A Java array is kind of like a credit card holder

Sleeves to put credit cards in

Outer case of holder
However, the "sleeves" of a Java array are numbered in permanent ink on the outside corners.

The numbering goes 0, 1, 2, 3, ...
INSTEAD of 1, 2, 3, 4, ...
Sleeve numbers are called **SUBSCRIPTS**

Sleeves are called **subscripted ELEMENTS**

Numbers IN sleeves are called **subscripted VALUES**

**Data item IN sleeve #0**

**Data item IN sleeve #1**

**Data item IN sleeve #2**
Using the "Easy" library, an array can be input from the user's console as ONE THING.

Here is an input command with the usual four parts:

\[
\begin{align*}
\text{int[]} & \quad \text{mystuff} & \quad = & \quad \text{cin.nextIntArray()} ; \\
\text{MAKE} & & \text{NAME} & & \text{VALUE} & & \text{PUT}
\end{align*}
\]
Make a variable inside of which ONE integer array can be kept - remember that an array is like a wallet with a number in each sleeve.

The name for the ONE THING which is an array of integers - mystuff is the ONE name of the WHOLE wallet.

The "Easy" operation to read a SINGLE array of integers from the user's console - The input must be in brackets notation. (see next page)

As usual PUT is done last. The VALUE on the right is to stored into the variable on the left.
Brackets Notation is used by the Easy library to allow the user to print and input an ENTIRE array to and from the console.

Here is an example of brackets notation:

\[
[5, -17, 6, 5, 2]
\]

- Commas represent the spaces between the "sleeves" in the "wallet"
- Integer in sleeve #0
- Integer in sleeve #1
- Integer in sleeve #2
- Integer in sleeve #3
- Integer in sleeve #4
- Represents outer "case" of the "wallet"
Five "sleeves"  
Hold five integers

TWO NOTES:

(1) Order of numbers - first to last is IMPORTANT
(2) Numbers CAN be repeated.

REMEMBER:
"sleeve" = array element
"sleeve index" = subscript
Using the "Easy" library, an array can be printed onto the user's console as ONE THING.

Here is an output command with the usual four parts:

```java
out.println("an array=")

content(mystuff)
```

The usual print to the console command

A legend of some kind

Name of the variable wherein the array has been PUT.

Special "Easy" operation to put array into brackets notation

See page named "Easy Array Input"
The simple program below (just the main procedure) prompts the user to enter a single array of integers (in bracket notation) and then copies that same array back to the console (again in bracket notation).

```java
out.print("Enter brackets array: ");
int[] listx = cin.nextIntArray();
out.println("You entered this array = " + content(listx) );
```

AGAIN NOTE: The ENTIRE array is input as ONE THING and is printed out as ONE THING.

Below is a snapshot of the user's console for running the above program

Enter brackets array: [ 3 , 2, -5,6,2,1]
You entered this array = [3, 2, -5, 6, 2, 1]

Note user input is in blue.
It can be necessary to process the content of each individual "sleeve" of an array one at a time.

One way to do this is to use a `for-in` loop:

```java
for ( int item : mydata ) {
    // process the ONE sleeve content named item
}
```

This command block is exploded on the next page.
for (int item : mydata) {
    // java steps that will
    // process just ONE sleeve content named item
}

The array of ALL those "sleeves" that are going to be processed - one at a time.

A loop variable name to hold just ONE sleeve value at a time.

Type of each item to be found INSIDE each sleeve of the array.

Keywords and punctuation for a for-in command.

for-in command
Suppose we have an integer array named mystuff and want to print out each "sleeve" element (to be named x) one at a time - each on a line by itself - namely all the elements will appear in one column. Here is one way to do that:

```java
for( int x : mystuff ) {
    out.println(x);
}
```

**NOTES:**
(1) The loop variable name is x
(2) x will hold one integer at a time
(3) The array name is mystuff
(4) Each "sleeve" of mystuff will hold just one integer
(5) Each x will be printed (one at a time) by the command:
    `out.println(x);`
The processing put inside a for-in loop can be any of the models you have already learned - or any models you learn in the future - or any models you invent yourself.

This java code adds up and prints out the sum of the real number elements in the sleeves of the the array named grades - which is read in from the user.

```java
out.print("Enter grades in [] : "); // prompt
double[] grades = cin.nextDoubleArray(); // input WHOLE array
double sum = 0; // start total at 0
for(double g : grades) {
    sum = sum + g; // add grade g to total - one at a time
} // marks end of for-in loop
out.println("Total was "+sum); // tell user the total of all the grades
```
One way to get an array is to input one from the user by a command like:

```java
int[] mylist = cin.nextIntArray();
```

Another way is to use an array `initializer` in place of the user input operation:

```java
int[] mylist = { 5 , 6 , 1 , 3 };
```

This has the usual parts: MAKE, NAME, PUT, VALUE
Except that the VALUE is a list of the array element values in braces.
YES - sometimes Java uses braces for arrays and sometimes it uses brackets - SORRY! ! !
So far: brackets are for input/output and braces are for writing java program commands.
Given an array - you can refer to parts INSIDE the array.

First, you can refer to the number of "sleeves" in the array by using the notation .length

Then you can refer to each number INSIDE a sleeve of the array using the notations:

[0]  
[1]  
[2]  
[3]  
[4]  
...

The number xx inside the brackets [ xx ] is called a subscript - it is just the index number permanently written on the outside of each sleeve - see the page named "Wallet Torn Apart".
For example, if we had an array named theStuff and wanted to print out the product of the first number and second number INSIDE that array, we could write:

```java
out.println("product=\"+ theStuff[0] * theStuff[1] \");
```

Recall that the sleeves are numbered starting from 0 rather than 1.
This program code reads a series of integer arrays until the user enters an empty array. For each such array (except the empty one) it prints out the product of the first two numbers in that array. It is assumed that the user will not try to mess-up the program by entering an array with only one element.

```java
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0] * thedata[1];
    out.println("product="+product);
}
```

Detailed explanation appears in the following pages
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0]  *  thedata[1] ;
    out.println("product="+product);
}

Sentinel value "infinite" loop
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0] * thedata[1] ;
    out.println("product="+product);
}

NOTE: input gets A WHOLE ARRAY
(but which usually has only two elements)
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if( thedata.length == 0 ) break;
    int product = thedata[0] * thedata[1];
    out.println("product=\"+product\")
}

Sentinel value break - note use of .length notation to see how many sleeves are in array. Empty array will have a length of 0.

Breaks out of loop when user enters an array with NO elements in it - i.e. user types:

    []
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0] * thedata[1] ;
    out.println("product=\"+product);
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0] * thedata[1];
    out.println("product="+product);
}

Second array element is in sleeve number 1.
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0] * thedata[1];
    out.println("product=\"+product);
while(true) {
    out.print("Enter two nos in []: ");
    int[] thedata = cin.nextIntArray();
    if(thedata.length==0) break;
    int product = thedata[0] * thedata[1] ;
    out.println("product=\"+product);  
}

Show user the product for most recently entered array.

Thus there will be an output line in the console for each one of the input arrays that the user types - except for the last empty input array typed as [ ]
User input is in blue

Enter two nos in []: [4,3]  
product=12
Enter two nos in[]: [12,7]  
product=84
Enter two nos in []: [7,3,5,9]  
product=21
Enter two nos in []: [20,11]  
product=220
Enter two nos in []: []

Program ends after last input - namely the []

User CAN enter MORE than two numbers if they want.