Digital Career Institute

Basics - Exceptions

Digital Career Institute

Raising exceptions

raise

Digital Career Institute

Sometimes, an exception has to be thrown in code when a developer wants to write a program that would throw errors if it does not fulfill certain logic.

Python provides **`raise`** statement to **throw** exceptions in code.

The single arguments in the **`raise`** statement shows the exception that has to be raised.

We can raise exceptions in cases such as receiving wrong data or a validation failure.

Steps to `raise` exceptions

- Create an exception of the appropriate type.
- Use the existing built-in exceptions or create your own exception according to our requirement.
- Pass the appropriate data while raising an exception. Execute a raise statement, by providing the `Exception` class.

The syntax for `raise statements:`

```
raise  SomeExceptionClass(<value>)
```

Example

Let's look at a function that will throw an exception if the interest rate is greater than 100.

```
def simple_interest(amount, year, rate):
    if rate > 100:
        raise ValueError(
            f"Your rate is out of range {rate}."
        )
    else:
        interest = (amount * year * rate)/100
        print("Simple interest:", interest)
        return interest

simple_interest(800, 6, 800)
```

Output

```
ValueError: Your interest rate is out of range 800.
```

Exception classes: Custom exceptions

Exception Classes

Digital Career Institute

Sometimes we have to define and `raise` exceptions that are not provided natively by Python.

Such type of exceptions are called **user-defined exception** or **customized exception.**

The user can define custom exceptions by creating a new class. This new exception class has to derive either directly or indirectly from the built-in class `Exception`.

Example

exceptions.py

```
class UnderAgeError(Exception):
    """Raised when the age is below 18."""
    pass
```

The custom exception is defined by simply extending the `Exception` class.

payments.py

```
from exceptions import UnderAgeError

def pay(user, amount):
    if user.age < 18:
        raise UnderAgeError(
            "You must be 18 or older to make"
            " payments."
        )
    else:
        subtract_amount(user, amount)
```

Then, the custom exception can be raised whenever it is needed.

Example

transactions.py

```
from exceptions import UnderAgeError
from payments import pay

def buy(user, item):
    try:
        pay(user, item.price)
        ship(item, user.address)
    except UnderAgeError as error:
        log(error)
        user.call_parents()
        redirect_to("forbidden.html")
```

The custom exception can be catched by importing it and using it in the **`except`** clause.

Because the **`pay`** function raises an error when the user is younger than 18, no transaction will be able to charge any amount to an underage user.

Customizing Exception Classes

```
from exceptions import UnderAgeError
from payments import pay

def buy(user, item):
    try:
        pay(user, item.price)
        ship(item, user.address)
    except UnderAgeError as error:
        log(error)
        user.call_parents()
        redirect_to("forbidden.html")
```

We can **customize** the **classes** by accepting arguments as per our requirements.

Any custom exception class must be extending from **`BaseException`** class or subclass of **`BaseException`**.

The `BaseException` class is, as the name suggests, the base class for all built-in exceptions in Python.

Example

```
class NegativeAgeError(Exception):

        def __init__(self, age):
                self.message = f"Age should not
be negative. Age was {age}."
                super().__init__(self.message)


age = int(input("Enter age: "))
if age < 0:
        raise NegativeAgeError(age)
```

Output

```
Enter age: -28
 Traceback (most recent call last):
   File "/exception.py", line 11, in
     raise NegativeAgeError(age)
   __main__.NegativeAgeError: Age
  should not be negative. Age was -5.
```

At the core of the lesson

- Errors and exceptions allow programmers to debug their code
- Exceptions are logical errors, that can be thrown and caught using `try-except` blocks
- Code that must execute can be put in a `finally` block
- `Raise` statements allow developers to throw errors in their code, this could be in-built python exceptions or custom exceptions
- Developers can create their exceptions classes from the BaseException class to create custom exceptions

Built-in exceptions

Built-in exceptions

Python **automatically** generates many exceptions and errors with bulitin exceptions under the `BaseException` class.

Runtime exceptions, are generally a result of programming errors, such as:

- Reading a file that is not present
- Trying to read data outside the available index of a list
- Dividing an integer value by zero

Please have a look at the list of built-in exception classes on Python Docs:

https://docs.python.org/3/library/exceptions.html#bltin-exceptions

Important Built-in exceptions

AssertionError	Raised when an `assert` statement fails
AttributeError	Raised when attribute assignment or reference fails
EOFError	Raised when input() function hits the end-of-file condition
FloatingPointError	Raised when a floating-point operation fails
GeneratorExit	Raise when a generator's close() method is called
ImportError	Raised when the imported module is not found
IndexError	Raised when the index of a sequence is out of range
KeyError	Raised when a key is not found in a dictionary
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+C or delete)

Important Built-in exceptions

MemoryError	Raised when an operation runs out of memory
NameError	Raised when a variable is not found in the local or global scope
OSError	Raised when system operation causes system related error
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.

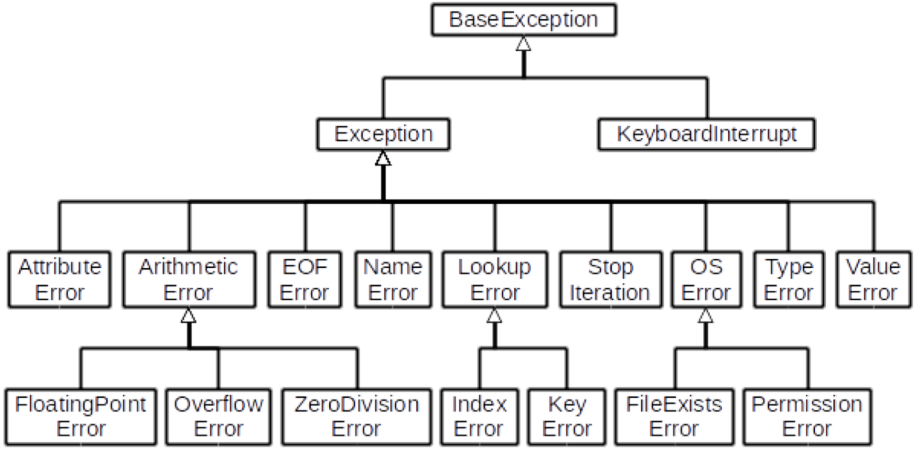
Exception class hierarchy

Python has **arranged** the built-in exception into a **class hierarchy** using **inheritance**.

Therefore, any class of exceptions used in an exception statement will also catch errors from its corresponding sub-class as well.

Check out the full list of exception classes and subclasses :
https://docs.python.org/3/library/exceptions.html#exception-hierarchy

Exception class hierarchy - visualisation



Source: exception_hierarchy.png

For Example, raising an exception of type LookUp Error will also raise errors of types:

- Index error
- Key error

You can check the class hierarchy of an exception in Python by using the following syntax:

```
<ExceptionClass>.mro()
```

This will return a list of the entire class tree. Try it out!

Example

Both the `lookups()` functions below should give you identical outputs since `IndexError` and `KeyError` are sub-classes of `LookupError`. Try it!

```
def lookups():
        s = [1,4,6]
        try:
                item = s[5]
        except IndexError:
                print("Handled
IndexError")
        d = dict(a=1, b=2)
        try:
                value = c["x"]
        except KeyError:
                print("Handled
KeyError")
lookups()
```

```
def lookups():
        s = [1,4,6]
        try:
                item = [5]
        except LookupError:
                print("Handled
IndexError")
        d = dict(a=1, b=2)
        try:
                value = c["x"]
        except LookupError:
                print("Handled
KeyError")
lookups()
```

At the core of the lesson

- There many in-built python exceptions, handling many different types of errors
- These built-in exceptions have sub-classes of “children” exceptions.
- The sub-classes of exceptions can be substituted with the parent class to have the exact same exceptions in a program

Documentation

Documentation

Digital Career Institute

1. Python.org documentation
2. W3Schools

Contact Details
DCI Digital Career Institute gGmbH

Digital Career Institute