Why `print` became a function in Python 3

After writing my post on why Python 3 exists which included an explanation about the most common question/complaint about why Python 3 forced textual and binary data into separate types, I heard from people about the second most common question/complaint about Python 3: why did we make `print` a function?

Who can do what?

The `print` *statement*

In its most basic form, `print A` is the equivalent of `sys.stdout.write(str(A) + '\n')`. If you pass extra arguments separated by commas then they will also be passed to `str()` and be printed with a space between each argument. For example, `print A, B, C` is equivalent to `sys.stdout.write(' '.join(map(str, [A, B, C])) + '\n')`. If there is a trailing comma then the added newline is left off, e.g. `print A,` is the same as `sys.stdout.write(str(A))`.

Introduced in Python 2.0, the `print` chevron can be used to redirect where the `print` statement writes the final string. For instance `print >> output, A` is the same as

For instance, `print >> output, A` is the same as `output.write(str(A) + '\n')`.

The `print` *function*

The equivalent definition of the `print` function is:

```
import sys

def print(*objects, sep=None, end=None, file=None,
flush=False):
    """A Python translation of the C code for
builtins.print().


"""
    if sep is None:
        sep = ' '
    if end is None:
        end = '\n'
    if file is None:
        file = sys.stdout
    file.write(sep.join(map(str, objects)) + end)
    if flush:
        file.flush()
```

As you may have noticed, all the features of the `print` statement are covered by the `print` function.

- `print A` == `print(A)`
- `print A, B, C` == `print(A, B, C)`
- `print A,` == `print(A, end='')`
- `print >> output, A` == `print(A, file=output)`

The obvious thing the `print` function gets you that the above examples implicitly demonstrate is the ability to specify a different separator and end string compared to the syntactic `print` statement.

It's all about flexibility

But the real key to the `print` function is somewhat subtle and it all has to do with flexibility, both for the users and the Python development team. For users, making `print` a function lets you use `print` as an expression, unlike the `print` statement which can only be used as a statement. As an example, let's say you wanted to print an ellipsis after every line to indicate that more work was to be done. With the `print` statement you would have two options:

```
# Manually ...
print A, '...'

# For a reusable solution (which also works with a
functional print) ...
def ellipsis_print(*args):
    for arg in args:
        print arg, '',
    print '...'
```

But for Python 3, you have much better solutions:

```
# Manually ...
print(A, end='...\n')

# Multiple reusable solutions that won't work with a
syntactic print...
ellipsis_print = lambda *args, **kwargs: print(*args,
**kwargs, end='...\n')
# Or ...
import functools
ellipsis_print = functools.partial(print, end='...\n')
```

In other words, with `print` as a function it becomes composable while as a statement it isn't. Heck, you can even override the `print` function by assigning to `builtins.print` while you can't do that with a statement.

The flexibility that the Python development team gains is

being unshackled from having to make the features of `print` work in a syntactic fashion. For instance, if you wanted to add the same flexibility of being able to specify what string to use as the separator in the `print` statement, how would you do it? It's a rather tough design problem to solve. But with a function, you just add a new argument and you're done. The richness of Python's function parameters give a greater amount of flexibility than syntax does.

It should also be mentioned that as a general guideline, syntax is reserved for things that are either impossible to do otherwise or there is a clear readability benefit to the syntax. In the case of `print`, the difference between `print A` and `print(A)` is negligible and thus there is no loss in readability. And since we were able to completely replace `print` with a function there was no loss in functionality, hence why we reduced the amount of syntax in Python 3 by making `print` a function (along with removing a bunch of other bits of syntax).

You might also be interested in these articles...

Why Python 3 exists

Brett Cannon

December 31, 2015

Posted in: programming python