```
Print s1[11]                 prints   (nothing)
```

A string indexed with two indices, such as s1[p1,p2], contains all of the bytes from s1[p1] to s1[p2]. For example:

```
Print s1[0,10]               prints   hello there
Print s1[-1,11]              prints   hello there
Print s1[-2,-1]              prints   (nothing)
Print s1[11,12]              prints   (nothing)
Print s1[10,0]               prints   (nothing)
```

You can use INF to mean “end of string”:

```
Print s1[10,INF]             prints   (nothing)
```

The indices in these examples are byte positions, not character positions. See **Characters Versus Bytes** on page III-483 for a discussion of this distinction. See **Character-by-Character Operations** on page IV-173 for an example of stepping through characters using user-defined functions.

Because the syntax for string indexing is identical to the syntax for wave indexing, you have to be careful when using text waves. For example:

```
Make/T textWave0 = {"Red", "Green", "Blue"}

Print textWave0[1]            prints   Green
Print textWave0[1][1]         prints   Green
Print textWave0[1][1][1]      prints   Green
Print textWave0[1][1][1][1]   prints   Green
Print textWave0[1][1][1][1][1]prints   r
```

The first four examples print row 1 of column 0. Since waves may have up to four dimensions, the first four [1]’s act as dimension indices. The column, layer, and chunk indices were out of range and were clipped to a value of 0. Finally in the last example, we ran out of dimensions and got string indexing. Do not count on this behavior because future versions of Igor may support more than four dimensions.

The way to avoid the ambiguity between wave and string indexing is to use parentheses like so:

```
Print (textWave0[1])[1]       prints   r
```

The way to avoid the ambiguity between wave and string indexing is to use parentheses like so:

```
String tmp = textWave0[1]
Print tmp[1]                  prints   r
```

String Assignment

You can assign values to string variables using string assignment. We have already seen the simplest case of this, assigning a literal string value to a string variable. You can also assign values to a subrange of a string variable, using string indexing. Once again, assume we create a string variable called s1 and assign a value to it as follows:

```
String s1="hello there"
```

Then,

```
s1[0,4]="hi"; Print s1             prints      hi there
s1[0,4]="greetings"; Print s1      prints      greetings there
s1[0,0]="j"; Print s1              prints      jello there
s1[0]="well "; Print s1            prints      well hello there
s1[100000]=" jack"; Print s1       prints      hello there jack
s1[-100]="well ";print s1          prints      well hello there
```

When the s1[p1,p2]= syntax is used, the right-hand side of the string assignment *replaces* the subrange of the string variable identified by the left-hand side, after p1 and p2 are clipped to 0 to n, where n is the number of bytes in the destination.