Arrays of Numbers

Programs often need to store multiple values of the same type, such as a list of phone numbers, or the names of your top 20 favorite songs. Rather than create a separate variable for each one, we can store them together using an array.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain course policies about academic honesty.
- Declare and initialize array variables of primitive types.
- Draw a memory diagram of an array of reference types.
- Write a for loop that iterates the contents of an array.

Process Skill Goals

During the activity, students should make progress toward:

- Developing algorithms that loop through arrays to compute a result. (Problem Solving)

Model 1 Case Study: Panic Attack

Frank was behind in his programming assignment. He approached Martin to see if he could get some help. But he was so far behind and so confused that Martin just gave him his code with the intent that he would "just look at it to get some ideas."

In the paraphrased words of Frank: "I started the assignment three days after you put it up. But then other assignments came in and I started on them too. I felt like I was chasing rabbits and began to panic. It was already past the due date and I got really scared. That's when I went to Martin to see if he could help." Frank copied much of the code and turned it in as his own.

Questions (7.5 min) **Start time:** _____

1. Which, if any, of the students were at fault? Why?

2. Which specific Honor Code violations occurred?

3. What should Martin have done in this situation?

4. What options did Frank have besides cheating?

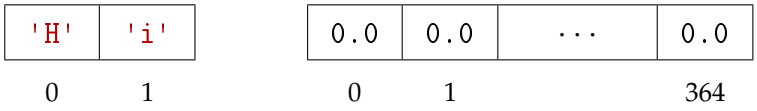
Model 2 Array Syntax

An ***array*** allows you to declare a collection of related variables (of the same type) at once. Each value in an array is known as an ***element***. The programmer must specify the ***length*** of the array (the number of array elements). Once the array is created, its length cannot be changed.

```
char[] letterArray = {'H', 'i'};
System.out.println(letterArray[0]);          // outputs H
System.out.println(letterArray.length);      // outputs 2

double[] numberArray = new double[365];
System.out.println(numberArray[0]);          // outputs 0.0
System.out.println(numberArray.length);      // outputs 365
```

Array elements are accessed by ***index*** number, starting at zero:



Questions (15 min) Start time: _____

5. Examine the results of the above code.

   a) What is the index for the element `'i'`?

   b) What is the length of `letterArray`?

   c) What is the length of `numberArray`?

   d) How would you access the last element of `numberArray`?

6. Now examine the syntax of the code.

   a) What are three ways that square brackets [] are used?

   b) In contrast, how are curly braces {} used for an array?

7. What are the type and value for each of the four *expressions* below?

```
int[] a = {3, 6, 15, 22, 100, 0};
double[] b = {3.5, 4.5, 2.0, 2.0, 2.0};
String[] c = {"alpha", "beta", "gamma"};
```

a) `a[3] + a[2]`

b) `b[2] - b[0] + a[4]`

c) `c[1].charAt(a[0])`

d) `a[4] * b[1] <= a[5] * a[0]`

Array variables can be initialized without the `new` keyword:

```
int[] picks = {3, 5, 7, 2, 1};
String[] names = {"Grace", "Alan", "Tim"};
```

However, if the variable is already declared, `new` is required:

```
picks = new int[] {3, 5, 7, 2, 1};
names = new String[] {"Grace", "Alan", "Tim"};
```

8. Write *expressions* that create the following `new` arrays. (Do not declare variables.)

a)

0	14	1024	127	3	5521

b)

3.23	1.52	4.23	32.5	2.45	5.23	3.33

9. Write *statements* that both declare and initialize variables for these `new` arrays.

a)

0	14	1024	127	3	5521

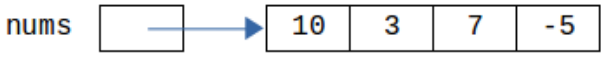
b)

3.23	1.52	4.23	32.5	2.45	5.23	3.33

Model 3 Array Diagrams

Array elements are stored together in one contiguous block of memory. To show arrays in memory diagrams, we simply draw adjacent boxes.

```
int[] nums = {10, 3, 7, -5};
```




```
double[] stats = new double[3];
```

Questions (15 min)

Start time: _____

10. Draw a memory diagram for the following array declarations.

a)
```
int[] sizes = new int[5];
sizes[2] = 7;
```

b)
```
double[] costs = new double[4];
costs[0] = 0.99;
```

c)
```
String[] names = new String[3];
names[1] = "Anita";
```

11. What is the ***default*** value for uninitialized array elements? (Hint: You should have no empty boxes in your memory diagrams above.)

12. Like strings, arrays are reference types. What is the *value* of an array variable?

13. Draw a memory diagram of the following array. (Hint: You should have four arrows.)

```
String[] greek = {"alpha", "beta", "gamma"};
```

Model 4 Arrays and Loops (optional)

The real power of arrays is the ability to process them using loops, i.e., performing the same task for multiple elements.

```
for (int i = 0; i < array.length; i++) {
   // ... process array[i] ...
}
```

Here are two specific examples:

```
// set all of the elements of x to -1.0
double[] x = new double[100];
for (int i = 0; i < x.length; i++) {
    x[i] = -1.0;
}
// sum the elements of scores
int sum = 0;
for (int i = 0; i < scores.length; i++) {
    sum += scores[i];
}
```

Questions (15 min) Start time: _____

14. What is the value of `array` and `accumulator` at the end of the following code? Trace the loop by hand in the space below.

```
int[] array = {5, 26, 13, 12, 37, 15, 16, 4, 1, 3};
int accumulator = 0;
for (int i = 0; i < array.length; i++) {
    if (array[i] % 2 == 1 && i + 1 < array.length) {
        array[i] *= -1;
        accumulator += array[i+1];
    }
}
```

15. Implement the following method that creates and returns a new array.

```
/**
 * Return a new array containing the pairwise maximum value of
 * the arguments. For each subscript i, the return value at [i]
 * will be the larger of x[i] and y[i].
 *
 * @param x an array of double values
 * @param y an array of double values
 * @return pairwise max of x and y
 */
public static double[] pairwiseMax(double[] x, double[] y) {









}
```

16. Implement the following method that reads through two integer arrays.

```
/**
 * Computes the final average grade for a student. The labs are
 * worth 40% and the exams are worth 60%. All scores range from
 * 0 to 100, inclusive.
 *
 * @param labs the student's lab scores
 * @param exams the student's exam scores
 * @return weighted average of all scores
 */
public static double finalGrade(int[] labs, int[] exams) {









}
```