Python Raise Exception

Summary: in this tutorial, you'll learn how to raise exceptions by using the Python `raise` statement.

Introduction to the Python raise statement

To raise an exception (https://www.pythontutorial.net/python-oop/python-exceptions/), you use the `raise` statement:

```
raise ExceptionType()
```

The `ExceptionType()` must be subclass of the `BaseException` class. Typically, it is a subclass of the `Exception` class. Note that the `ExceptionType` doesn't need to be directly inherited from the `Exception` class. It can indirectly inherit from a class that is a subclass of the Exception class.

The `BaseException` class has the `__init__` method that accepts an `*args` argument. It means that you can pass any number of arguments to the exception object when raising an exception.

The following example uses the `raise` statement to raise a `ValueError` exception. It passes three arguments to the `ValueError` `__init__` method:

```
try:
    raise ValueError('The value error exception', 'x', 'y')
except ValueError as ex:
    print(ex.args)
```

Output:

```
('The value error exception', 'x', 'y')
```

Reraise the current exception

Sometimes, you want to log an exception and raise the same exception again. In this case, you can use the `raise` statement without specifying the exception object.

For example, the following defines a `division()` function that returns the division of two numbers:

```
def division(a, b):
    try:
        return a / b
    except ZeroDivisionError as ex:
        print('Logging exception:', str(ex))
        raise
```

If you pass zero to the second argument of the `division()` function, the `ZeroDivisionError` exception will occur. However, instead of handling the exception, you can log the exception and raise it again.

Note that you don't need to specify the exception object in the `raise` statement. In this case, Python knows that the `raise` statement will raise the current exception that has been caught by the `except` clause.

The following code causes a `ZeroDivisionError` exception:

```
division(1, 0)
```

And you'll see both the logging message and the exception in the output:

```
Logging exception: division by zero
Traceback (most recent call last):
  File "c:/pythontutorial/app.py", line 9, in <module>
    division(1, 0)
  File "C:/pythontutorial/app.py", line 3, in division
    return a / b
ZeroDivisionError: division by zero
```

Raise another exception during handling an exception

When handling an exception, you may want to raise another exception. For example:

```
def division(a, b):
    try:
        return a / b
    except ZeroDivisionError as ex:
        raise ValueError('b must not zero')
```

In the `division()` function, we raise a `ValueError` exception if the `ZeroDivisionError` occurs.

If you run the following code, you'll get the detail of the stack trace:

```
division(1, 0)
```

Output:

```
Traceback (most recent call last):
  File "C:/pythontutorial/app.py", line 3, in division
```

```
    return a / b
ZeroDivisionError: division by zero

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:/pythontutorial/app.py", line 8, in <module>
    division(1, 0)
  File "C:/pythontutorial/app.py", line 5, in division
    raise ValueError('b must not zero')
ValueError: b must not zero
```

First, the `ZeroDivisionError` exception occurs:

```
Traceback (most recent call last):
  File "C:/pythontutorial/app.py", line 3, in division
    return a / b
ZeroDivisionError: division by zero
```

Second, during handling the `ZeroDivisionError` exception, the `ValueError` exception occurs:

```
Traceback (most recent call last):
  File "C:/pythontutorial/app.py", line 8, in <module>
    division(1, 0)
  File "C:/pythontutorial/app.py", line 5, in division
    raise ValueError('b must not zero')
ValueError: b must not zero
```

Summary

- Use the Python `raise` statement to raise an exception.
- When handling exception, you can raise the same or another exception.