Print with `sep` and `end`

We have seen that when you use the print function to display lists of items you see not only the items converted to strings, but also blanks inserted as item separators, and a newline generated at the end so:

```
print(1, 2, 3)
print(4, 5, 6)
print(7, 8)
```

produces

```
1 2 3
4 5 6
7 8
```

Though blanks and newlines were inserted automatically, it turns out they are *defaults*, that the programmer can change.

We have seen Optional parameters (Optionalparameters.html#optional-parameters). In that section we had default values assigned to the last formal parameters in the heading for the definition of `find3`. In calling the function, if we shortened the actual paramter list, omitting final optional ones, then we got the default values assigned. If we lengthen the actual parameter list, we supplied values replacing in order some or all of optional parameters' default values.

The `print` function also provides defaults for formal parameters: `sep=' '` and `end=`\n'`. Since `print` takes an arbitrary number of arguments, we cannot just count arguments to the proper position to include new values for `sep` or `end`! They are *keyword only* arguments:: In calling `print`, you must specifiy the formal parameter name to assign a value as in:

```
print(1, 2, 3, sep=' and ')
print(4, 5, 6, sep = '')
print(7, 8, sep='***')
```

which produces

```
1 and 2 and 3
456
7***8
```

so the general syntax for using a keyword argument when calling a function is

```
keywordArgument=value
```

You must have all three parts. By convention this is the one place you do *not* put space around the assignment symbol `=`.

The same ideas apply to the `end` keyword argument, to replace the standard newline appended at the end of printing:

```
print(1, 2, 3, end=': also ')
print(4, 5, 6, end='')
print(7, 8, end='and then\n   ')
print(9)
```

which produces

```
1 2 3: also 4 5 67 8and then
   9
```

Look carefully at how the parts run together!

Both keyword arguments can be used together, in either order, but always as the last parameters for `print`:

```
print(1, 2, 3, sep=' and ', end=': also ')
print(4, 5, 6, end='', sep='')
print(7, 8, sep='***', end='and then\n   ')
print(9)
```

which produces

```
1 and 2 and 3: also 4567***8and then
   9
```

Note that a newline is explicitly made part of the last `end` keyword parameter value.

Common examples of use are `sep=''` when you want no space between items, and `end=' '` in a loop where you want one space between items generated in the loop like:

```
for i in range(5):
    print(i*i, end=' ')
print('is the sequence of squares.')
```

which produces

```
0 1 4 9 16 is the sequence of squares.
```

---

user not logged in