Created in Canva | Edited by
Varun Singh

Python was built with one special programming style and that was — using underscores more than we ever used in any other programming language before. It was called the **Snake Case**.

What if I tell you that, recently I have come to know 3 cool features of ***underscores*** that very few of us know out there and it has helped me become a better Python developer lately. Let's cut to the chase and I am listing these 3 features of underscores that we all wish we knew before —

1. Recover Result from the Last Session in REPL

Underscores(_) can be used in the Python session to refer to the last return result. Yes, this becomes really helpful when we forget to assign the result to some variable. Also, this can help if you are running a very slow computation or a function and forgot to catch the result —

```
>>> 1 + 1
2
>>> _
2
>>> sum(range(100_000_000))     # Takes couple of seconds to finish.
4999999950000000
>>> _
4999999950000000
```

So, next time you are playing around REPL and forget to assign the result of a function call, or some other piece of code, remember to use _ to refer back to it.

2. Magic Functions using Underscores

Many of us, when started learning Python, were afraid from one thing — The weird function names starting and ending with double underscores. For example —

```
>>> my_string= 7
>>> str(my_string) '3'
>>> my_string.__str__() '3'
```

For example, many functions like `__str__`, `__repr__`, `__bool__`, and `__init__`, are sometimes referred to as “magic” functions because they interact, in some way, with Python's “internal” functioning.

Here, if you would notice, `my_string.__str__()` is the same as writing `str(my_string)`. The only difference is that the latter is more readable and developer-friendly.

A better name for these magic functions and variables is “dunder function”, or “dunder variable”, or “dunder method”, depending on the context. (The word “dunder” — a common word in the Python world — is short for “double underscore”!)

3. Underscore as a Sink

One of my favorite use cases for the underscore is when we use the underscore as the target for an assignment.

It is a widely-spread convention that using _ as a variable name means “I don't care about this value”. Having said this, you should be asking yourself this: If I don't care about a value, why would I assign it in the first place? Excellent question!

For example —

```
_ = 5      # I don't care about this 5.
```

More About The Author😄

I am a full-time software engineer with 4+ years of experience in Python, AWS, and many more technologies. I have started writing recently as it helps me to read more. I am looking forward to sharing technical knowledge and my life experiences with people out there.

Register Here for my Programmer Weekly newsletter where I share interesting tutorials, python libraries, tools, etc.