Array:

Bash arrays have numbered indexes only, but they are sparse, ie you don't have to define all the indexes. An entire array can be assigned by enclosing the array items in parenthesis:

```
arr=(Hello World)
```

Individual items can be assigned with the familiar array:

```
arr[0]=Hello
arr[1]=World
```

But it gets a bit ugly when you want to refer to an array item:

```
echo ${arr[0]} ${arr[1]}
```

To quote from the man page: The braces are required to avoid conflicts with pathname expansion.

In addition, the following constructs are available:

```
${arr[*]}       # All of the items in the array
${!arr[*]}      # All of the indexes in the array
${#arr[*]}      # Number of items in the array
${#arr[0]}      # Length of item zero
```

A Sample Script Using Array:

```
#!/bin/bash
array=(one two three four [5]=five)
echo "Array size: ${#array[*]}"
echo "Array items:"
for item in ${array[*]}
do
   echo  “$item”
done

echo "Array indexes:"
for index in ${!array[*]}
do
   echo “$index”
done

echo "Array items and indexes:"
for index in ${!array[*]}
do
   #we can use ‘printf’ in bash script
   printf "%4d: %s\n" $index ${array[$index]}
done
```

Output:

```
Array size: 5
Array items:
  one
  two
  three
  four
  five
Array indexes:
  0
  1
  2
  3
  5
Array items and indexes:
  0: one
  1: two
  2: three
  3: four
  5: five
```

Declare an array take input and print the odd elements of the array:

```
declare -a b;
for ((i=0; i<=10; i++))
do
        read b[$i];
```

Output:

```
1
3
5
7
9
```

```
done

for ((i=0; i<=10; i++))
do
if (( (("b[$i]" % 2)) == 1 ))
       then
           echo ${b[$i]};
fi
done
```