Theory: Sep and end arguments of print

16 minutes 8 / 8 problems solved Start practicing

2693 users solved this topic. Latest completion was 7 minutes ago.

You already know that to print out objects you can use the `print()` function. For example, the following lines of code will show us the value of a variable:

```
number = 5
print(number)  # 5
```

But did you know that `print()` can take multiple keyword arguments?

§1. The separator

For example, with the `sep` argument you can specify the separator between objects to be printed (the separator itself must be a string).

```
print('Chip', 'Dale', sep='&')  # Chip&Dale

print('Chip', 'Dale', sep=0)    # TypeError: sep must be None or a string, not int
```

The default value of this argument is space, that is, writing `print('Gadget', 'Hackwrench', sep=' ')` gives the same output as writing `print('Gadget', 'Hackwrench')`:

```
print('Gadget', 'Hackwrench', sep=' ')
# Gadget Hackwrench

print('Gadget', 'Hackwrench')
# Gadget Hackwrench
```

You can also use an empty string as `sep` when you want to print several objects together:

```
print(13, 'th', sep='')  # 13th
```

It will work even if you combine different values, such as integers and strings, in the example above.

§2. The end

The argument `end` determines how the string we want to print should end. The default value is `'\n'`, which means that it ends with a newline. So if you give no arguments to the function and simply write `print()`, it will shift you to a new line:

```
print('Monterey')
print()
print('Jack')
```

The output will be as follows:

```
Monterey

Jack
```

However, just as with the `'sep'` argument, we can set this argument to end with any other string.

```
print('Tick-Tock', end=' the ')
print('Crocodil', end='e')
# Tick-Tock the Crocodile
```

Current topic:

✓ Sep and end arguments of print ···

Topic depends on:

✓ Arguments Stage 1 7★ ···

Topic is required for:

✓ Flush and file arguments of print ···

Table of contents:

§3. Unpacking objects

The first arguments of the `print()` function are objects we want to print. For example, if it's a list `characters = ['Humphrey the Bear', 'Spike the Bee', 'Fat Cat']`, writing `print(characters)` will yield the output `['Humphrey the Bear', 'Spike the Bee', 'Fat Cat']`. But what if we want to print objects from a list, not the list as a whole? One way would be to use a loop:

```
for element in characters:
    print(element)
```

However, in Python, there's a more convenient and neat way to do so. Writing an asterisk * before a list means that its elements will be unpacked and passed to the function one after another:

```
print(*characters)
```

An important detail to understand, however, is that, in the first case, the elements will be printed one in one line, whereas in the second case, they will be separated by spaces. This is so because the first snippet of code is equivalent to writing these lines with the default `end="\n"`:

```
print('Humphrey the Bear')
print('Spike the Bee')
print('Fat Cat')
```

while the second one could be replaced by the line `print('Humphrey the Bear', 'Spike the Bee', 'Fat Cat')` with the default `sep=' '`:

```
print(*characters)
# Humphrey the Bear Spike the Bee Fat Cat
```

§4. Recap

We rediscovered the built-in `print()` function and examined some of its arguments, `sep` and `end`.

Note that the mentioned arguments are so-called keyword arguments. You should explicitly specify them when calling the function because all other (positional) arguments are considered as objects to be printed.

As you can see, arguments of the `print()` function provide useful ways of managing output. Keep that in mind when working with strings!

Report a typo

Thanks for your feedback!

Write here how we could improve this theory

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion