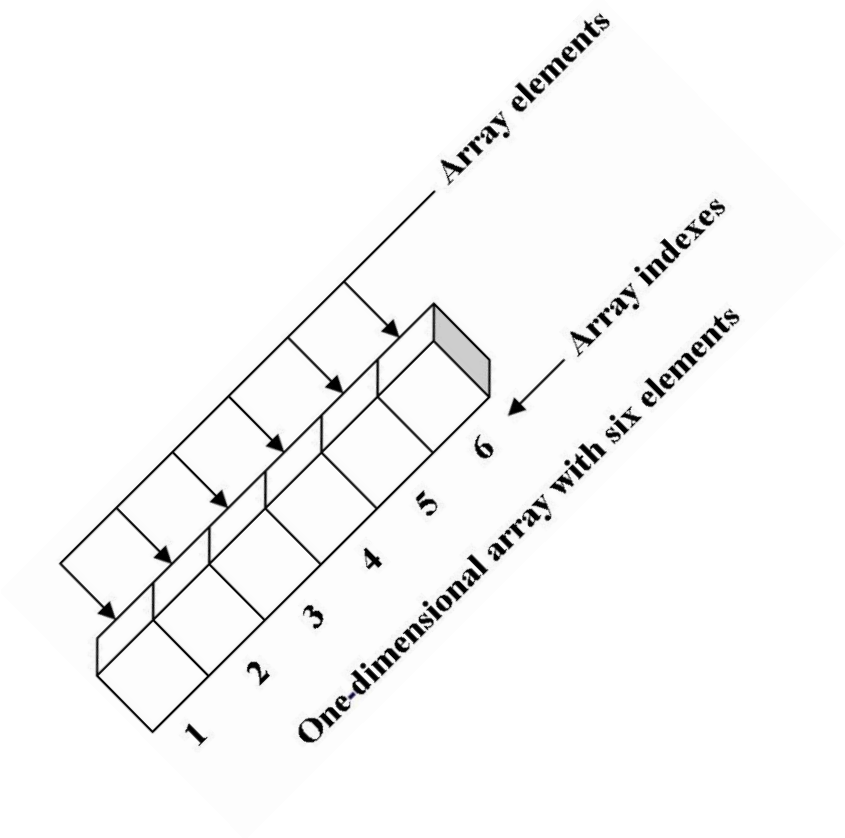


From Python to Java: Lists vs. Arrays

Computer Science 112
Boston University

Christine Papadakis

Recall:

Simple Java Program

```
import java.util.*;

public class Play {

   public static void main( String[] args ) {

      Scanner scan = new Scanner( System.in );

      System.out.print( “Enter three numbers:” );

      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.print(“The numbers entered are: “ );
      System.out.prinln( num + “ “ + num2 + “ “ + num3 );
              .
              .
              .
   }
}
```

Recall:

Simple Java Program

```
import java.util.*;

public class Play {

   public static void main( String[] args ) {

      Scanner scan = new Scanner( System.in );

      System.out.print( “Enter three numbers:” );

      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.print(“Show which number? “ );
      int number = scan.nextInt();
              .
              .
              .
   }
}
```

Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.print("Show which number ?" );
      int number = scan.nextInt();

      System.out.print("Number entered is: ");
      if (number == 1 )
          System.out.println(num1);
      else if (number == 2 )
         System.out.println(num2);
      else
         System.out.println(num3);
   }
}
```

Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int num1 = scan.nextInt();
      int num2 = scan.nextInt();
      int num3 = scan.nextInt();

      System.out.println("Show which
      int number = scan.nextInt();

      System.out.print("Number entered is: ");
      if (number == 1 )
          System.out.println(num1);
      else if (number == 2 )
         System.out.println(num2);
      else
         System.out.println(num3);
   }
}
```

The problem is that each memory cell is mapped to a specific variable name.

Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int numbers = int[3];  // array declaration for
                             // three integer cells

      System.out.println("Show which number ?" );
      int number = sca  nextInt();

      System.out.p  ll(  ered is: ");
      if

   }
}
```

Arrays allow us to store a sequence of ***variables*** in one data structure and …

Our Simple Java Program

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int numbers = int[3];  // array declaration for
                             // three integer cells

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.[illegible]ered is: ");
      if [illegible]

   }
}
```

… access any data item in that sequence through one ***variable*** reference!

Sequences

- A *sequence* is a collection of values in which each value has a *position* or *index* within the sequence.

0	*1*	***2***	*3*	*4*	← indices
51	50	36	29	30	← elements

- The values are known as *elements* of the sequence.

- Another example: a string is a sequence of characters
  - the characters are the elements
  - each character has a position/index

Sequences

- A *sequence* is a collection of values in which each value has a *position* or *index* within the sequence.

0	*1*	*2*	*3*	*4*	← indices
51	50	36	29	30	← elements

- The values are known as *elements* of the sequence.

- Another example: a string is a sequence of characters
  - the characters are the elements
  - each character has a position/index

Sequences

- A *sequence* is a collection of values in which each value has a *position* or *index* within the sequence.

0	*1*	*2*	*3*	*4*	← indices
51	50	36	29	30	← elements

- The values are known as *elements* of the sequence.

- Another example: a string is a sequence of characters
  - the characters are the elements
  - each character has a position/index

- We use the index is used to reference a specific element in the sequence.

Sequences in Python and Java

- In Python, a *list* is a sequence of *arbitrary* values.

```
[2, 4, 6, 8]
['CS', 'math', 'english', 'psych']
```

  - the elements can have arbitrary types:

```
['Star Wars', 1977, 'PG', [35.9, 460.9]]
```

- Java provides a similar construct known as an *array*.
  - the elements must have the same data type
  - less flexible than a Python list, with less built-in functionality

Sequences in Python and Java

- In Python, a *list* is a sequence of *arbitrary* values.

```
[2, 4, 6, 8]
['CS', 'math', 'english', 'psych']
```

  - the elements can have arbitrary types:

```
['Star Wars', 1977, 'PG', [35.9, 460.9]]
```

- Java provides a similar construct known as an *array*.
  - the elements must have the same data type
  - less flexible than a list, with less built-in functionality
  - **but**, it also has less overhead
    - example: a Java array of 1000 integers will use much less memory than a Python list of 1000 integers
  - it is easier to use it efficiently

Arrays

An Array is a *fixed* data structure.

It is a *container* that holds
a fixed number of elements of the same data type.

The length of an array is established when
memory for the array is allocated and
the physical size of the array
cannot be altered during run time.

Arrays

Specifically:

An array *variable* references
a *contiguous* memory allocation of
some fixed number of elements
of the same data type.

Why is this important?

This is why arrays are more efficient and
require less overhead than Python lists!

Contiguous allocation means that all the elements
of the array are stored in consecutive memory cells.

The fact that all the elements are of the *exact data type* means
that each element is allocated the same number of bytes.

Therefore if we know the address location of the first element,
we can create an ***offset*** from that starting address and
directly access every element.

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[3];    // declare an array of
                               // three integers
      nums[0] = scan.nextInt(); // offsets
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      nums[number-1];

   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[3];   // declare an array of
                              // three integers
      nums[0] = scan.nextInt();
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      nums[number-1];

   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[3];   // declare an array of
                              // three integers
      nums[0] = scan.nextInt();
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      nums[0];  // when number == 1

   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[3];   // declare an array of
                              // three integers
      nums[0] = scan.nextInt();
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      nums[1];  // when number == 2

   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String [][] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[3];   // declare an array of
                              // three integers
      nums[0] = scan.nextInt();
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      nums[2];  // when number == 3

   }
}
```

Array Variables

- We use a variable to represent the array as a whole.
- Example of declaring an array *variable*:

```
int[] temps = {1, 2, 3, 4, 5};
```

  - the `[]` indicates that variable temps represents an array
  - the `int` indicates that the elements will be of type `int`
  - creates an array in memory, *initialized with the specified elements*, and assigns the memory location (of the first element) to variable temps.

Array Variables

- We use a variable to represent the array as a whole.
- Example of declaring an array *variable*:

```
int[] temps = new int[5];
```

  - the `[]` indicates that variable temps represents an array
  - the `int` indicates that the elements will be of type `int`
  - creates an array of five elements (initialized to default value) in memory and assigns the memory location (of the first element) to variable temps.

Array Variables

- General pattern:

```
type[] variable = new type[length];
```

```
double[] vals = new double[100]; // array for 100 doubles
String[] names = new String[10]; // array for 10 Strings
```

- Initially, the arrays are filled with the default value of their type:

```
int     0          boolean  false
double  0.0        objects  the special value null
```

- Once an array is created you can use the *length* attribute of the array to know the size the array was declared: Example:
  - `vals.length`
  - `names.length`

Array Variables

- General pattern:

```
type[] variable = new type[length];
```

```
double[] vals = new double[100]; // array for 100 doubles
String[] names = new String[10]; // array for 10 Strings
```

- Initially, the arrays are filled with the default value of their type:

```
int     0          boolean  false
double  0.0        objects  the special value null
```

- Once an array is created you can use the *length* attribute of the array to know the size the array was declared: Example:

  - `System.out.println(vals.length);  // 100`
  - `System.out.println(names.length);  // 10`

Array Variables

- General pattern:

```
type[length];
```

```
[100]; // array for 100 doubles
g[10]; // array for 10 Strings
```

Note that we are accessing an *attribute* of the `Array` class and not calling a method:

`vals.length` and **not**…

```
String str = "Hello";
str.length()
```

- the default value of their type:

```
ean  false
cts  the special value null
```

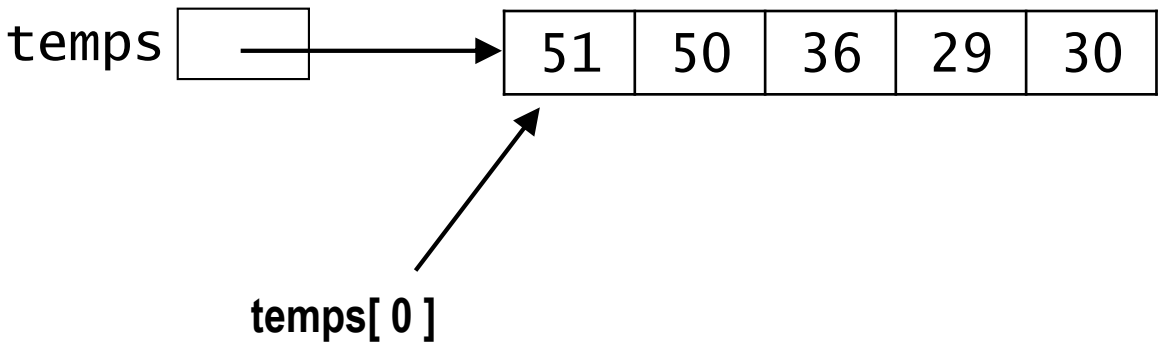
- Once an array is created you can use the *length* attribute of the array to know the size the array was declared: Example:

```
System.out.println(vals.length);     // 100
System.out.println(names.length);    // 10
```

Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

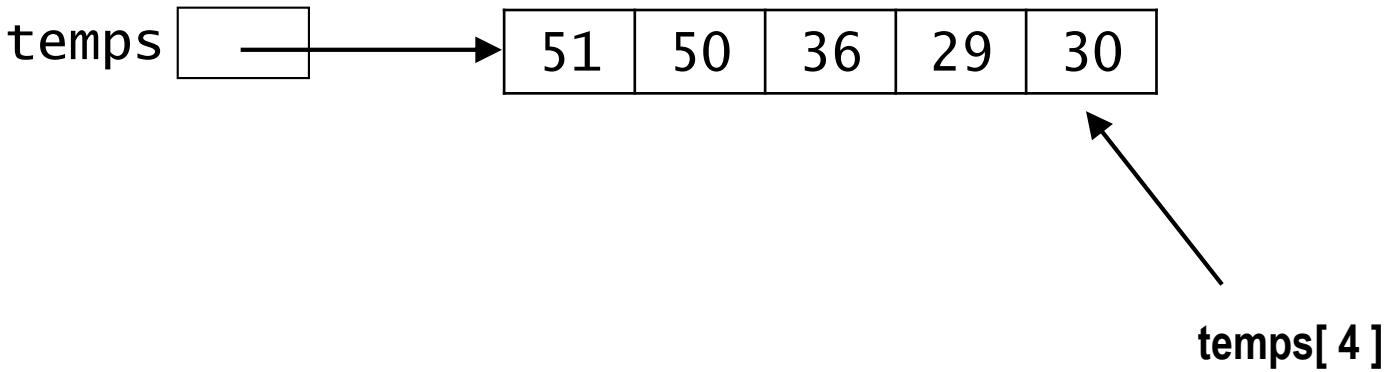
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

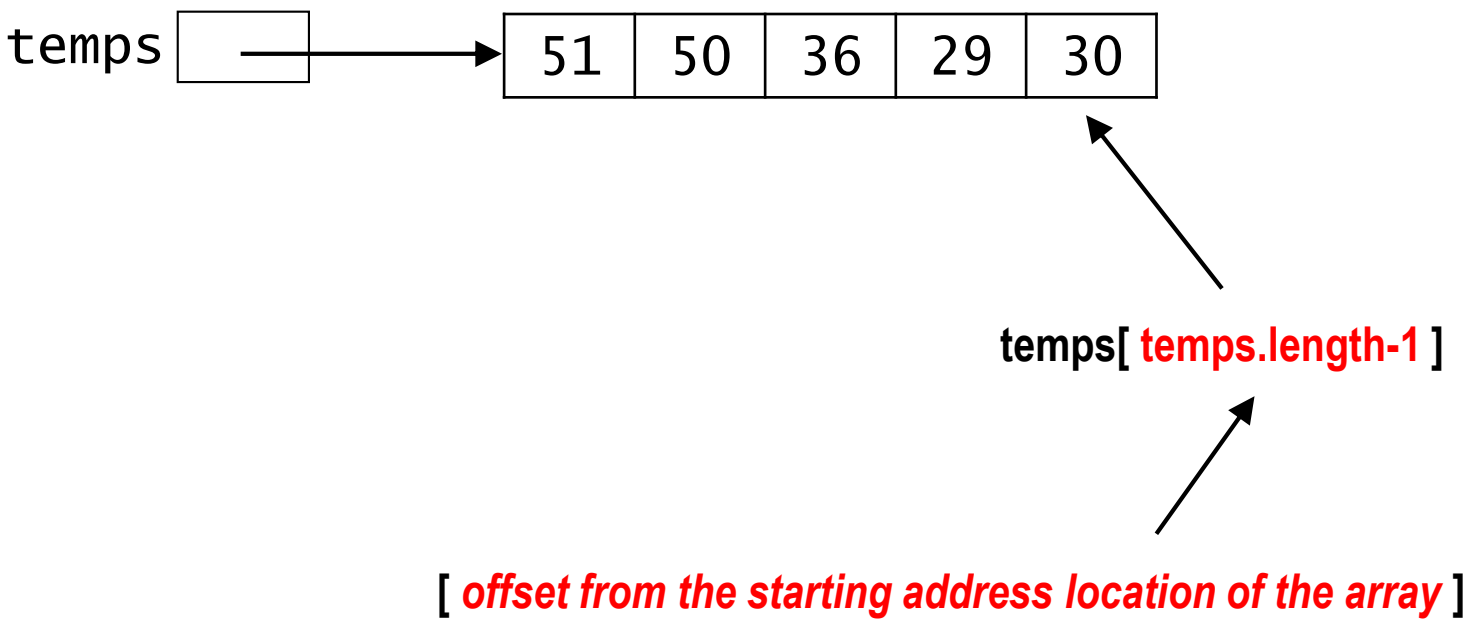
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

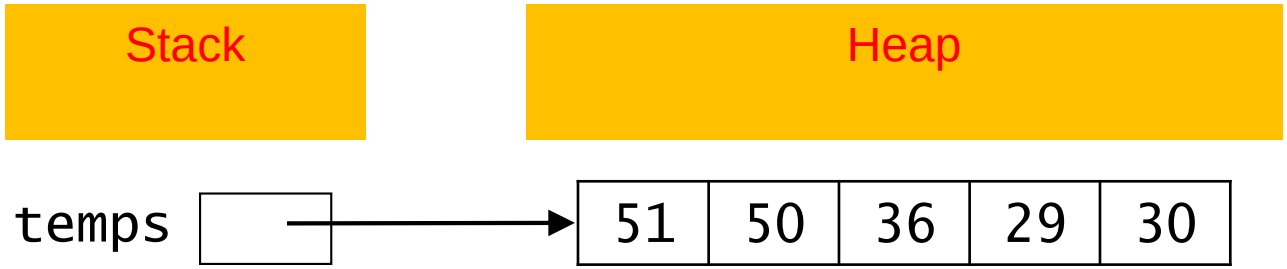
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

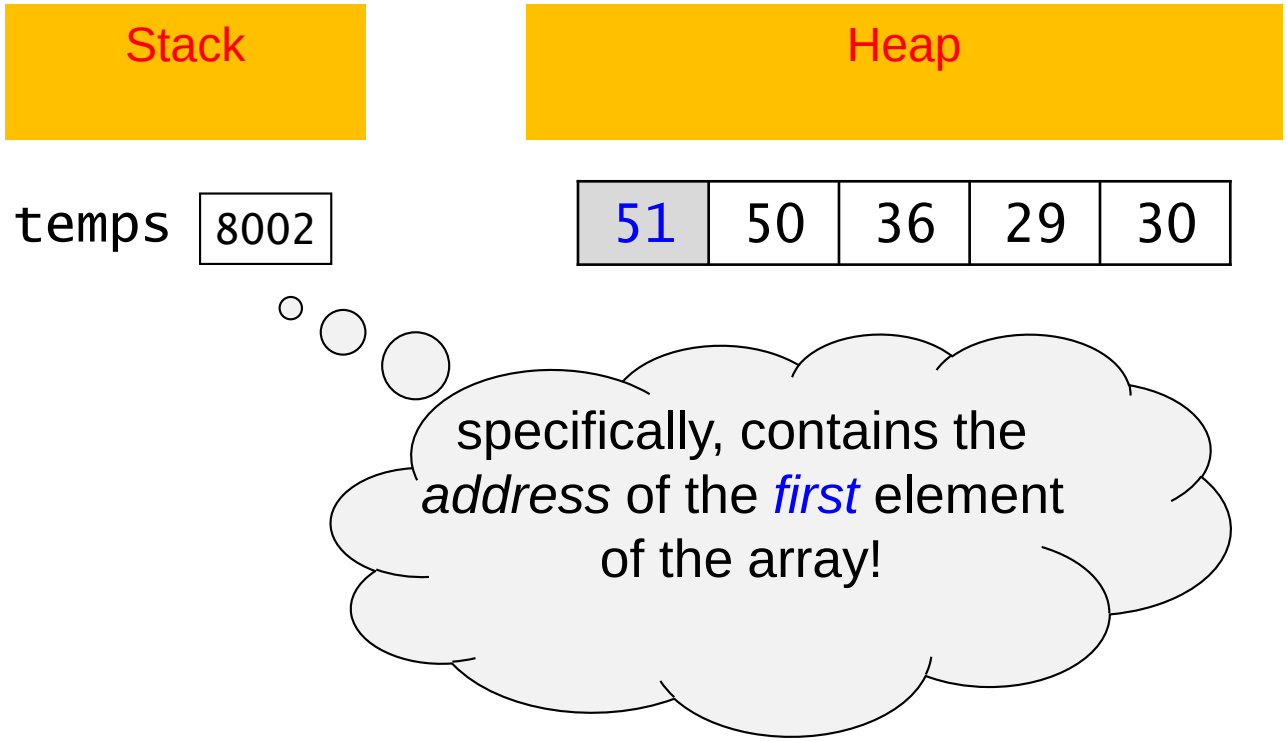
```
int[] temps = {51, 50, 36, 29, 30};
```



Arrays and References

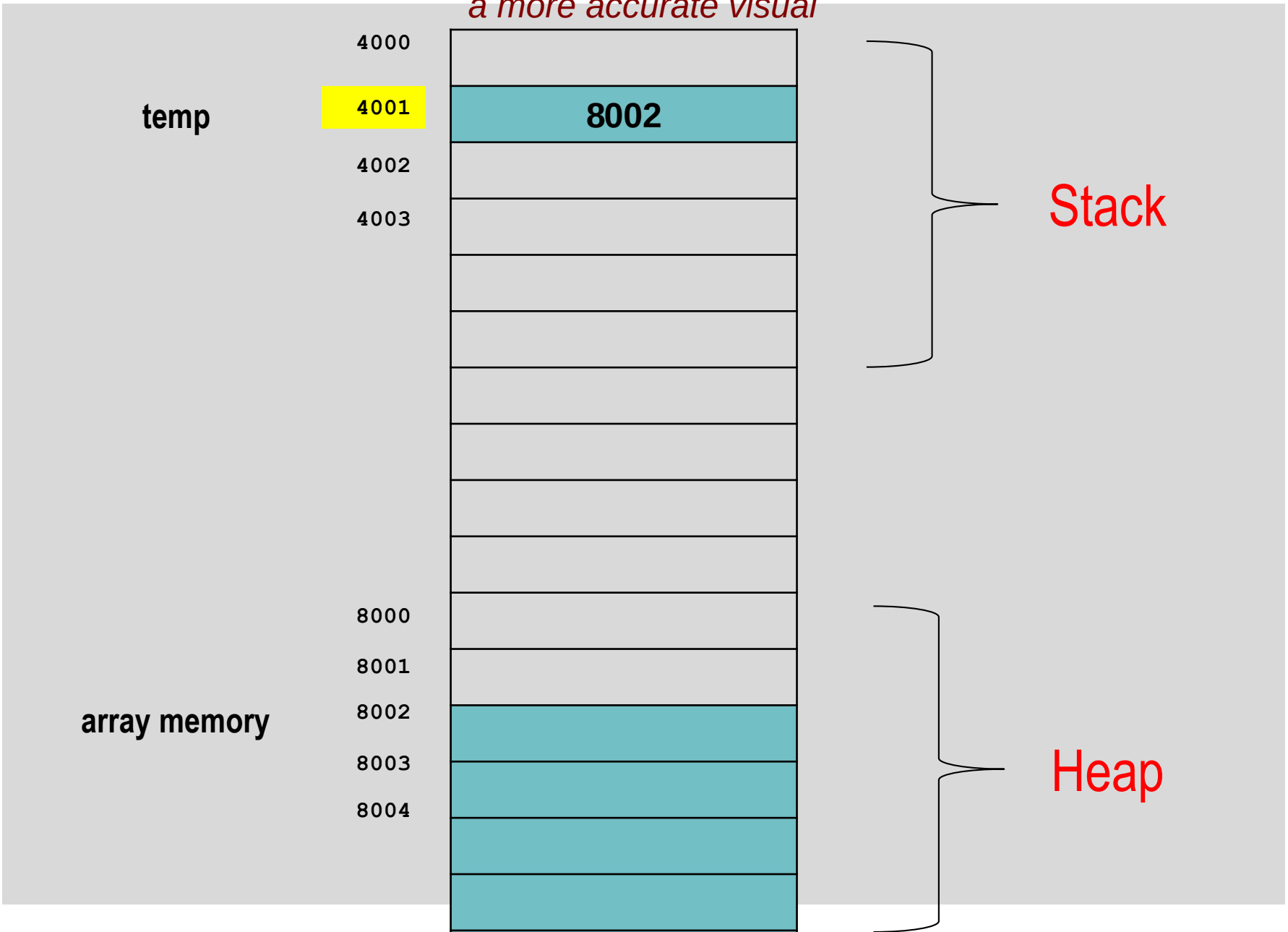
- An array variable does *not* store the array itself.
- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```



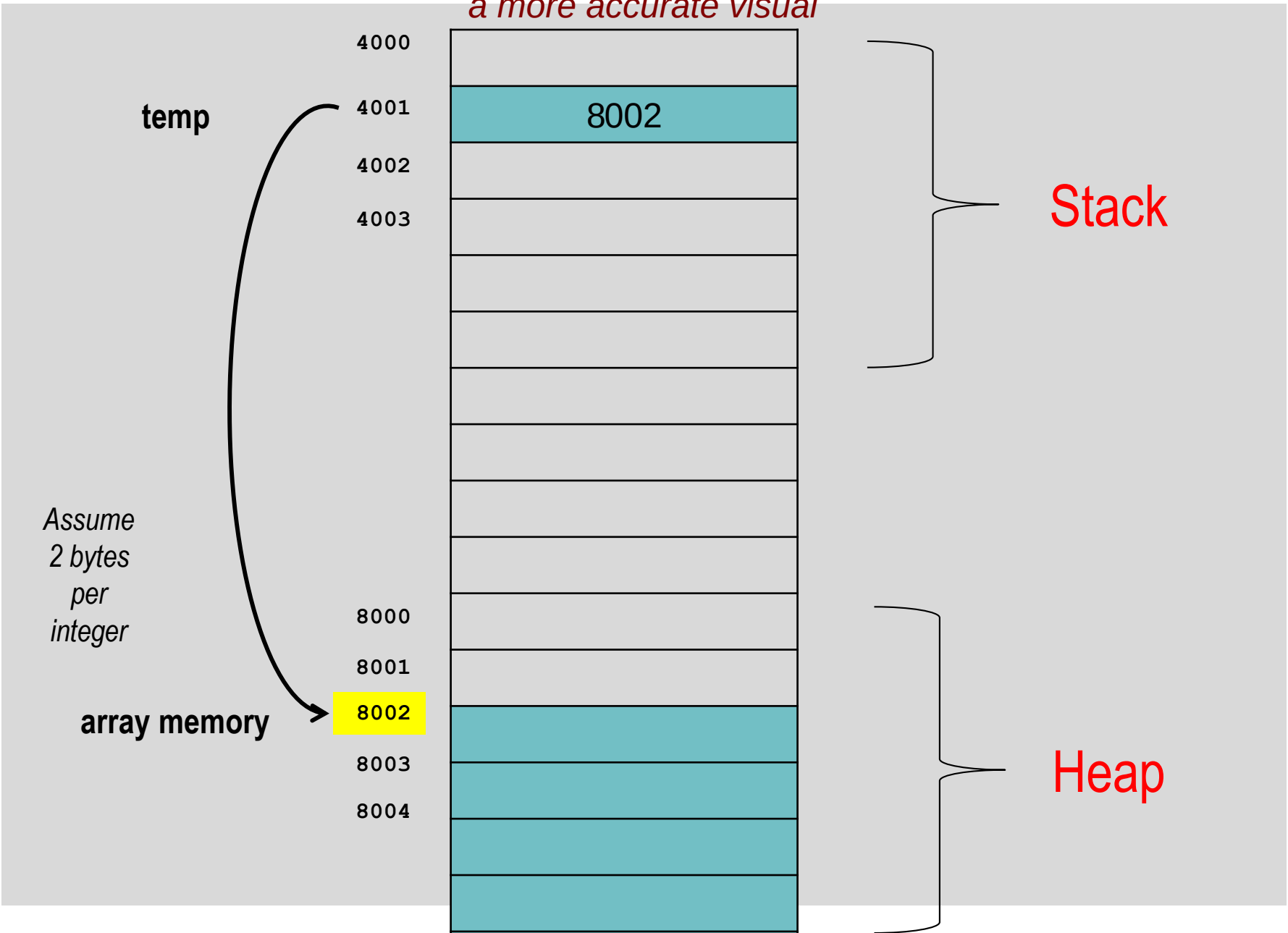
Arrays and References:

a more accurate visual



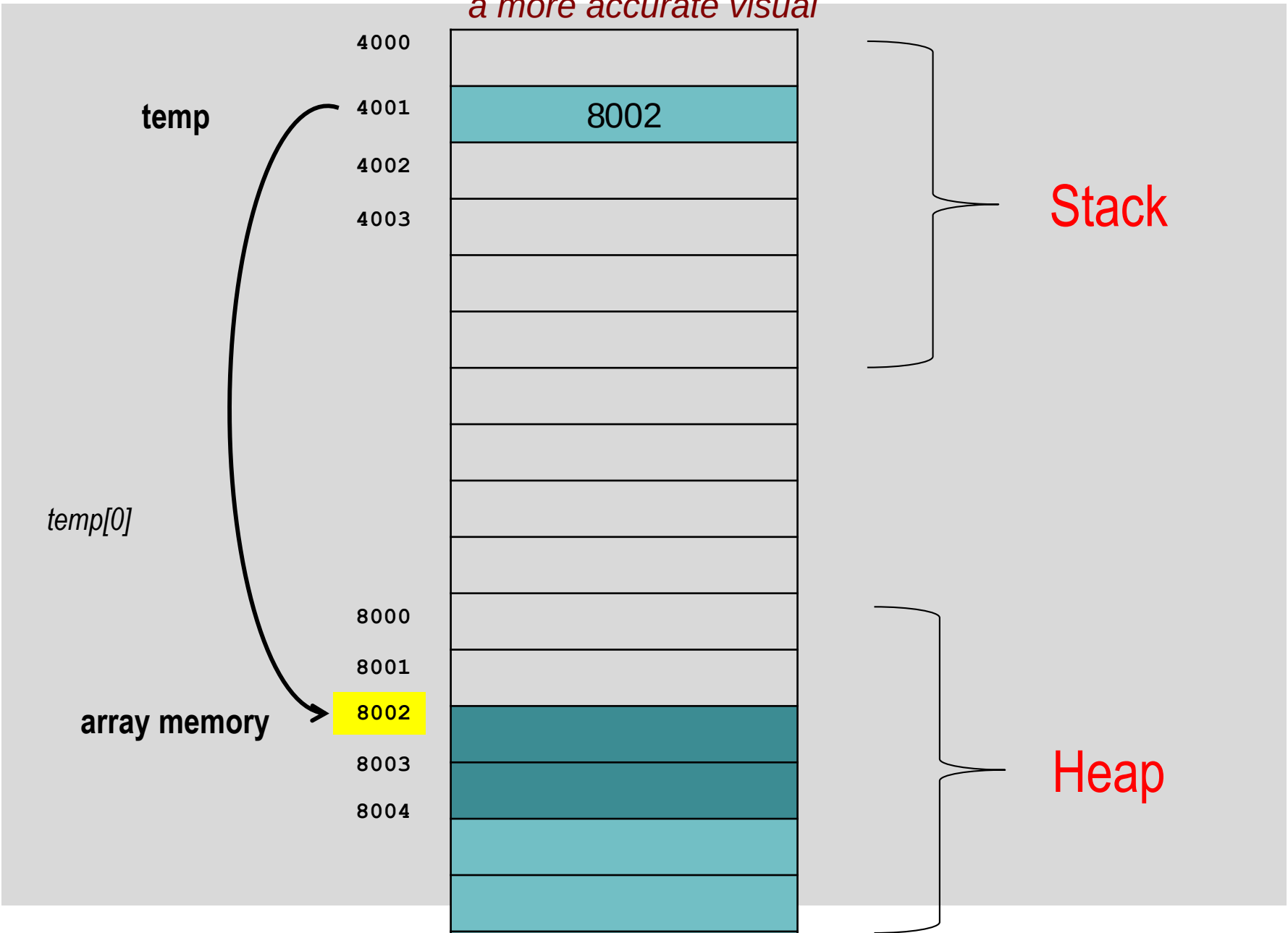
Arrays and References:

a more accurate visual



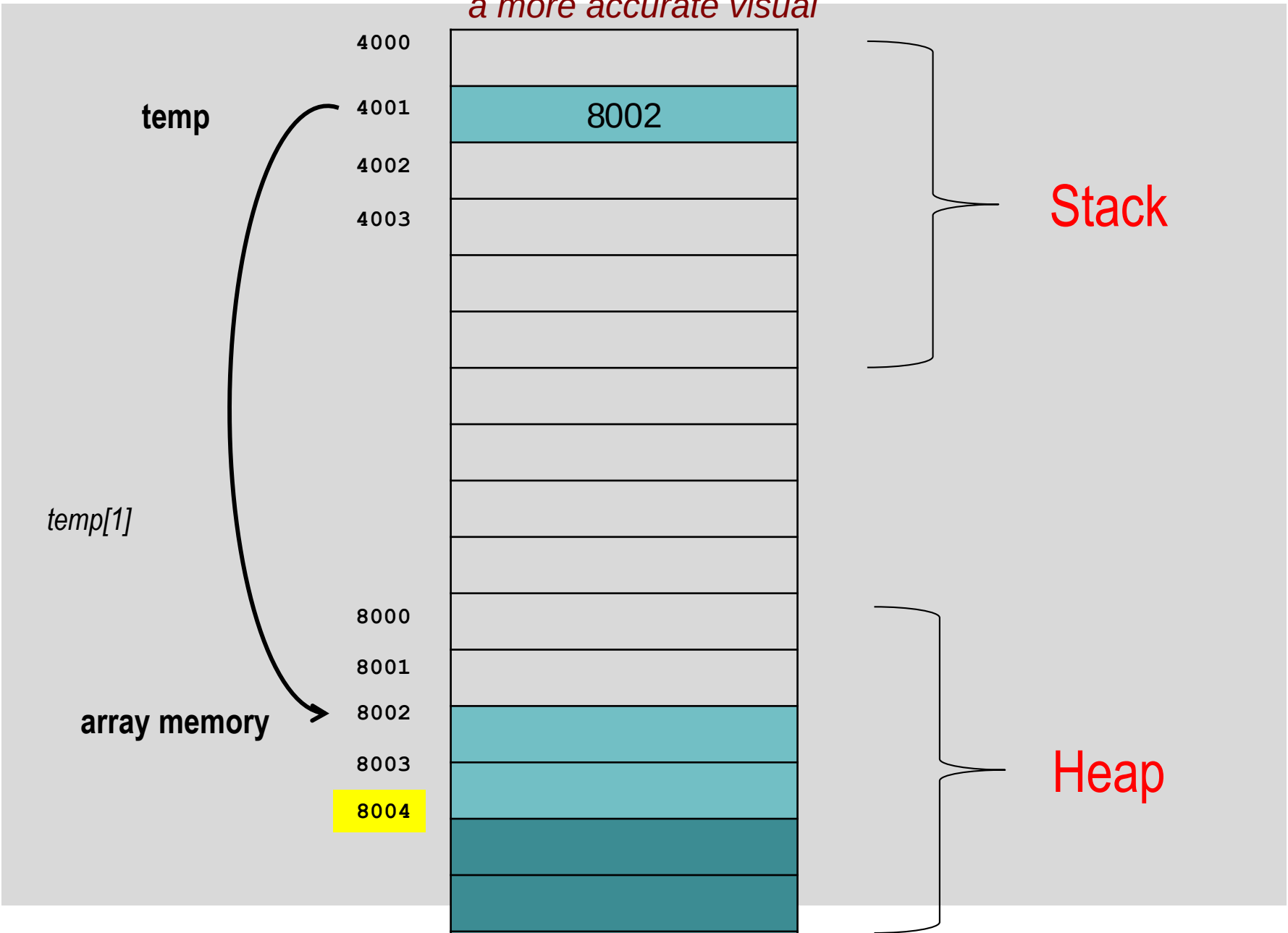
Arrays and References:

a more accurate visual



Arrays and References:

a more accurate visual

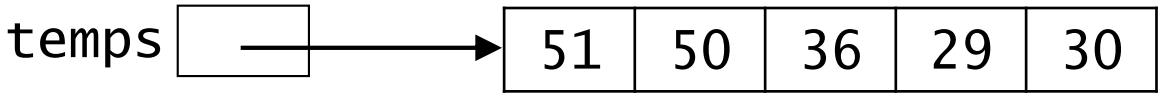


Arrays and References

- An array variable does *not* store the array itself.

- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```



- If we print an array variable, we print the contents or value of the variable, which is the memory address of the array!

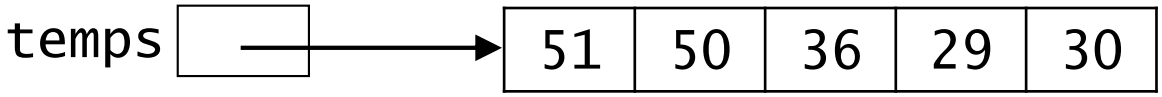
```
System.out.println(temps);
```

Arrays and References

- An array variable does *not* store the array itself.

- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```



- If we print an array variable, we print the contents or value of the variable, which is the memory address of the array!

```
System.out.println(temps);
```

output:

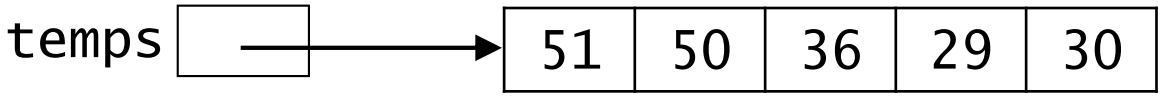
```
[I@1e1fd124
```

Arrays and References

- An array variable does *not* store the array itself.

- It stores a *reference* to the array.
  - the memory address of the array

```
int[] temps = {51, 50, 36, 29, 30};
```

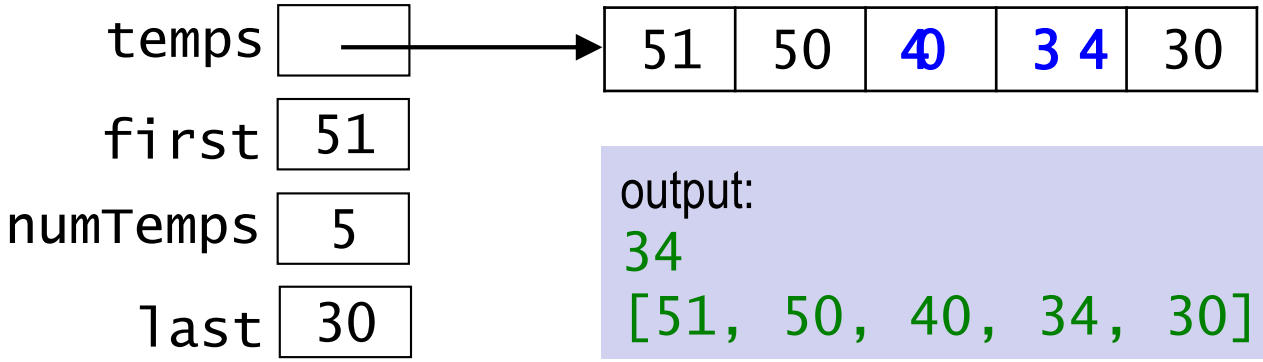


- To print the contents of the array, we must first invoke a ***static*** method of the Array class that returns a string representation of the specified array:

```
System.out.println( Arrays.toString(temps) );
```

What is the output of the full program?

```
import java.util.*;

public class FunWithArrays {
    public static void main(String[] args) {
        int[] temps = {51, 50, 36, 29, 30};
        int first = temps[0];
        int numTemps = temps.length;
        int last = temps[numTemps - 1];

        temps[2] = 40;
        temps[3] += 5;
        System.out.println(temps[3]);
        System.out.println(Arrays.toString(temps));
    }
}
```



output:

```
34
[51, 50, 40, 34, 30]
```

Printing an Array

```
import java.util.*;

public class FunWithArrays {
    public static void main(String[] args) {
        int[] temps = {51, 50, 36, 29, 30};
        int first = temps[0];
        int numTemps = temps.length;
        int last = temps[numTemps - 1];

        temps[2] = 40;
        temps[3] += 5;
        System.out.println(temps[3]);
        System.out.println(Arrays.toString(temps));
    }
}
```

allows the compiler to find the `Arrays` class, which is in the `java.util` package

pass the array into a method called `toString()`, which returns a string representation of the array

toString() is a static method from a class called `Arrays`

What does this print?

```
import java.util.*;

public class FunWithArrays {
    public static void main(String[] args) {
        int[] vals = {2, 4, 5, 7, 3};
        vals[1] = 6;
        vals[2] *= vals[1];
        System.out.println(Arrays.toString(vals));
    }
}
```

vals[2] = vals[2]*vals[1]

A. `[2, 4, 5, 7, 3]`

B. `[6, 24, 5, 7, 3]`

C. `[6, 8, 5, 7, 3]`

D. `[2, 6, 20, 7, 3]`

E. `[2, 6, 30, 7, 3]`

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[100]; // declare an array of
                              // one hundred integers
      nums[0] = scan.nextInt();
      nums[1] = scan.nextInt();
      nums[2] = scan.nextInt();
       .
       .
       .
       .

      System.out.println("S
      int number = scan.next

      System.out.printl("Number entered      );
      nums[number-1];
```

Should we add 100 hundred input statements?

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[100]; // declare an array of
                              // one hundred integers

      for (int i = 0; i < 100; i++ )
          nums[i] = scan.nextInt();

      System.out.println("Show which number ?" );
      int number = scan.nextInt();

      System.out.printl("Number entered is: ");
      nums[number-1];

   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[100]; // declare an array of
                              // one hundred integers

      for (int i = 0; i < nums.length; i++ )
          nums[i] = scan.nextInt();

                                  "Show which number ?" );
                                  extInt();

                                  Number entered is: ");
```

This is important, because accessing an array *out of bounds* will result in a program exception.

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      Scanner scan = new Scanner( System.in );
      int [] nums = int[100]; // declare an array of
                              // one hundred integers

      for (int i = 0; i < nums.length; i++ )
          nums[i] = scan.nextInt();

      // Call a method to compute and return the sum
      System.out.println( sumOf(nums) );
   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      .
      .

      System.out.println( sumOf(nums) );
   }

   public static int sumOf( int[] arr ) {
      int sum = 0;

      for (int i = 0; i < arr.length; i++ )
         sum += arr[i];

      return( sum );
   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      .
      .

      System.out.println( sumOf(nums) );
   }

   public static int sumOf( int[] arr ) {
      int sum = 0;

      for (int i = 0; i < arr.length; i++ )
         sum += arr[i];

      System.out.println( sum ); // outputs the value
   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      .
      .

      System.out.println( sumOf(nums) );
   }

   public static int sumOf( int[] arr ) {
      int sum = 0;

      for (int i = 0; i < arr.length; i++ )
         sum += arr[i];

      return( sum );              // returns the value
   }
}
```

Our Simple Java Program
with an array

```
import java.util.*;

public class Play {
   public static void main( String[] args ) {
      .
      .

      System.out.println( sumOf(nums) );
   }

   public static int sumOf( int[] arr ) {
      int sum = 0;

      for (int i = 0; i < arr.length; i++ )
         sum += arr[i];

      return( sum );              // returns the value
   }
}
```

Processing a Sequence Using a Loop

Index-based:

Python

```
for i in range(len(list)):
    do something with list[i]
```

where *list* is the list variable

Java

```
for (int i = 0; i < array.length; i++) {
    do something with array[i]
}
```

where *array* is the array variable

Element-based:

Python

```
for val in list:
    do something with val
```

where *list* is the list variable

Java

```
for (int val: array) {
    do something with val
}
```

where *array* is the array variable

- Index-based is more flexible:
  - you can use it to *change* the element with index `i`
  - you can keep track of where you saw a given value

Processing a Sequence Using a Loop

Index-based:

Python

```
for i in range(len(list)):
    do something with list[i]
```

where *list* is the list variable

Java

```
for (int i = 0; i < array.length; i++) {
    do something with array[i]
}
```

where *array* is the array variable

Element-based:

Python

```
for val in list:
    do something with val
```

where *list* is the list variable

Java

```
int sum = 0;

for (int val:temps) {
    sum += val;
}
```

- Index-based is more flexibl
  - you can use it to *change*
  - you can keep track of where you saw a given value

Processing a Sequence Using a Loop

Index-based:

Python

```
for i in range(len(list)):
    do something with list[i]
```

where *list* is the list variable

```
for (int i =
    do son
}
```

where *array*

. The loop iterates over the elements of the array, so it depends on the array data type.

Element-based:

Python

```
for val in list:
    do something with val
```

where *list* is the list variable

Java

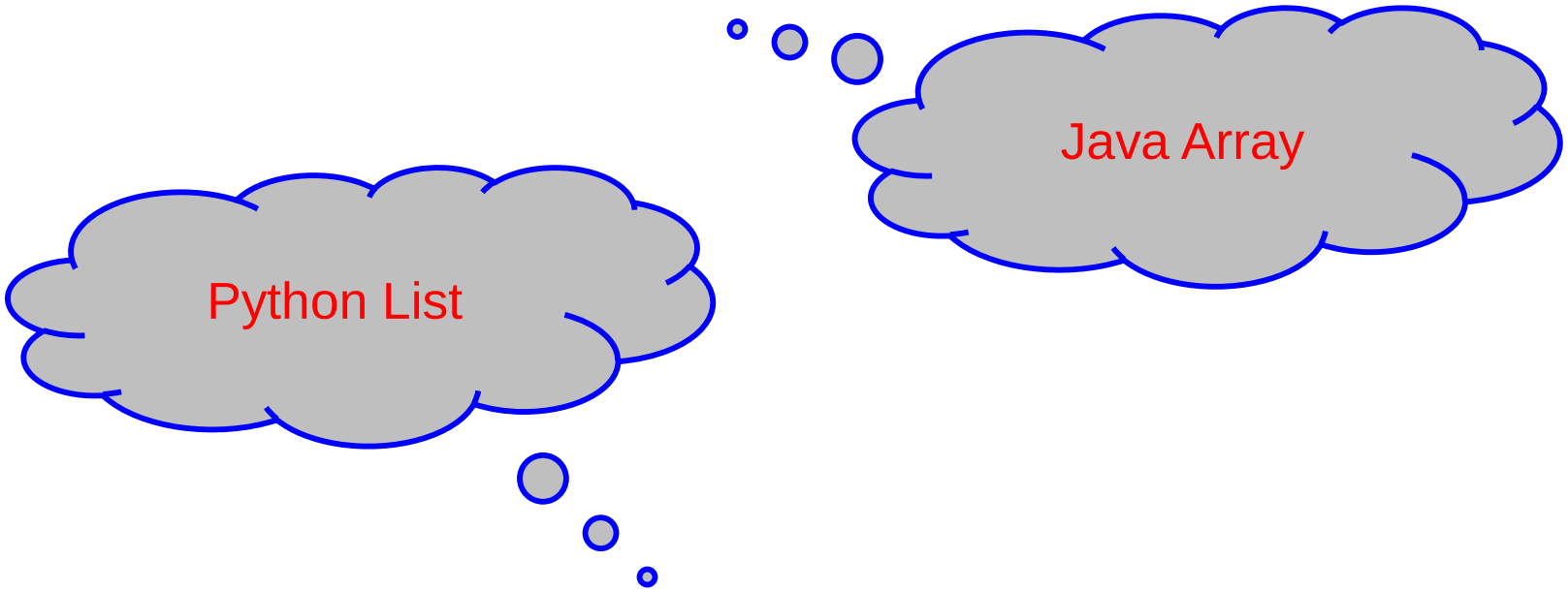
```
int sum  );

for (int val:temps) {
    sum += val;
}
```

- Index-based is more flexibl
  - you can use it to *change*
  - you can keep track of where you saw a given value

Array vs. Python Lists

a summary

```
int arr = {51, 50, 36, 29, 30};
```




```
temps = [51, 50, 36, 29, 30]
```

Basic Operations on Lists vs. Arrays

Python

```
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

Basic Operations on Lists vs. Arrays (cont.)

Python

```python
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```java
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

Basic Operations on Lists vs. Arrays (cont.)

Python

```
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

- Python uses [ ] to both:
  - surround list literals
  - index into the list

- Java uses:
  - { } to surround array literals
  - [ ] to index into the array

Basic Operations on Lists vs. Arrays (cont.)

Python

```python
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```java
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

- Python uses `[ ]` to both:
  - surround list literals
  - index into the list
    - from both ends *(of the list)*

- Java uses:
  - `{ }` to surround array literals
  - `[ ]` to index into the array
    - cannot use negative indices

Basic Operations on Lists vs. Arrays (cont.)

Python

```
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);  // no!
```

- `len(values)` gives the length of the list `values`
- printing a list displays its contents.

- `temps.length` gives the length of the array `values`
  - `length` is *not* a method, it is an attribute of the Arrays class
  - recall finding the length of a string: `s.length()`
- printing an array **does *not*** display its contents

Other Differences

Python

```
temps = [51, 50, 36, 29, 30]
first_two = temps[0:2]
temps = temps + [45, 29]
new_temps = [65] * 5
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
// no operator for slicing!
// no operator for concatenating!
// no operator for multiplying!
```

- In Java, the only array operator is `[]` for indexing.

Other Differences

Python

```
temps = [51, 50, 36, 29, 30]
first_two = temps[0:2]
temps = temps + [45, 29]
new_temps = [65] * 5
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
// no operator for slicing!
// no operator for concatenating!
// no operator for multiplying!
```

- In Java, the only array operator is `[]` for indexing.
- The `Array` class has static methods that provide the functionality of some of Python's operators.
  - example: `Arrays.copyOfRange(values, start, end)` returns the *slice* `values[start : end]`

Note the use of the *dot* operator to connect the method with the class!

Other Differences

Python

```
temps = [51, 50, 36, 29, 30]
first_two = temps[0:2]
temps = temps + [45, 29]
new_temps = [65] * 5
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
// no operator for slicing!
// no operator for concatenating!
// no operator for multiplying!
```

- In Java, the only array operator is `[]` for indexing.
- The `Array` class has static methods that provide the functionality of some of Python's operators.
  - example: `Arrays.copyOfRange(values, start, end)` returns the slice `values[start : end]`
- If you really need the extra functionality, it's more common to use one of Java's built-in *collection classes*.
  - they allow you to construct a list *object* for a sequence
  - we'll soon be building our own collection classes!

Constructing an Array

Python

```
temps = [0] * 4
```

Java

```
int[] temps = new int[4];
```

- General pattern:

type`[]` *variable* `= new` *type*`[`*length*`];`

```
double[] vals = new double[100];    // array for 100 doubles
String[] names = new String[10];    // array for 10 string
                                    // references!
```

Constructing an Array

Python

```
temps = [0] * 4
```

Java

```
int[] temps = new int[4];
```

- General pattern:

type`[]` *variable* `= new` *type*`[`*length*`];`

```
double[] vals = new double[100];   // array for 100 doubles
String[] names = new String[10];   // array for 10 Strings
```

- Initially, the arrays are filled with the default value of their type:

`int`	`0`	`boolean`	`false`
`double`	`0.0`	objects	the special value `null`

Constructing and Filling a List / Array

Python

```
temps = [0] * 4
```

Java

```
int[] temps = new int[4];
```

Now that we have created
these structures,
how can we fill them
with data?

Constructing and Filling a List / Array

Python

```
temps = [0] * 4
print('enter 4 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[4];
System.out.println('enter 4 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

…. print out the contents of the array...

Constructing and Filling a List / Array

Python

```
temps = [0] * 100
print('enter 100 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

Anyone want to explicitly write 100 input statements?

Constructing and Filling a List / Array

Python

```
temps = [0] * 100
print('enter 100 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

more significantly, what if we do not know until run time the number of elements we need?

Constructing and Filling a List / Array

Python

```
temps = [0] * 100
print('enter 100 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

A more practical alternative is to use a loop!

Constructing and Filling a List / Array:

arrays and loops

Python

```python
temps = [0] * 100
print('enter 100 temps:')
for i in range(100):
    temps[i] = int(input())
print(temps)
```

Java

```java
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
for (int i = 0; i < 100; i++) {
    temps[i] = scan.nextInt();
}
System.out.println(
  Arrays.toString(temps));
```

To make the code more flexible…

Python

```python
temps = [0] * 100
print('enter 100 temps:')
for i in range(len(temps)):
    temps[i] = int(input())
print(temps)
```

Java

```java
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
for (int i = 0; i < temps.length; i++) {
    temps[i] = scan.nextInt();
}
System.out.println(
  Arrays.toString(temps));
```

Code to sum all the values of the array…

Python

```python
temps = [0] * 100
print('enter 100 temps:')
for i in range(len(temps)):
    temps[i] = int(input())
print(temps)
```

Java

```java
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
for (int i = 0; i < temps.length; i++) {
    temps[i] = scan.nextInt();
}
System.out.println(
  Arrays.toString(temps));
```

```java
int sum = 0;

for (int i = 0; i < temps.length; i++) {
    sum += temps[i];
}
```

This is important, because accessing an array *out of bounds* will result in a program exception.