

You have 2 free member-only stories left this month. Sign up for Medium and get an extra one



How to Catch Multiple Exceptions in Python

Handling multiple exceptions in Python



Photo by **CHUTTERSNAP** on **Unsplash**

A well developed application must always be capable of handling unexpected events — such as exceptions — in a proper way. This is also important when it comes to debugging your source code during development, but also when it comes to inspecting the application logs when the application is up and running (eventually in production).

In today's short tutorial we will showcase how one can handle multiple Exceptions in Python. We will also explore some new features in Python that can help you do so in a more intuitive and clean way. Let's get started!

For Python < 3.11

Now let's suppose that we have the following (fairly dumb) code snippet, that raises AttributeError, ValueError and TypeError.

```
def my_function(x):
    if x == 1:
        raise AttributeError('Example AttributeError')
    elif x == 2:
        raise ValueError('Example ValueError')
    elif x == 3:
        raise TypeError('Example TypeError')
    else:
        print('Hello World')
```

And let's also suppose that we want to to call the function <code>my_function</code> but at the same time, we also need to ensure that we handle any unexpected errors appropriately. To do so, we can use the <code>try-except</code> clauses.

But let's assume that we want to perform a certain action if an AttributeError is being thrown and a different action when either of ValueError or TypeError are raised by my_function.

```
try:
    my_function(x)
except AttributeError:
    # Do something
    ...
except (ValueError, TypeError):
    # Do something else
    ...
```

A \underline{try} statement may have more than one except clause, to specify handlers for different exceptions. At most one handler will be executed. Handlers only handle exceptions that occur in the corresponding try clause, not in other handlers of the same \underline{try} statement. An except clause may name multiple exceptions as a parenthesized tuple.

— <u>Python Docs</u>

If you are not planning to do anything special with any of the errors being raised (e.g. you just pass) you can even use the suppress Context Manager as illustrated below:

Open in app 7

(Sign up

Sign In



Search Medium



Note that <u>suppress()</u> is available from Python 3.4 onwards. Additionally, you must only use this approach only if you want the specific part of your program to fail silently and continue with the execution. In most of the cases though, you may wish to take certain actions upon certain exceptions.

Handling multiple exceptions with Python ≥ 3.11

As of Python 3.11, a new standard exception type was introduced, namely ExceptionGroup. This new exception is used to propagate together a group of unrelated exceptions.

In the example create below, we create an instance of ExceptionGroup containing four different types of errors, namely TypeError, ValueError, KeyError and AttributeError. We then use multiple except* clauses to handle ExceptionGroup either for individual exception types or multiple ones.

except* (KeyError, AttributeError) as e:
...

Note though that exceptions raised in one except* clause are not eligible to match other clauses from the same try statement

For more details around the rationale behind ExceptionGroup and except* clause you can refer to PEP-654.

Additionally, for a more comprehensiv

Python 3.11 — including the one we discussed carrier — you can refer to one of my recent articles shared below.

What to Expect in Python 3.11

Exploring the new additions and updates in Python 3.11 and how to get early access to 3.11 Alpha

towardsdatascience.com

Final Thoughts

In today's short tutorial we showcased various different approaches when it comes to handling multiple exceptions in Python. We've seen how to catch multiple exceptions using the traditional except clause but we also showcased how to do so using the new except* clause that will be introduced in Python 3.11.

As a final note, you should always remember that handling unexpected events throughout your source code is an important aspect that can significantly increase the code quality when performed properly.

<u>Become a member</u> and read every story on Medium. Your membership fee directly supports me and other writers you read. You'll also get full access to every story on Medium.

gmyrianthous.medium.com	
elated articles you may also like	
How to Write Switch Statements in Python Understanding how to write switch statements in Python using pattern matching or dictionaries towardsdatascience.com	
How To Merge Pandas DataFrames Performing left, right, inner and anti joins with pandas DataFrames towardsdatascience.com	
requirements.txt vs setup.py in Python Understanding the purpose of requirements.txt, setup.py and setup.cfg in Python when developing and distributing towardsdatascience.com	

As a Medium member, a portion of your membership fee goes to writers you

read, and you get full access to every story...

Python

Programming

Software Development

Data Science

Machine Learning

Enjoy the read? Reward the writer. Beta

Your tip will go to Giorgos Myrianthous through a third-party platform of their choice, letting them know you appreciate their story.

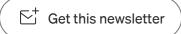


Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. <u>Take a look.</u>

By signing up, you will create a Medium account if you don't already have one. Review our <u>Privacy Policy</u> for more information about our privacy practices.



About Help Terms Privacy

Get the Medium app



