

When you pass an array to a function, a shallow copy of the actual parameter is made into the formal parameter. For example, in the following code, **data** becomes a shallow copy of **arr**.

```
1 void caller() {  
2     int[] arr = {10, 70, 50};  
3     int sum = total(arr);  
4 }  
5  
6 int total(int[] data) {  
7     int result = 0;  
8     for(int i=0; i < data.length; i++) {  
9         result+=data[i];  
10    }  
11    return result;  
12 }
```

**scope: caller()**

arr

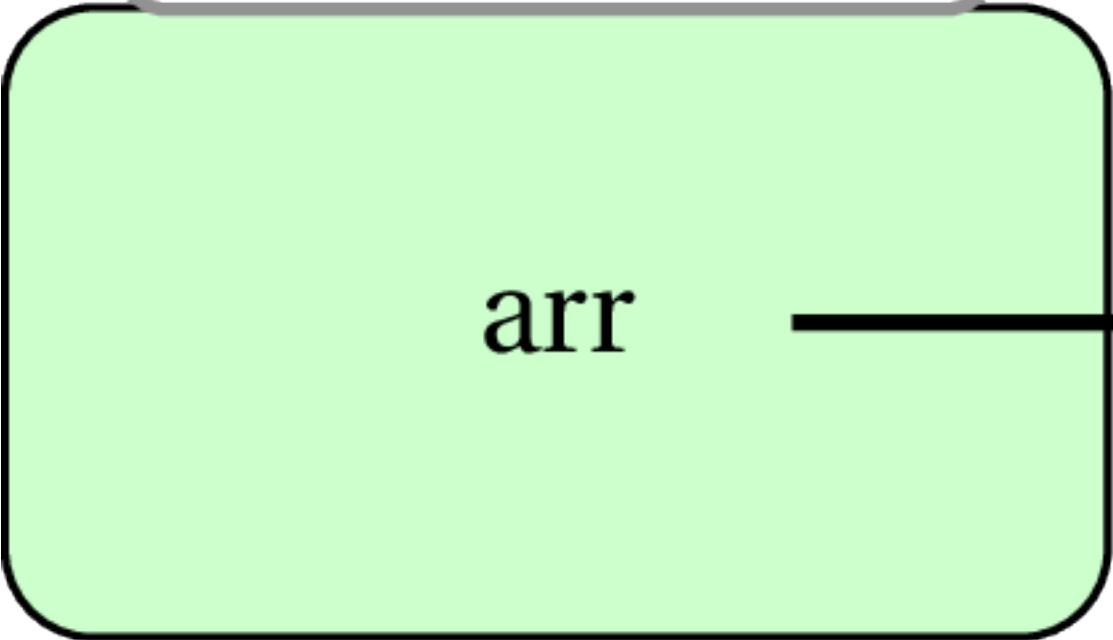
sum

If you have a function that modifies the passed array, the contents of the actual parameter will also change.

```
1 void caller() {  
2     int[] arr = {10, -70, 0};  
3     negate(arr);  
4 }  
5  
6 void negate(int[] data) {  
7     for(int i=0; i < data.length; i++) {  
8         data[i]=data[i]*-1;  
9     }  
10 }
```

**scope: caller()**

arr



The diagram shows a light green rounded rectangle representing a function's local environment. Inside this rectangle, the text 'arr' is centered. A horizontal black line extends from the right side of the rectangle, passing through the text 'arr'.

If a function returns an array (source) and the caller copies it into an array (destination), it's a shallow copy as demonstrated in the following example.

If you have a function that modifies the passed array, the contents of the actual parameter will also change.

```
1 void caller() {  
2     int[] data = getDiceOutcomes(5);  
3 }  
4  
5 int[] getDiceOutcomes(int n) {  
6     int[] outcomes = new int[n];  
7     for(int i=0; i < outcomes.length; i++) {  
8         outcomes[i] = (int)random(1, 7);  
9     }  
10    return outcomes;  
11 }
```

**scope: caller()**

data

