

Exception, Logging

Exception

- Виняткові ситуації або виключення (exceptions) - це помилки, виявлені під час виконання. Наприклад, до чого призведе спроба поділити на 0 чи читання неіснуючого файлу? Або якщо файл був випадково видалений поки програма працювала? Такі ситуації обробляються за допомогою винятків.
- Якщо ж Python не може зрозуміти, як обійти ситуацію, що склалася, то йому не залишається нічого окрім як підняти руки і повідомити, що виявив помилку.
- Загалом, винятки необхідні, щоб повідомляти програмісту про помилки.

- Розглянемо найпростіший приклад виключення - ділення на нуль:

Traceback (most recent call last):

File "", line 1, in

100 / 0

ZeroDivisionError: division by zero

- В даному випадку інтерпретатор повідомив нам про виключення ZeroDivisionError - ділення на нуль.

Traceback

- У великій програмі виключення часто виникають всередині. Щоб спростити програмісту розуміння помилки і причини такої поведінки Python пропонує Traceback або на сленгу – «Трейс». Кожне виключення містить коротку інформацію, але при цьому повну, інформацію про місце появи помилки. За Трейсом знайти і виправити помилку стає простіше.

python error messages: C++ error messages:



Приклад трейсу

Traceback (most recent call last):

```
File "/home/username/Develop/test/app.py", line 862, in _handle
    return route.call(**args)
```

```
File "/home/username/Develop/test/app.py", line 1729, in wrapper
    rv = callback(*a, **ka)
```

```
File "/home/username/Develop/test/__init__.py", line 76, in wrapper
    body = callback(*args, **kwargs)
```

```
File "/home/username/Develop/test/my_app.py", line 16, in index
    raise Exception('test exception')
```

- В даному прикладі чітко видно, який шлях виконання у програми. Дивимося знизу до верху і по кроках розуміємо, як же ми "докотилися" до такого виключення.

Ще приклад

Traceback (most recent call last):

File

"C:\Users\Misha\PycharmProjects\pythonProject2\main.py", line 1, in <module>

a = 2 + '1'

TypeError: unsupported operand type(s) for +: 'int' and 'str'

- В даному прикладі при спробі скласти ціле число і рядок ми отримуємо виняток `TypeError`. В описі відразу ж стає ясно, що ж ми не так написали.

І ще приклад

Traceback (most recent call last):

```
File  
"C:\Users\Misha\PycharmProjects\pythonProject2\main.py",  
line 1, in <module>
```

```
    a = int('qwerty')
```

```
ValueError: invalid literal for int() with base 10:  
'qwerty'
```

- Приведення рядку до цілого числа призводить до виключення ValueError.
- У Трейсі цих двох прикладів можна прочитати, що в такому-то файлі на такий-то рядку є помилки.

Останній приклад (чесне слово)

Traceback (most recent call last):

File "C:\Users\Misha\PycharmProjects\pythonProject2\main.py", line 4, in
<module>

func()

File "C:\Users\Misha\PycharmProjects\pythonProject2\main.py", line 2, in func
func()

File "C:\Users\Misha\PycharmProjects\pythonProject2\main.py", line 2, in func
func()

File "C:\Users\Misha\PycharmProjects\pythonProject2\main.py", line 2, in func
func()

[Previous line repeated 996 more times]

RecursionError: maximum recursion depth exceeded

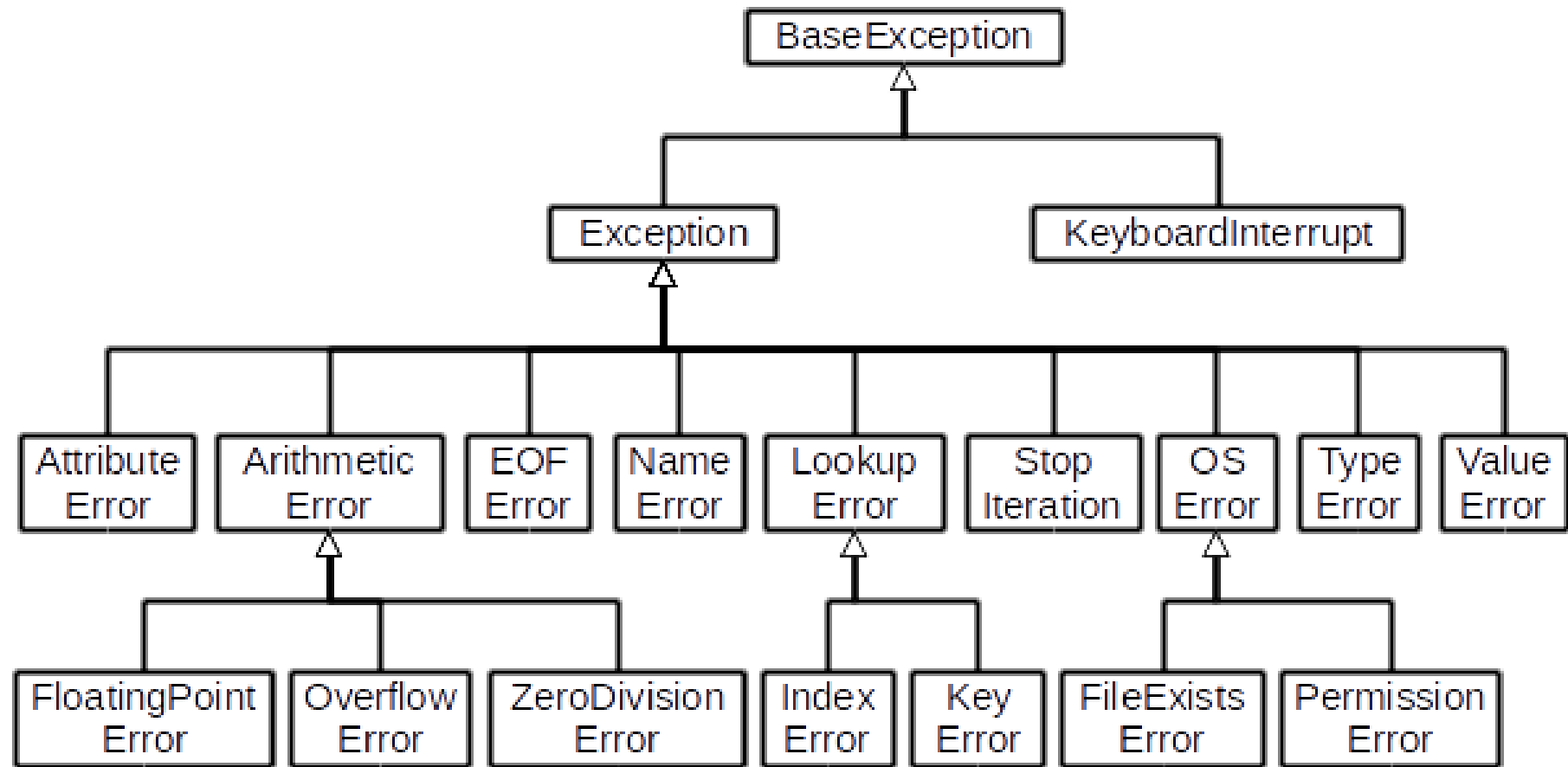
Ієрархія виключень

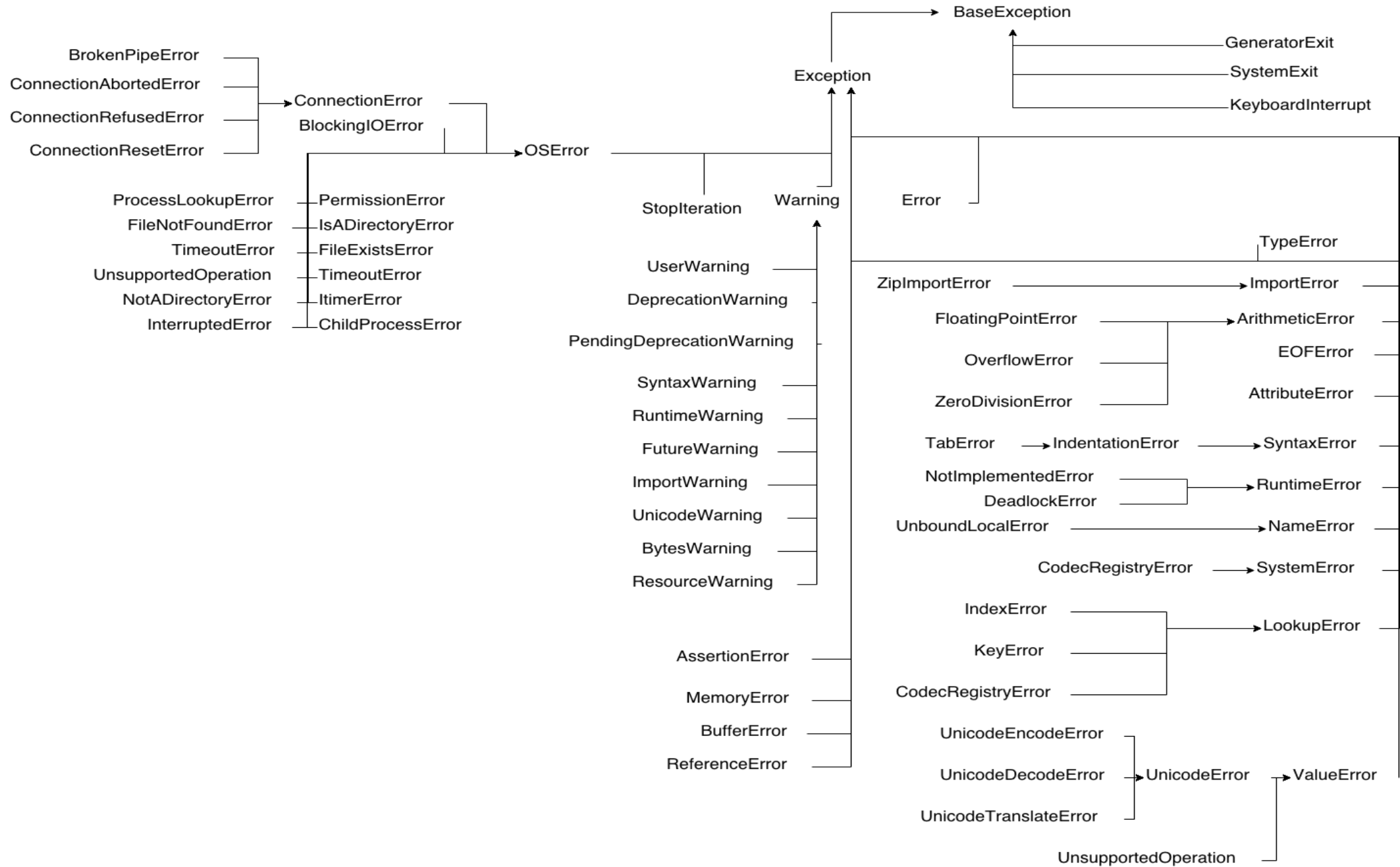
- Виключення, яке ви не побачите при виконанні коду - це `BaseException` - базове виключення, від якого беруть початок інші.
- В ієрархії виключень дві основні групи:
 1. Системні виключення і помилки
 2. Звичайні виключення
- Якщо обробку перших краще не робити (якщо і робити, то треба чітко розуміти для чого), то обробку других цілком і повністю Python покладає на плечі програміста.

До системних можна сміливо віднести:

- `SystemExit` - виняток, що породжується функцією `sys.exit` при виході з програми.
- `KeyboardInterrupt` - виникає при перериванні програми користувачем (зазвичай сполучення клавіш `Ctrl + C`).
- `GeneratorExit` - виникає при виклику методу `close` об'єкта `generator`.

Решта виключення це "звичайні". Спектр вже готових винятків великий.





Інструкція raise

- Інструкція raise використовується для того щоб "кинути" виключення з певної ділянки коду
- `raise TypeError`
- `raise StopIteration`

Власні виключення

- При написанні власних програм розумне бажання додати виразності коду і так само звернути інших програмістів на особливі виняткові ситуації. Для вирішення цього завдання варто використовувати власні виключення.
- У мінімальному виконанні необхідно успадковуватися від якого-небудь класу в ієрархії виключень. Наприклад так:

```
class MyException(Exception):  
    pass
```

- Легко помітити, ми створюємо клас, а значить все, що ми знаємо про класи справедливо і для винятків. Можна завести змінні і робити їх обробку. Як правило, виключення це дуже маленькі класи. Вони повинні виконуватися максимально швидко.

Приклад

```
class MyError(Exception):  
    def __init__(self, text):  
        self.txt = text
```

```
a = input("Input positive integer: ")
```

Покращений варіант

```
class MyError(Exception):  
    def __init__(self, text, num):  
        self.txt = text  
        self.n = num  
  
a = input("Input positive integer: ")
```

І що з цим всім робити? Ловити і
опрацьовувати!



Try... except...

Конструкція Try except потрібна для захоплення і опрацювання помилок в потенційних місцях виникнення

В блоці Try пишемо код який повинен виконатись. В блоці except захоплюємо помилку яка може виникнути при виконанні інструкцій в блоці Try

```
try:
```

```
    a = 100
```

```
    b = 0
```

```
    c = a / b
```

```
except ZeroDivisionError:
```

```
    c = "inf"
```

- Нехай у нас файл з даними в файловій системі і необхідно його прочитати. У цьому випадку відразу ж спливають кілька виняткових ситуацій, такі як: немає файлу, файл битий; файл порожній (за завданням ми знаємо, що в ньому дані) та інші.

Використання декількох блоків except

- Використовуючи виключення можна таким чином вирішити дане завдання

try:

```
    filepath = 'test_file.txt'
    with open(filepath, 'r') as fio:
        result = fio.readlines()
    if not result:
        raise Exception("File is empty")
```

except IOError:

```
    result = []
```

except Exception:

```
    result = []
```

```
    print(e)
```

Повна форма конструкції Try except

```
try:
```

```
    print("виконуємо щось")
```

```
except Exception:
```

```
    print("опрацювання виключення")
```

```
else:
```

```
    print("код який спрацює якщо виключення не було  
кинуто")
```

```
finally:
```

```
    print("код який гарнатовано спрацює в кінці в будь-  
якому випадку")
```

Логування

- Список (лог-файли) - це файли, що містять системну інформацію роботи сервера або комп'ютера, в які заносяться певні дії користувача або програми.
- Їх призначення - протоколювання операцій, що виконуються на машині, для подальшого аналізу адміністратором. Регулярний перегляд журналів дозволить визначити помилки в роботі системи в цілому, конкретного сервісу або сайту (особливо приховані помилки, які не виводяться при перегляді в браузері), діагностувати зловмисну активність, зібрати статистику відвідувань сайту.

Логування в Python

- Логування можна здійснювати будь якими методами, це може бути звичайний принт, або запис в певний лог-файл з подавленням виключень для виводу їх в логи. Для логування може бути написаний окремий модуль або використаний уже існуючий

Модуль Logging

- Модуль logging в Python - це готовий до використання, потужний модуль. Він використовується більшістю сторонніх бібліотек Python, тому ви можете інтегрувати ваші логи з повідомленнями з цих бібліотек для створення єдиного журналу логів в вашого додатку.
- Він відразу "в коробці" з Python тому для його підключення потрібно тільки імпортувати.

```
>>> import logging
```

Ступені «важливості»

- За замовчуванням існує 5 стандартних рівнів severity, що вказують на важливість подій. У кожного є відповідний метод, який можна використовувати для логування подій на обраному рівні severity. Список рівнів в порядку збільшення важливості:

1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

Приклад

```
import logging
```

```
logging.debug('This is a debug message')
```

```
logging.info('This is an info message')
```

```
logging.warning('This is a warning message')
```

```
logging.error('This is an error message')
```

```
logging.critical('This is a critical message')
```

Вивід з попереднього прикладу

```
WARNING:root:This is a warning message
```

```
ERROR:root:This is an error message
```

```
CRITICAL:root:This is a critical message
```

Зверніть увагу, що повідомлення `debug ()` та `info ()` не були відображені. Це пов'язано з тим, що за замовчуванням модуль реєструє повідомлення лише з рівнями `WARNING` або вище.

- Вивід показує рівень важливості перед кожним повідомленням разом з root, який модуль журналу дає своє логеру за замовчуванням. Цей формат, який показує рівень, ім'я та повідомлення, розділеними двокрапкою (:), є форматом виводу за замовчуванням, і його можна змінити для включення таких речей, як мітка часу, номер строки та інші деталі.

Метод basicConfig

- Часто використовувані параметри для basicConfig ():
- level : Корневий логер із встановленим вказівним рівнем важливості
- filename : файла логів
- filemode: Режим відкриття файлу. За умовою це «a»
- format : Формат повідомлень.

```
Template_format = ' %(asctime)s - %(name)s - %(levelname)s -  
%(message)s '
```

```
logging.basicConfig(level=logging.DEBUG, filename='app.log',  
filemode='w', format= Template_format)
```

format

- <https://docs.python.org/3/library/logging.html#logrecord-attributes>
- За посиланням вище можна знайти різні атрибути для формату повідомлень логера

Логування змінних

- У більшості випадків вам потрібно буде включати динамічну інформацію з вашого застосування в журнали. Ви бачили, що методи ведення журналу приймають рядок в якості аргументу, і може здатися нормальним відформатувати рядок зі змінними в окремому рядку і передати її методу `log`. Але насправді це можна зробити безпосередньо, використовуючи рядок формату для повідомлення і додаючи змінні в якості аргументів.

```
name = "Misha"
```

```
logging.info(f'This is an info message {name}')
```

Логування Трейсу

- Модуль реєстрації також дозволяє вам захоплювати стек виконання. Інформація про виключення може бути отримана, якщо параметр `exc_info` переданий як `True`, а функції ведення журналу викликаються таким чином:

```
import logging
a = 5
b = 0
try:
    c = a / b
except Exception as e:
    logging.error("Exception occurred", exc_info=True)
```

- Тоді результатом логування буде

```
2021-03-15 12:27:50,856 - root - ERROR - Exception occurred
```

```
Traceback (most recent call last):
```

```
  File "C:\Users\Misha\PycharmProjects\pythonProject2\main.py", line 8, in  
<module>
```

```
    c = a / b
```

```
ZeroDivisionError: division by zero
```

Чому важливо логувати помилки

- Якщо для `exc_info` не задано значення `True`, вихідні дані вищенаведеної програми не повідомлять нам нічого про виключення, яке в реальному сценарії може бути не таким простим, як `ZeroDivisionError`. Уявіть, що ви намагаєтеся налагодити помилку в складній кодовій базі за допомогою журналу, який показує тільки це:

```
2021-03-15 12:27:50,856 - root - ERROR - Exception
occurred
```