Image by RitaE from Pixabay

Do Not Use Print For Debugging In Python Anymore

A refined “print” function for debugging in Python

Christopher Tao

Jun 20·7 min read

What is the most frequently used function in Python? Well, probably in most of the programming languages, it has to be the `print()` function. I believe most of the developers like me, would use it to print messages into the console many times during the development.

Of course, there is no alternative that can completely replace the `print()` function. However, when we want to output something for debugging purposes, there are definitely better ways of doing so. In this article, I’m going to introduce a very interesting 3rd party library in Python called “Ice Cream”. It could create lots of conveniences for quick and easy debugging.

A Bad Example

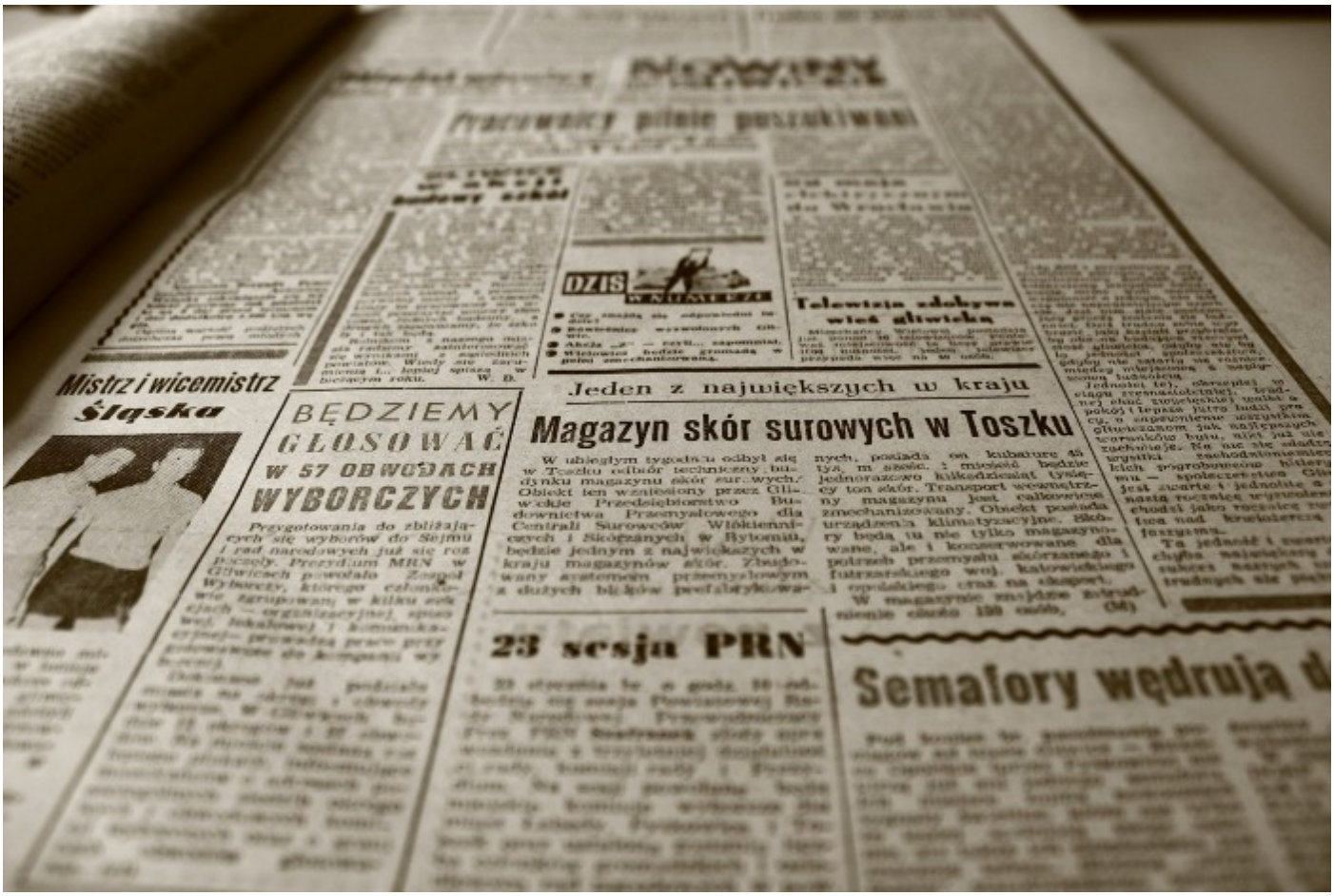

Image by Krzysztof Pluta from Pixabay

Let's start with a relatively bad example. Suppose we have defined a function and we want to see whether it works as expected.

```
def square_of(num):
    return num*num
```

This function simply returns the square of the number passed in as an argument. We may want to test it multiple times as follows.

```
print(square_of(2))
print(square_of(3))
print(square_of(4))
```

```
[2] def square_of(num):
        return num*num
```

```
[3] print(square_of(2))
    print(square_of(3))
    print(square_of(4))
```

```
4
9
16
```

This is OK for now. However, we will have much more lines of code in practice. Also, there could be many `print()` functions that print different things into the output area. In this case, sometimes we may be confused about which output is generated by which `print()` function.

Therefore, it is a good manner to add some brief description to the content of the `print()` function to remind us what it is about.

```
print('square of 2:', square_of(2))
print('square of 3:', square_of(3))
print('square of 4:', square_of(4))
```

```
[4] print('square of 2:', square_of(2))
    print('square of 3:', square_of(3))
    print('square of 4:', square_of(4))

    square of 2: 4
    square of 3: 9
    square of 4: 16
```

It is much better now, but it is too tiring to do this every time. Also, when we finish the development, very likely have to remove most of the debugging prints.

Basic Usage — Inspect Variables

Image by StockSnap from Pixabay

Let’s have a look at the Ice Cream library. How it solves the problems that were mentioned above?

First of all, we need to install it from the PyPI repository simply using `pip`.

```
pip install icecream
```

Then, let’s import the library as follows.

```
from icecream import ic
```

Now, we can use it for everything we want to print as debug information.

Call a Function

We can directly use ice cream to print a function just like what we have done using the `print()` function previously.

```
ic(square_of(2))
ic(square_of(3))
ic(square_of(4))
```

```
[5] from icecream import ic
```

```
[6] ic(square_of(2))
    ic(square_of(3))
    ic(square_of(4))
```

```
ic| square_of(2): 4
ic| square_of(3): 9
ic| square_of(4): 16
```

Great! We never specify anything in the `ic()` function, but it automatically outputs the function name and argument together with the outcome. So, we don't have to manually add the "brief description" anymore.

Access a Dictionary

Not only calling a function, but Ice Cream can also output everything verbose that is convenient for debugging purpose, such as accessing a key-value pair of a dictionary.

```
my_dict = {
    'name': 'Chris',
    'age': 33
}ic(my_dict['name'])
```

```
[7] my_dict = {
        'name': 'Chris',
        'age': 33
    }

    ic(my_dict['name'])
```

```
ic| my_dict['name']: 'Chris'
```

In this example, I have defined a dictionary and try to access a value in it from its key. The Ice Cream output both the variable name of the dictionary and the key that I was accessing.

Access Attributes of an Object

One more example, let's define a class and instantiate an object from it.

```
class Dog():
    num_legs = 4
    tail = Truedog = Dog()
```

Now, let's use Ice Cream to output an attribute of it.

```
ic(dog.tail)
```

[8]
```
class Dog():
    num_legs = 4
    tail = True

dog = Dog()
```

[9]
```
ic(dog.tail)
```

```
ic| dog.tail: True
```

Debug in If-Condition

Image by silviarita from Pixabay

The Ice Cream library is not only useful for inspecting a variable, but also in a control statement such as an if-condition. For example, let's write a simple if-else condition as follows.

```
input = 'Chris'if input == 'Chris':
    ic()
else:
    ic()
```

We just put the Ice Cream function in the if and else blocks, see what happen.

```
[10]  1 input = 'Chris'
      2
      3 if input == 'Chris':
      4     ic()
      5 else:
      6     ic()

ic| <ipython-input-10-364104949a0a>:4 in <module> at 12:25:22.332
```

Although the if-else statement does nothing at the moment, the `ic()` function still tells us where and when it has been called, as well as the line number.

BTW, I'm using Python Notebooks for this demo. If this is running in a ".py" file, it will also tell us the file name that it was called from.

Let's consider a more practical usage as follows.

```
def check_user(username):
    if username == 'Chris':
        # do something
        ic()
    else:
        # do something else
        ic()check_user('Chris')
check_user('Jade')
```

The function will do something for different user. For debugging purposes, we always want to know which is the current user. Then, the `ic()` function will always tell us that.

```
[11]  1 def check_user(username):
      2     if username == 'Chris':
      3         # do something
      4         ic()
      5     else:
      6         # do something else
      7         ic()
      8
      9 check_user('Chris')
     10 check_user('Jade')
```

```
ic| <ipython-input-11-8a6ba6981490>:3 in check_user() at 12:25:22.429
ic| <ipython-input-11-8a6ba6981490>:5 in check_user() at 12:25:22.497
```

Insert Into Existing Code

Image by StockSnap from Pixabay

This cool feature of the Ice Cream library needs to be highlighted in my opinion. That is, the `ic()` function will not only output the verbose information but also pass the value through so that it can be a wrap of anything. In other words, we can put the `ic()` function to anything in our code without affecting it.

Let's keep using the `sqaure_of()` function that we defined in the previous section.

```
num = 2square_of_num = square_of(ic(num))
```

```
[12] num = 2

     square_of_num = square_of(ic(num))

     ic| num: 2
```

In this example, suppose we have a variable `num` and we want to calculate its square. Instead of `square_of(num)`, I put the `ic()` function out of the variable `num`. Therefore, the value of the variable `num` is printed, and the result assigned to `square_of_num` will not be affected.

We can test the result as follows.

```
if ic(square_of_num) == pow(num, 2):
    ic('Correct!')
```

```
[13] if ic(square_of_num) == pow(num, 2):
         ic('Correct!')

     ic| square_of_num: 4
     ic| 'Correct!'
```

Therefore, `square_of_num` equals to the square of the variable `num`. Also, in this if-condition, I also used the `ic()` function without affecting the purpose, but the variable `square_of_num` is printed for debugging!

Disable Ice Cream

Image by Free-Photos from Pixabay

One of the biggest issue when using the `print()` function for debugging is that there are too many of them. It is very common that we have them everywhere when we finished the development. Then, it would be a disaster if we want to clean our code to remove them.

If we're using the Ice Cream library for debugging, what we need to do is simply disable it.

```
ic.disable()
```

After that, all the `ic()` function will stop output anything. For example, the code below will output nothing.

```
[15] if ic(square_of_num) == pow(num, 2):
         ic('Correct!')
```

You may ask that how about the variable `square_of_num`? Will it still be passed through if we disabled the Ice Cream function? Don't worry, the disabling feature will only disable the output, we don't need to worry about any other features.

```
if ic(square_of_num) == pow(num, 2):
    print('Correct!')
```

If we change the output back to the `print()` function, it still can be output. That means the `ic(square_of_num)` still equivalent to `square_of_num`.

```
[16] if ic(square_of_num) == pow(num, 2):
         print('Correct!')

     Correct!
```

Of course, if we want to go back to the debug mode, the Ice Cream can be re-enabled.

```
ic.enable()
```

Customising Ice Creame Output

Image by Jan Vašek from Pixabay

The Ice Cream can also be customised for its output. The most commonly used customisation would be changing the prefix. You may have noticed that the default output always has the prefix `ic | `. Yes, we can customise it.

For example, we can change it to `Debug | ` which makes more sense for its debugging purpose.

```
ic.configureOutput(prefix='Debug | ')
ic('test')
```

```
[18] ic.configureOutput(prefix='Debug | ')
     ic('test')

     Debug | 'test'
```

In fact, rather than a static string, the prefix can also be set to a function. For example, let's define a function that returns the current timestamp in a formatted string.

```
from datetime import datetimedef now():
    return f'[{datetime.now()}] '
```

Then, we can set that function as the Ice Cream prefix.

```
ic.configureOutput(prefix=now)
ic('test')
```

```
[19] from datetime import datetime

     def now():
         return f'[{datetime.now()}] '

     ic.configureOutput(prefix=now)
     ic('test')
```

```
[2021-06-20 12:25:22.838763] 'test'
```

Summary

Image by Seksak Kerdkanno from Pixabay

In this article, I have introduced an awesome 3rd party library for Python called “Ice Cream”. It enhanced the regular `print()` function of Python with verbose output. Therefore, it makes debugging very convenient.

The Ice Cream library will never replace the `print()` function, because it is designed for debugging purposes. Also, it does not mean to replace the logging system as well. In my opinion, it is in between these two. Check out and try it out!

Life is short, use Python!

From: https://towardsdatascience.com/do-not-use-print-for-debugging-in-python-anymore-6767b6f1866d