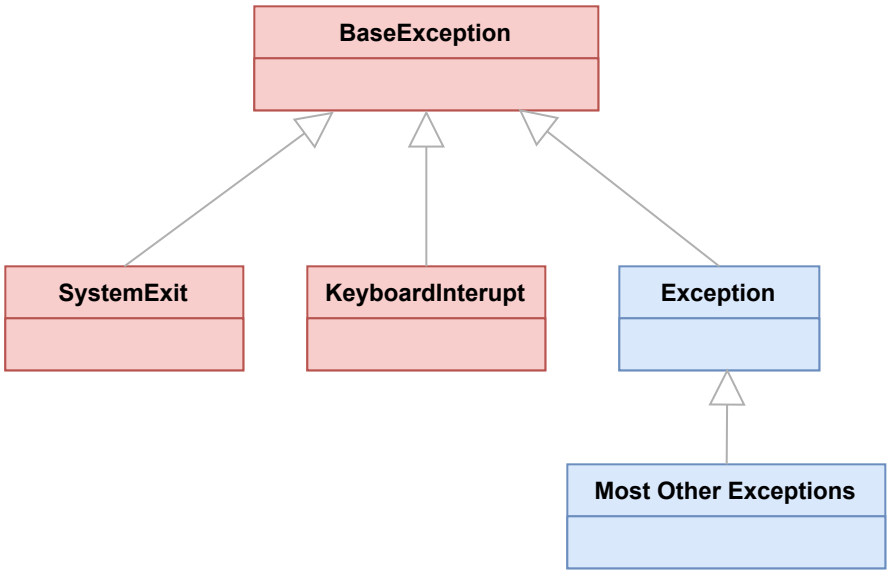
Python Exceptions

Summary: in this tutorial, you'll learn about the Python exceptions and how to handle them gracefully in programs.

Introduction to Python exceptions

In Python, exceptions are objects of the exception classes (https://www.pythontutorial.net/python-oop/python-class/) . All exception classes are the subclasses of the `BaseException` class.

However, almost all built-in exception classes inherit from the `Exception` class, which is the subclass of the `BaseException` class:



This page shows a complete class hierarchy for built-in exceptions in Python (https://docs.python.org/3/library/exceptions.html#exception-hierarchy) .

The following example defines a list of three elements and attempts to access the fourth one:

```
colors = ['red', 'green', 'blue']

print(colors[3])
```

The invalid index caused the `IndexError` exception as expected:

```
IndexError: list index out of range
```

When an exception occurs, Python stops the program unless you handle it. To handle an exception, you use the `try...except` (https://www.pythontutorial.net/python-basics/python-try-except/) statement. For example:

```
colors = ['red', 'green', 'blue']

try:
    print(colors[3])
except IndexError as e:
    print(e)
```

```
print('Continue to run')
```

Output:

```
<class 'IndexError'> - list index out of range
Continue to run
```

In this example, we use the `try...except` statement to handle the `IndexError` exception. As you can see from the output, the program continues to run after the `try...except` statement.

The `IndexError` class inherits from the `LookupError` class which inherits from the `Exception` class:

And you can catch either `LookupError` or `Exception` class when an `IndexError` exception occurs. For example:

```
colors = ['red', 'green', 'blue']

try:
    print(colors[3])
except LookupError as e:
    print(e.__class__, '-', e)

print('Continue to run')
```

Output:

```
<class 'IndexError'> - list index out of range
Continue to run
```

In this example, the exception is still `IndexError` even though we catch the `LookupError` exception. Therefore, when you handle an exception, the exception handler will catch the exception type you specify and any of its subclasses.

The program runs the same if you use the `Exception` class instead of the `LookupError` class:

```
colors = ['red', 'green', 'blue']

try:
    print(colors[3])
except Exception as e:
    print(e.__class__, '-', e)

print('Continue to run')
```

Output:

```
<class 'IndexError'> - list index out of range
Continue to run
```

In practice, you should catch the exceptions as specific as possible so that you know how to deal with each exception in a specific way.

Python exception handling example

The following example defines a `division` function that returns the result of a is divided by b:

```
def division(a, b):
    return a / b


c = division(10, 0)
```

Output:

```
ZeroDivisionError: division by zero
```

In this example, if b is zero, the `ZeroDivisionError` exception will occur. To handle the `ZeroDivisionError` exception, you use the `try...except` statement as follows:

```
def division(a, b):
    try:
        return {
            'success': True,
            'message': 'OK',
            'result': a / b
        }
    except ZeroDivisionError as e:
        return {
            'success': False,
            'message': 'b cannot be zero',
            'result': None
```

```
        }


result = division(10, 0)
print(result)
```

In this example, the function returns a dictionary that has three elements:

- `success` is a boolean value that indicates whethere the operation is successful or not.
- `message` indicates the error or success message.
- `result` stores the result of a / b or `None` if b is zero.

The following shows the output if the `ZeroDivisionError` occurs:

```
{'success': False, 'message': 'b cannot be zero', 'result': None}
```

Now, if you don't catch the `ZeroDivisionError` exception but the more general exception like `Exception` class:

```
def division(a, b):
    try:
        return {
            'success': True,
            'message': 'OK',
            'result': a / b
        }
    except Exception as e:
        return {
            'success': False,
            'message': 'b cannot be zero',
            'result': None
        }
```

```
result = division(10, 0)
print(result)
```

The program works as before because the `try...except` also catches the exception type that is the subclass of the `Exception` class.

However, if you pass two strings instead of two numbers to the `division()` function, you'll get the same message as if the `ZeroDivisionError` exception occurred:

```
def division(a, b):
    try:
        return {
            'success': True,
            'message': 'OK',
            'result': a / b
        }
    except Exception as e:
        return {
            'success': False,
            'message': 'b cannot be zero',
            'result': None
        }


result = division('10', '2')
print(result)
```

Output:

```
{'success': False, 'message': 'b cannot be zero', 'result': None}
```

In this example, the exception is not `ZeroDivisionError` but the `TypeError`. However, the code still handles it like the `ZeroDivisionError` exception.

Therefore, you should always handle the exceptions from the most specific to the least specific. For example:

```
def division(a, b):
    try:
        return {
            'success': True,
            'message': 'OK',
            'result': a / b
        }
    except TypeError as e:
        return {
            'success': False,
            'message': 'Both a & b must be numbers',
            'result': None
        }
    except ZeroDivisionError as e:
        return {
            'success': False,
            'message': 'b cannot be zero',
            'result': None
        }
    except Exception as e:
        return {
            'success': False,
            'message': str(e),
            'result': None
        }
```

```
result = division('10', '2')
print(result)
```

In this example, we catch the `TypeError`, `ZeroDivisionError`, and `Exception` in the order that they appear in the `try...except` statement.

If the code that handles different exceptions are the same, you can group all exceptions into one as follows:

```
def division(a, b):
    try:
        return {
            'success': True,
            'message': 'OK',
            'result': a / b
        }
    except (TypeError, ZeroDivisionError, Exception) as e:
        return {
            'success': False,
            'message': str(e),
            'result': None
        }


result = division(10, 0)
print(result)
```

Output:

```
{'success': False, 'message': 'division by zero', 'result': None}
```

Summary

- Python exceptions are objects of classes, which are the subclasses of the BaseException class.
- Do handle the exception from the most specific to lest specific.