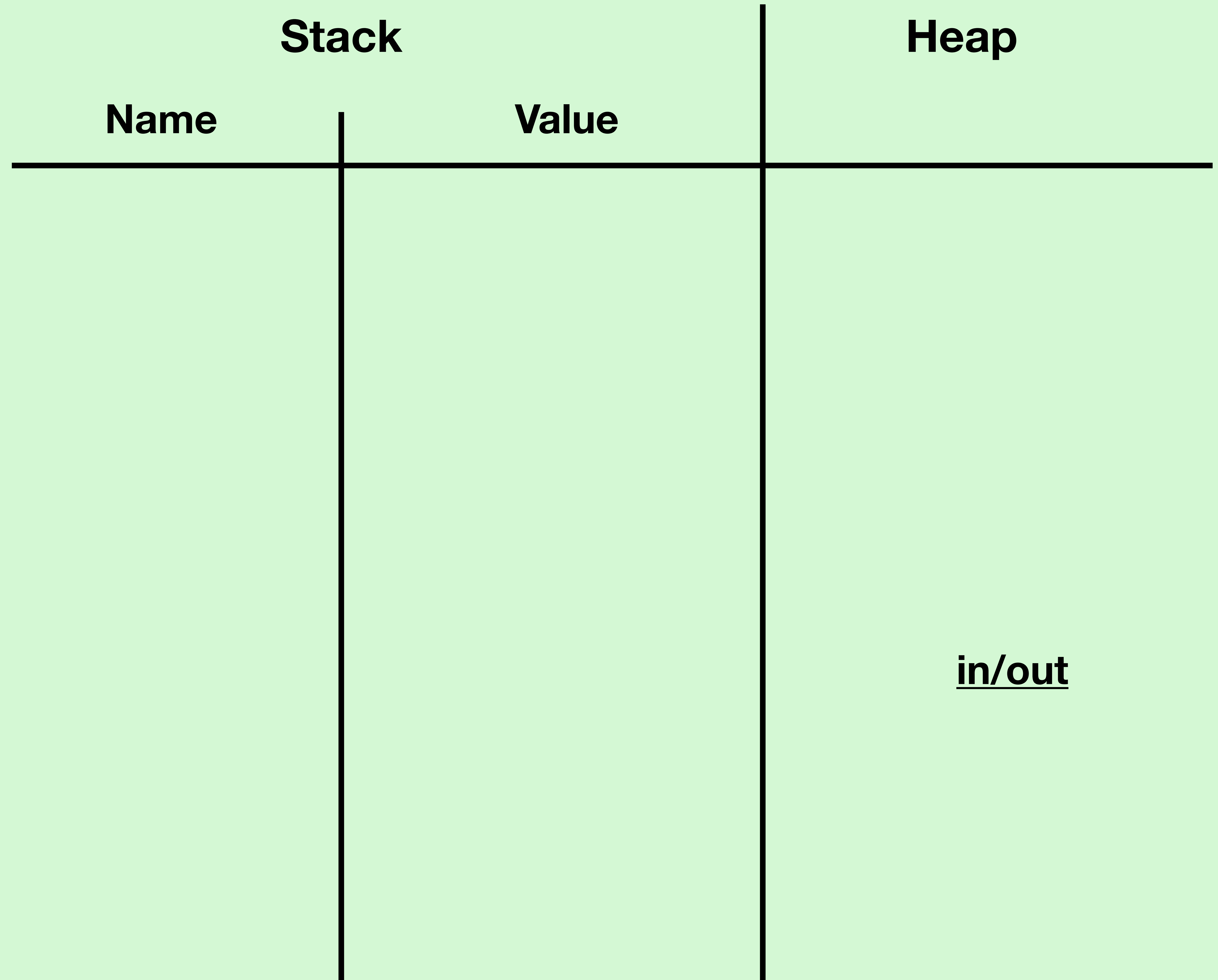
Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Start by setting up the diagram
- Separate columns for stack and heap memory



Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Separate the stack into name and value

Stack		Heap
Name	Value	
		in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Add a section for inputs and outputs
  - This is where you write what's printed to the screen

Stack		Heap
Name	Value	

in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Start execution at the main method

Stack

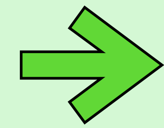
Name	Value

Heap

in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```



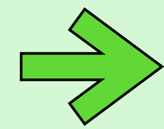
- Variable declaration
- Add the name and value to the stack

Stack		Heap
Name	Value	
testInput	"true;false"	

in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```



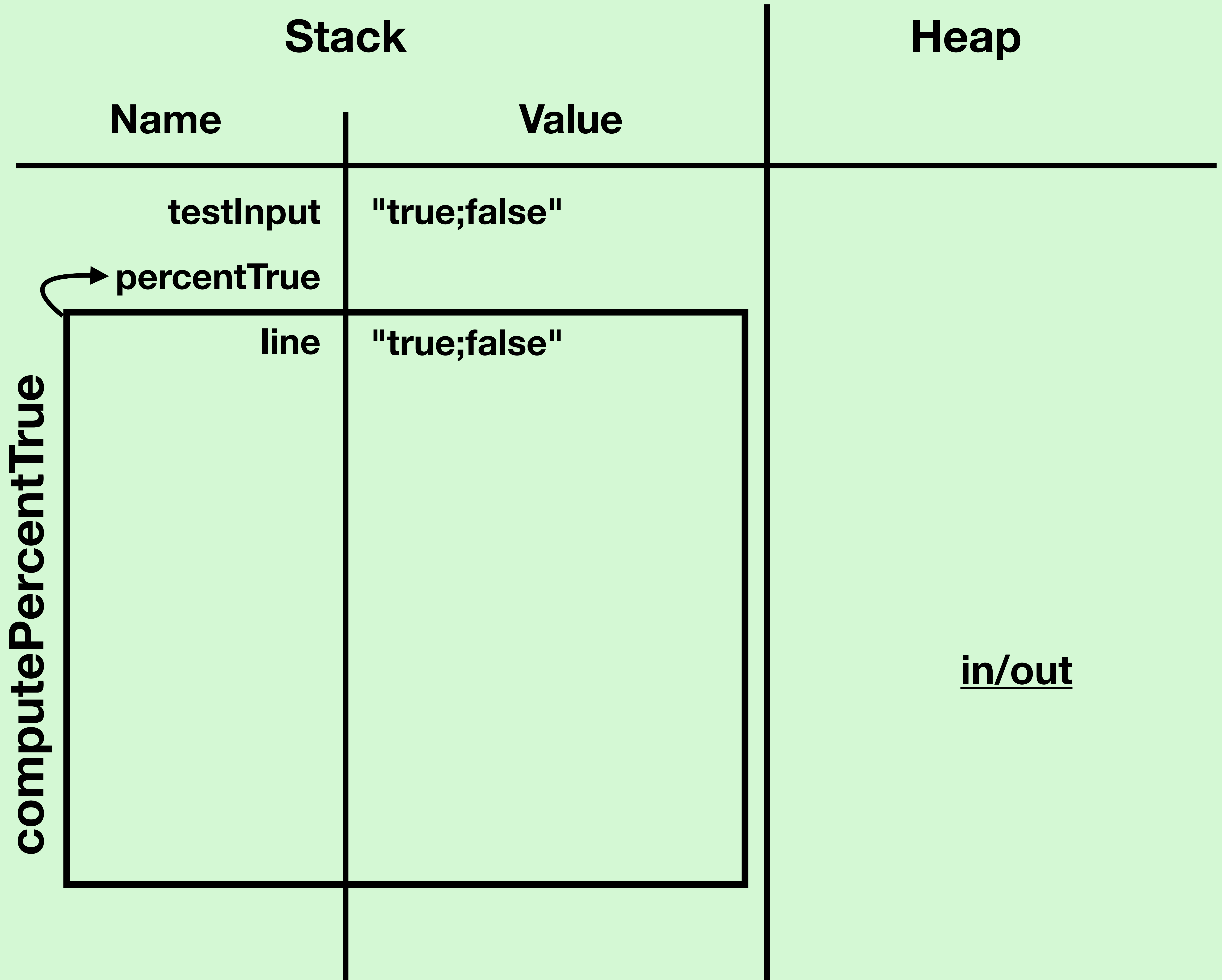
- Method call creates a new stack frame

Stack		Heap
Name	**Value**	
testInput	"true;false"	
percentTrue		
computePercentTrue (frame)		
		in/out

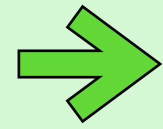
Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Add the parameter to the stack



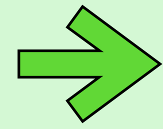
Array Example



```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Call split to separate the String into an Array
- We will not draw stack frames for library calls

Array Example



```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Arrays are stored on the heap
- Somewhere in memory separate from the stack

Stack

Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	

computePercentTrue

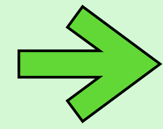
Heap

"true"	"false"

0x500

in/out

Array Example

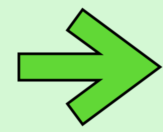


```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- We only know the memory address (reference) where we can find the Array
    - 0x500 in this example

	Stack		Heap
	Name	Value	
	testInput	"true;false"	"true" "false"
	percentTrue		0x500
computePercentTrue	line	"true;false"	
	splits		
			in/out

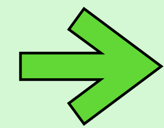
Array Example



```scala
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- We only store a reference to the Array on the stack
- To access the Array, follow the reference
  - Visualized as an arrow

Array Example



```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Continue with the program
- Add trueCount to the stack

Stack

Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	0x500
trueCount	0.0

computePercentTrue

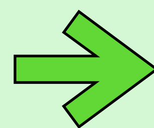
Heap

"true"	"false"

0x500

in/out

Array Example



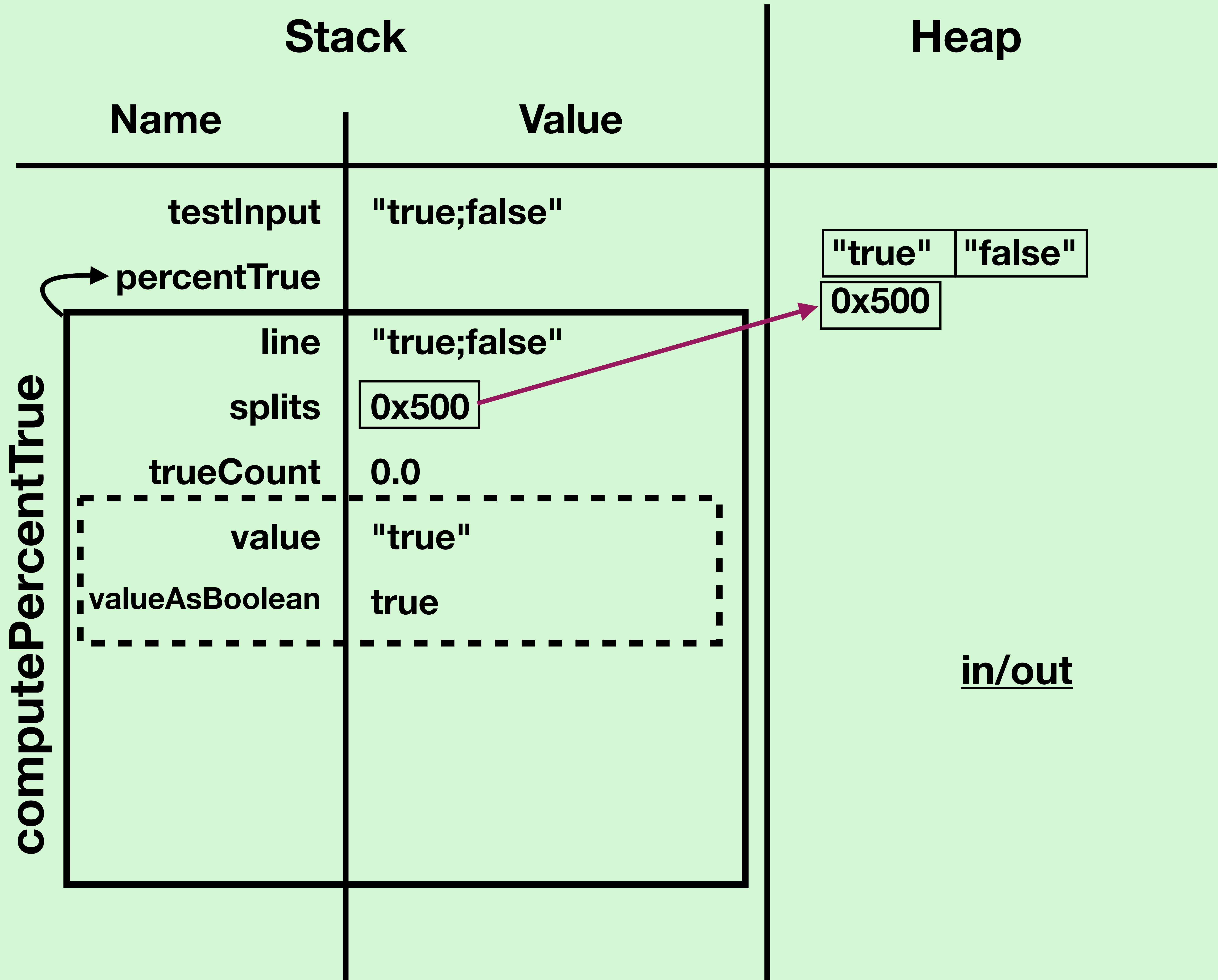
```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Add a code block for the for loop
- Iteration variable starts with the value at index 0 of the Array

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

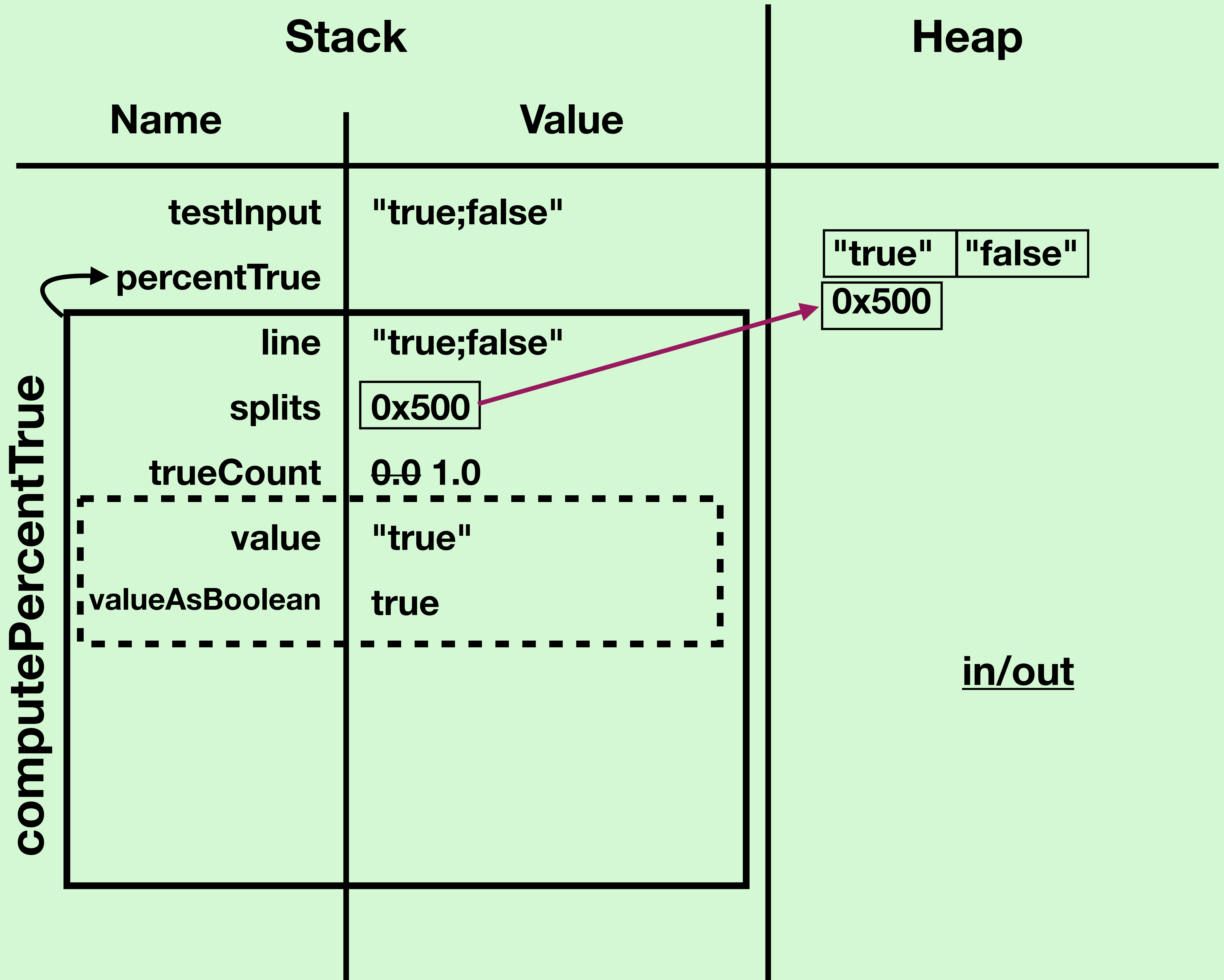
- Variable is declared inside the code block of the loop



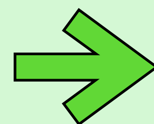
Array Example

```scala
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- If no variables are declared inside a block, you don't have to draw the block in your diagram



Array Example



```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Advance to the next value in the Array

Stack

Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	0x500
trueCount	~~0.0~~ 1.0
value	~~"true"~~ "false"
valueAsBoolean	true

computePercentTrue

Heap

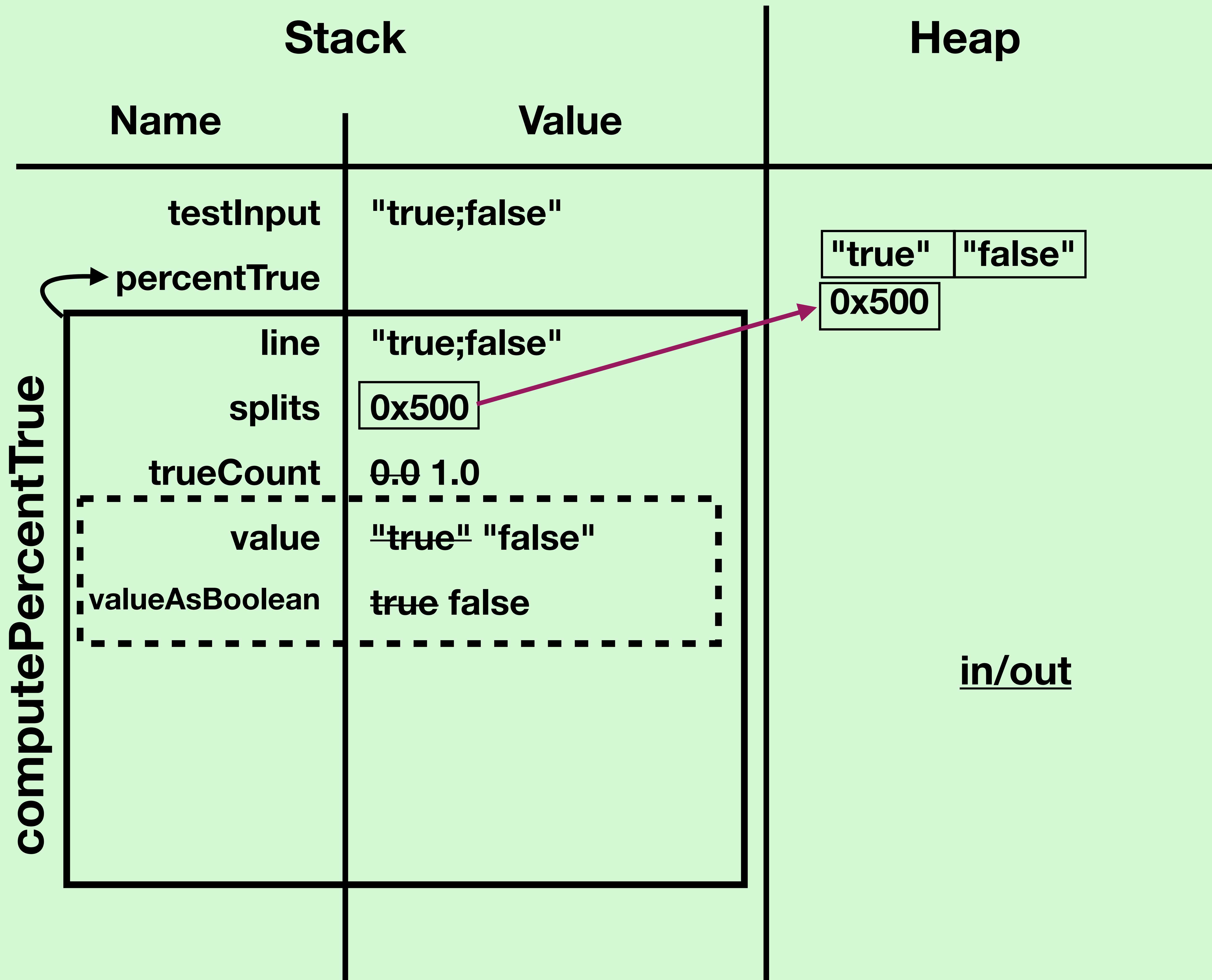
0x500: "true" "false"

in/out

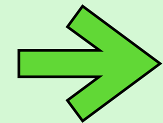
Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Declare a new valueAsBoolean by crossing out the old value and reusing the space from the old variable



Array Example



```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Declare a new valueAsBoolean by crossing out the old value and reusing the space from the old variable

Stack

Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	0x500
trueCount	~~0.0~~ 1.0
value	~~"true"~~ "false"
valueAsBoolean	~~true~~ false

computePercentTrue

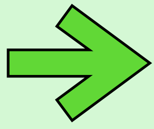
Heap

"true"	"false"

0x500

in/out

Array Example



```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- I hope you're having a great day!

Stack

Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	0x500
trueCount	~~0.0~~ 1.0
value	~~"true"~~ "false"
valueAsBoolean	~~true~~ false

computePercentTrue

Heap

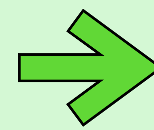
"true"	"false"

0x500

in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```



- End of the loop block
- Cross it out to show that value and valueAsBoolean are no longer in memory

Stack

Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	0x500
trueCount	~~0.0~~ 1.0
~~value~~	~~"true" "false"~~
~~valueAsBoolean~~	~~true false~~

computePercentTrue

Heap

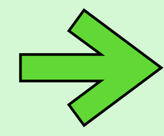
"true"	"false"

0x500

in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```



- Create a new variable and assign it the result of this division

Stack | **Heap**

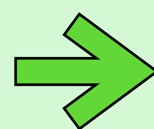
Name	Value
testInput	"true;false"
percentTrue	
line	"true;false"
splits	0x500
trueCount	~~0.0~~ 1.0
~~value~~	~~"true" "false"~~
~~valueAsBoolean~~	~~true false~~
toReturn	0.5

computePercentTrue

Heap: "true" "false" at 0x500

in/out

Array Example

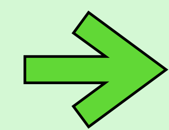


```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- toReturn is the last expression that's evaluated

Array Example

```scala
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```



- Return 0.5
- Cross out the stack frame
  - It's no longer in memory

Stack

Name	Value
testInput	"true;false"
percentTrue	0.5
line	"true;false"
splits	0x500
trueCount	~~0.0~~ 1.0
value	~~"true"~~ ~~"false"~~
valueAsBoolean	true false
toReturn	0.5

computePercentTrue

Heap

"true"	"false"

0x500

in/out

Array Example

```
def computePercentTrue(line: String): Double = {
  val splits: Array[String] = line.split(";")
  var trueCount: Double = 0.0
  for (value <- splits) {
    val valueAsBoolean: Boolean = value.toBoolean
    if (valueAsBoolean) {
      trueCount += 1.0
    }
  }
  val toReturn: Double = trueCount / splits.length
  toReturn
}


def main(args: Array[String]): Unit = {
  val testInput = "true;false"
  val percentTrue = computePercentTrue(testInput)
  println(percentTrue)
}
```

- Print to the screen
- End of program