Вбудовані виключення та помилки

Виключення (exceptions) - це об'єкти, які виникають при виконанні програми, коли виникає якась помилка або несправність. Виключення дозволяють визначити, що відбувається з програмою під час її виконання і як з цим пов'язано зупинення програми.

У Python існує безліч виключень, включаючи TypeError, ValueError, IndexError, KeyError та багато інших. Кожне виключення відображає певну помилку, яка сталася під час виконання програми.

Виключення (Exceptions): Виникають, коли виконання програми зустрічає непередбачувані ситуації, такі як ділення на нуль, відсутність файлів, невірно введені дані тощо.

Наприклад, якщо ми намагаємося викликати метод або атрибут, який не існує, то може виникнути AttributeError. Якщо ми намагаємося відкрити файл, який не існує, то може виникнути FileNotFoundError. Якщо ми намагаємося виконати ділення на нуль, то може виникнути ZeroDivisionError.

Нижче наведено кілька вбудованих виключень та їх пояснення:

1. **`SyntaxError`**
    - Виникає, коли знайдено синтаксичну помилку у програмі. Наприклад, неправильне використання ключових слів чи операторів.
2. **`IndentationError`**
    - Спричиняється, коли виявляється помилка відступів, що порушує структуру коду.
3. **`TypeError`**
    - Пов'язано з неправильним типом даних, переданим до функції чи операції.
4. **`NameError`**
    - Виникає, коли спроба використовувати ідентифікатор, який не було раніше визначено.
5. **`ValueError`**
    - Спричиняється, коли функція отримує аргумент правильного типу, але неприпустимого значення.
6. **`FileNotFoundError`**
    - Виникає, коли програма намагається відкрити файл, який не існує.

7. **`ZeroDivisionError`**
    - Спричиняється, коли відбувається спроба ділення на нуль.
8. **`IndexError`**
    - Виникає, коли програма намагається звертатися до елемента послідовності за межами індексів.
9. **`KeyError`**
    - Спричиняється, коли словник не містить вказаного ключа.
10. **`AttributeError`**
    - Виникає, коли об'єкт не має атрибута, з яким спробують взаємодіяти.
11. **`ImportError`**
    - Виникає, коли модуль не може бути імпортований.

try/except/else/finally

Для того, щоб обробити виключення, можна використовувати блоки try-except. Блок try містить код, який може викликати виключення, а блок except визначає, яке виключення потрібно обробити та як обробити помилку.

Наприклад:

```
try:
    x = int(input("Enter a number: "))
    y = 1 / x
except ValueError:
    print("Invalid input. Please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
else:
    print("Result is:", y)
finally:
    print(“its end”)
```

У цьому прикладі, ми спочатку намагаємося перетворити введене користувачем значення у ціле число.

Якщо введене число не дорівнює 0, то програма ділить 1 на це число та виводить результат.

Якщо введене значення не є числом, то виникає `ValueError`, і програма виводить повідомлення про неправильне введення.

Якщо користувач введе 0, то виникає `ZeroDivisionError`, і програма виводить повідомлення про неможливість ділення на 0.

Блок `else` виконується тільки тоді, коли жодне виключення не виникає

Блок `finally` виконується ЗАВЖДИ - незалежно від того сталося виключення чи ні. Практично це можна проілюструвати так: якщо у `try` трапилося непередбачене виключення - програма завершится без збереження данних. Тоді у нагоді може бути блок `finally,` де можна вставити код, що збереже дані.

Standard Exception Classes

lists exception classes raised by common runtime errors.

Exception class	Raised when
AssertionError	An **assert** statement failed.
AttributeError	An attribute reference or assignment failed.
ImportError	An **import** or **from**...**import** statement (covered in "The import Statement") couldn't find the module to import (in this case, what Python raises is actually an instance of ImportError's subclass ModuleNotFoundError), or couldn't find a name to be imported from the module.
IndentationError	The parser encountered a syntax error due to incorrect indentation. Subclasses SyntaxError.
IndexError	An integer used to index a sequence is out of range (using a noninteger as a sequence index raises TypeError). Subclasses LookupError.

KeyboardInterrupt	The user pressed the interrupt key combination (Ctrl-C, Ctrl-Break, Delete, or others, depending on the platform's handling of the keyboard).
KeyError	A key used to index a mapping is not in the mapping. Subclasses LookupError.
MemoryError	An operation ran out of memory.
NameError	A name was referenced, but it was not bound to any variable in the current scope.
NotImplementedError	Raised by abstract base classes to indicate that a concrete subclass must override a method.
OSError	Raised by functions in the module os (covered in "The os Module" and "Running Other Programs with the os Module") to indicate platform-dependent errors. OSError has many subclasses, covered in the following subsection.
RecursionError	Python detected that the recursion depth has been exceeded. Subclasses RuntimeError.
RuntimeError	Raised for any error or anomaly not otherwise classified.
SyntaxError	Python's parser encountered a syntax error.
SystemError	Python has detected an error in its own code, or in an extension module. Please report this to the maintainers of your Python version, or of the extension in question, including the error message, the exact Python version (sys.version), and, if possible, your program's source code.
TypeError	An operation or function was applied to an object of an inappropriate type.
UnboundLocalError	A reference was made to a local variable, but no value is currently bound to that local variable. Subclasses NameError.
UnicodeError	An error occurred while converting Unicode (i.e., a str) to a byte string, or vice versa. Subclasses ValueError.
ValueError	An operation or function was applied to an object that has a correct type but an inappropriate value, and nothing more specific (e.g., KeyError) applies.
ZeroDivisionError	A divisor (the righthand operand of a /, //, or % operator, or the second argument to the built-in function divmod) is 0. Subclasses ArithmeticError.

The hierarchy of built-in exception classes is:

```
BaseException
  Exception
    AssertionError, AttributeError, BufferError, EOFError,
    MemoryError, ReferenceError, OsError, StopAsyncIteration,
    StopIteration, SystemError, TypeError
    ArithmeticError (abstract)
      OverflowError, ZeroDivisionError
    ImportError
      ModuleNotFoundError, ZipImportError
    LookupError (abstract)
      IndexError, KeyError
    NameError
      UnboundLocalError
    OSError
      ...
    RuntimeError
      RecursionError
      NotImplementedError
    SyntaxError
      IndentationError
        TabError
    ValueError
      UnsupportedOperation
      UnicodeError
        UnicodeDecodeError, UnicodeEncodeError,
        UnicodeTranslateError
    Warning
      ...
  GeneratorExit
  KeyboardInterrupt
  SystemExit
```

Як створити виключення

В Python можна створити власні виключення, використовуючи вбудований клас `Exception`. Наприклад, щоб створити виключення, яке повідомляє про некоректний ввід, можна створити клас `InvalidInputException`, який буде успадковуватись від класу `Exception`:

```
class InvalidInputException(Exception):
```

```
    pass
```

У цьому прикладі ми визначили клас `InvalidInputException`, який успадковується від класу `Exception` і не має додаткових полів та методів. Можна додати додаткові параметри, які будуть доступні під час виникнення виключення, наприклад:

```
class InvalidInputException(Exception):
    def __init__(self, input):
        self.input = input

    def __str__(self):
        return f"Invalid input: {self.input}"
```

У цьому прикладі ми додали параметр `input` до конструктора класу та перевизначили метод `__str__()`, щоб повертати рядок, який містить інформацію про некоректний ввід.

Після створення виключення його можна використовувати, наприклад, у функціях для перевірки вхідних параметрів:

```
def calculate_sum(a, b):
    if not isinstance(a, int) or not isinstance(b, int):
        raise InvalidInputException(f"{a} or {b}")
    return a + b
```

У цьому прикладі функція `calculate_sum()` перевіряє, чи є вхідні параметри `a` та `b` цілими числами, і, якщо це не так, викликає виключення `InvalidInputException` з параметром, який містить інформацію про некоректний ввід.

При виконанні програми, якщо виключення буде виникати, то програма буде зупинена, і в консолі буде виведено інформацію про виключення, включаючи назву виключення та параметри, які були передані.

Тестування виключень

Тестування виключень - це процес перевірки, що виключення генеруються правильно і обробляються в коді. В Python для тестування виключень можна використовувати модуль unittest.

Модуль unittest має вбудовані методи для перевірки виключень, зокрема метод assertRaises(), який можна використовувати для перевірки, що певне виключення буде згенероване при виклику функції.

Наприклад, якщо ми хочемо перевірити, що при діленні на нуль генерується виключення ZeroDivisionError, то ми можемо написати наступний тестовий випадок:

```
import unittest

def divide(x, y):
    if y == 0:
        raise ZeroDivisionError("division by zero")
    return x / y

class TestDivide(unittest.TestCase):
    def test_divide_by_zero(self):
        self.assertRaises(ZeroDivisionError, divide, 10, 0)

if __name__ == '__main__':
    unittest.main()
```

У цьому прикладі ми визначили функцію divide, яка робить ділення, та написали тестовий випадок, який перевіряє, що при діленні на нуль генерується виключення ZeroDivisionError.

Метод assertRaises() очікує, що під час виклику функції divide з аргументами 10 і 0 буде згенеровано виключення ZeroDivisionError.

Модуль unittest також має інші методи для тестування виключень, наприклад assertRaisesRegex(), який дозволяє перевіряти, що виключення містить певний текстовий рядок. Для детальнішої інформації про тестування виключень в Python можна ознайомитися з документацією модуля unittest.