```
        echo Item: $item
    done

    IFS=$oldIFS
```

The output is as follows:

```
Item: name
Item: sex
Item: rollno
Item: location
```

The default value of IFS is a space component (newline, tab, or a space character).

When IFS is set as `,` the shell interprets the comma as a delimiter character, therefore, the `$item` variable takes substrings separated by a comma as its value during the iteration.

If IFS is not set as `,` then it would print the entire data as a single string.

## How to do it...

Let us go through another example usage of IFS by taking the `/etc/passwd` file into consideration. In the `/etc/passwd` file, every line contains items delimited by `":"`. Each line in the file corresponds to an attribute related to a user.

Consider the input: `root:x:0:0:root:/root:/bin/bash`. The last entry on each line specifies the default shell for the user. To print users and their default shells, we can use the IFS hack as follows:

```
#!/bin/bash
#Desc: Illustration of IFS
line="root:x:0:0:root:/root:/bin/bash"
oldIFS=$IFS;
IFS=":"
count=0
for item in $line;
do

     [ $count -eq 0 ]  && user=$item;
     [ $count -eq 6 ]  && shell=$item;
    let count++
done;
IFS=$oldIFS
echo $user\'s shell is $shell;
```

The output will be:

```
root's shell is /bin/bash
```

Loops are very useful in iterating through a sequence of values. Bash provides many types of loops. Let us see how to use them:

- Using a `for` loop:

  ```
  for var in list;
  do
      commands; # use $var
  done
  list can be a string, or a sequence.
  ```

  We can generate different sequences easily.

  `echo {1..50}` can generate a list of numbers from 1 to 50. `echo {a..z}` or `{A..Z}` or `{a..h}` can generate lists of alphabets. Also, by combining these we can concatenate data.

  In the following code, in each iteration, the variable `i` will hold a character in the range `a` to `z`:

  ```
  for i in {a..z}; do actions; done;
  ```

  The `for` loop can also take the format of the `for` loop in C. For example:

  ```
  for((i=0;i<10;i++))
  {
      commands; # Use $i
  }
  ```

- Using a `while` loop:

  ```
  while condition
  do
      commands;
  done
  ```

  For an infinite loop, use `true` as the condition.

- Using a `until` loop:

  A special loop called `until` is available with Bash. This executes the loop until the given condition becomes true. For example:

  ```
  x=0;
  until [ $x -eq 9 ]; # [ $x -eq 9 ] is the condition
  do
      let x++; echo $x;
  done
  ```