One way to process each element of an array is to use the for-in loop.

There is another older way - that is occasionally still needed:

(1) **When an array element needs to be changed.**
(2) When elements of two arrays are to be matched against each other
(3) When the elements need to be processed in reverse order
(4) When elements need to be skipped in an orderly manner.
(5) When elements near the beginning (or the end) of the array should NOT be processed.
(6) And lots of other reasons.

We will call this older way: "Subscript Looping"
Suppose that \texttt{data} is an array of integers. Its declaration would be:

\begin{verbatim}
int[] data;
\end{verbatim}

Here is the \texttt{for-in} loop to print its elements in a column:

\begin{verbatim}
for(int elem : data) {
    out.println(elem);
}
\end{verbatim}

Here is the older Subscript Looping

\begin{verbatim}
for(int k=0;k<data.length;++k) {
    int elem = data[k];
    out.println(elem);
}
\end{verbatim}
The previous page also introduces an older version of the for loop - just called a "for" loop. It is very generic:

```
for( init-part ; while-part ; advance-part ) {
   body-steps
}
```

and is completely equivalent to the following while loop:

```
init-part

while( while-part ) {
   body-steps
   advance-part
}
```

NOTE: the advance part appears at the BOTTOM of the seq of the while and is BELOW the body-steps.
There are three ways to say "add 1 to variable x". They have a few subtle differences that beginners can probably ignore:

\[
x = x + 1; \\
x += 1; \\
++x;
\]

You should probably avoid using the fourth way to add 1:

\[x++;\]

Because it is often misunderstood by both beginners and practicing programmers (even some faculty).
Here is the older way to process each element of the array named data.

```
for( int k = 0 ; k < data.length ; ++k ) {
    int elem  =  data[k]  ;
    out.println(elem);
}
```

1. **Subscript variable**: `k`
2. **First subscript value**: 0
3. **Limit (too far) for subscript values**: `data.length`
4. **Add 1 to subscript**: `++k`
5. **DONE AFTER steps of loop body**
6. **Element variable**: `elem`
7. **Extract element value from array at position k**: `data[k]`
8. **Process current array element**: `out.println(elem)%;

Exploded Subscript Looping
Tuesday, February 21, 2012
10:50 AM

Subscript Loops Page 5
Here is how to process every other element of an array. For variety, this array is named stuff, and the element variable is named item and the subscript variable is named s. The loop starts with the first element at subscript #0

```java
for( int s=0 ; s<stuff.length ; s=s+2 ) {
    int item = stuff[s];
    // do something with item
}
```

Some possible things to do with item - depending on what you need done:

1. Add item to a total
2. Count item if it is in particular range
3. Print item
Here is how to process every other element of an array. For variety, this array is named list, and the element variable is named member and the subscript variable is named j.
The loop starts with the second element at subscript #1

```java
for( int j=1 ; j<list.length ; j=j+2 ){
    int member=list[j];
    // do something with member
}
```
Here is how to add together consecutive pairs of elements of an array named `row` and print them in a column. Thus if the array `row` had the numbers 3, 5, 1, 8, 9, 21 in it then the console would show the numbers 8, 9, 30 in a column.

```java
for(  int n = 0;   n+1 < row.length;  n=n+2  ) {
    int sum = row[n] + row[n+1] ;
    out.println(sum);
}
```

**NOTE:** row[n] is the element at position n, but row[n+1] is the element at the next position to the right.

**NOTE:** the while condition uses n+1 to be sure that the last element of row is ignored if row's length is an odd number.
Here there are TWO arrays that are presumed to be available - one named stuff and the other named data. It is assumed they have the same length. It is desired to print out the sum of each matching pair of numbers on a single line with a comma in front of each sum.

Hence if stuff were \{3,8,5\} and data were \{7,20,2\} then the console output would be:

,10,28,7

```
for(int k=0; k<stuff.length; ++k ) {
    int sum = stuff[k] + data[k] ;
    out.print(","+sum);
}
out.println();
```

NOTES:

(1) Subscript k advances by 1 step each loop repetition.
(2) k is used to extract elements from both stuff and data at the same position.
(3) out.print is used to ensure all outputs are on the same line.
(4) out.println is ONLY used at the end when the loop has finished.
Here it is desired to change each third element of an array data to be zero.
Changing a number to zero is known as "clearing" it.
The first element to be cleared will be data[2] - don't forget subscripts start at 0.

```
for(int k=2 ; k < data.length ; k=k+3 ) {
    data[k] = 0;
}
```

NOTES:
(1) The usual step
    int elem = data[k];
    is missing
(2) This is an example of CHANGING an array.
This creates an array named `powers` whose `.length` is 10 and which has consecutive powers of two stored in it so that

```
powers[k] is 2 raised to the k-th power
```

```
int value = 1;  // 2 to the 0-th is 1
int[] powers = new int[10];  // create array with 10 elements
for (int k=0; k<10; ++k) {
    powers[k] = value;
    value = value * 2;
}
// Already know length of powers
```

**NOTES:**
(1) There are TWO loop variables:
(2) The variable `k` will progress 0,1,2,3,...,10
(3) But the 10 will NOT be used.
(4) The variable `value` will progress 1,2,4,8,....
(5) Previous example presumed the array was already available - but this example CREATES it.