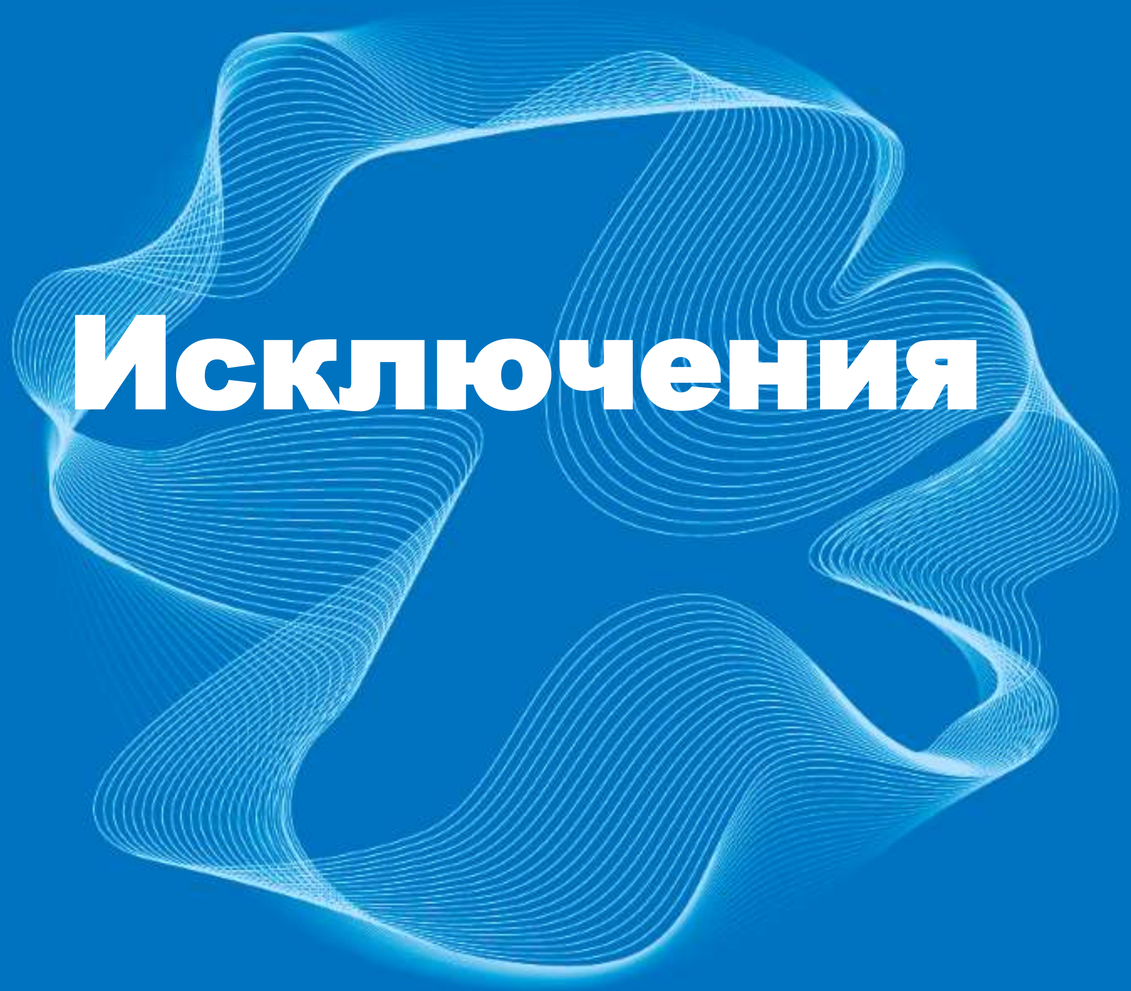
МФТИ



Исключения

Докладчик: Евграфов Михаил

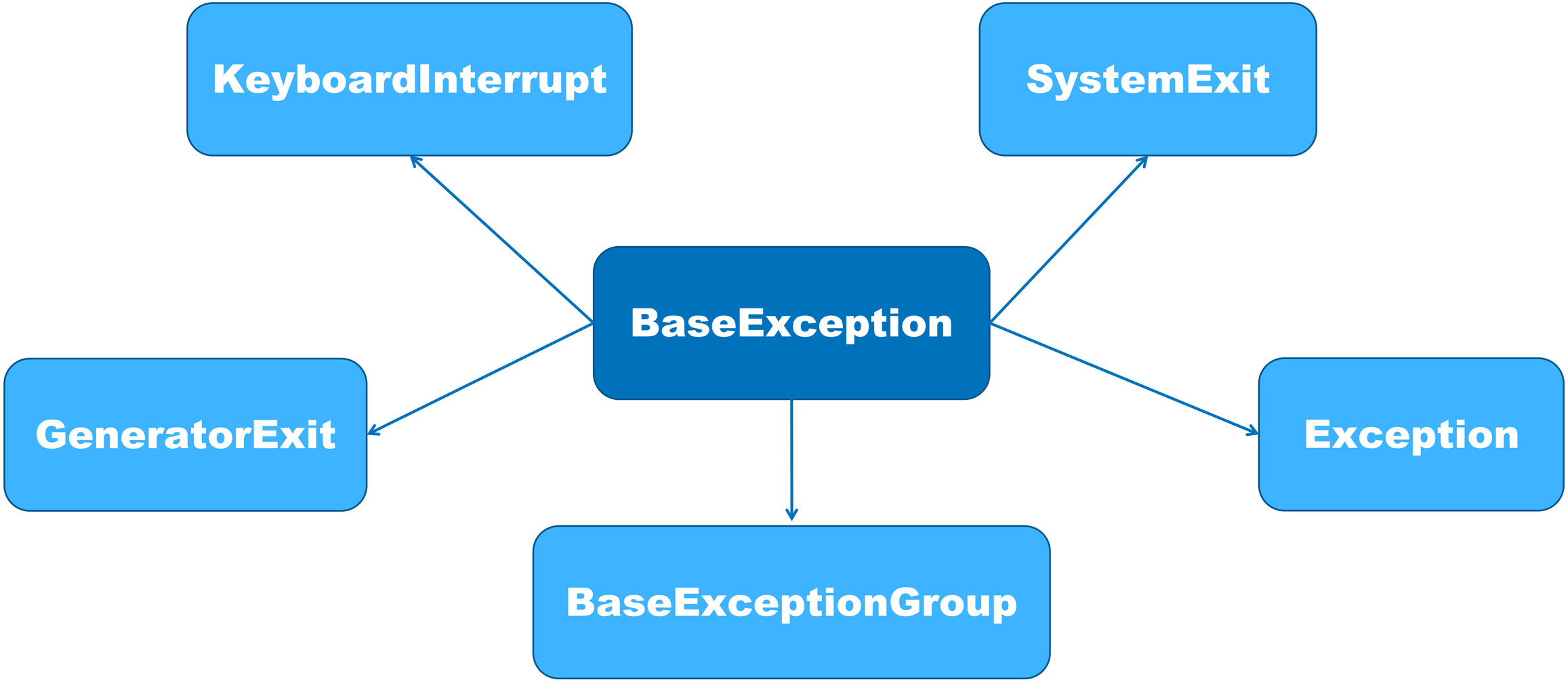
Исключения

Примеры исключений

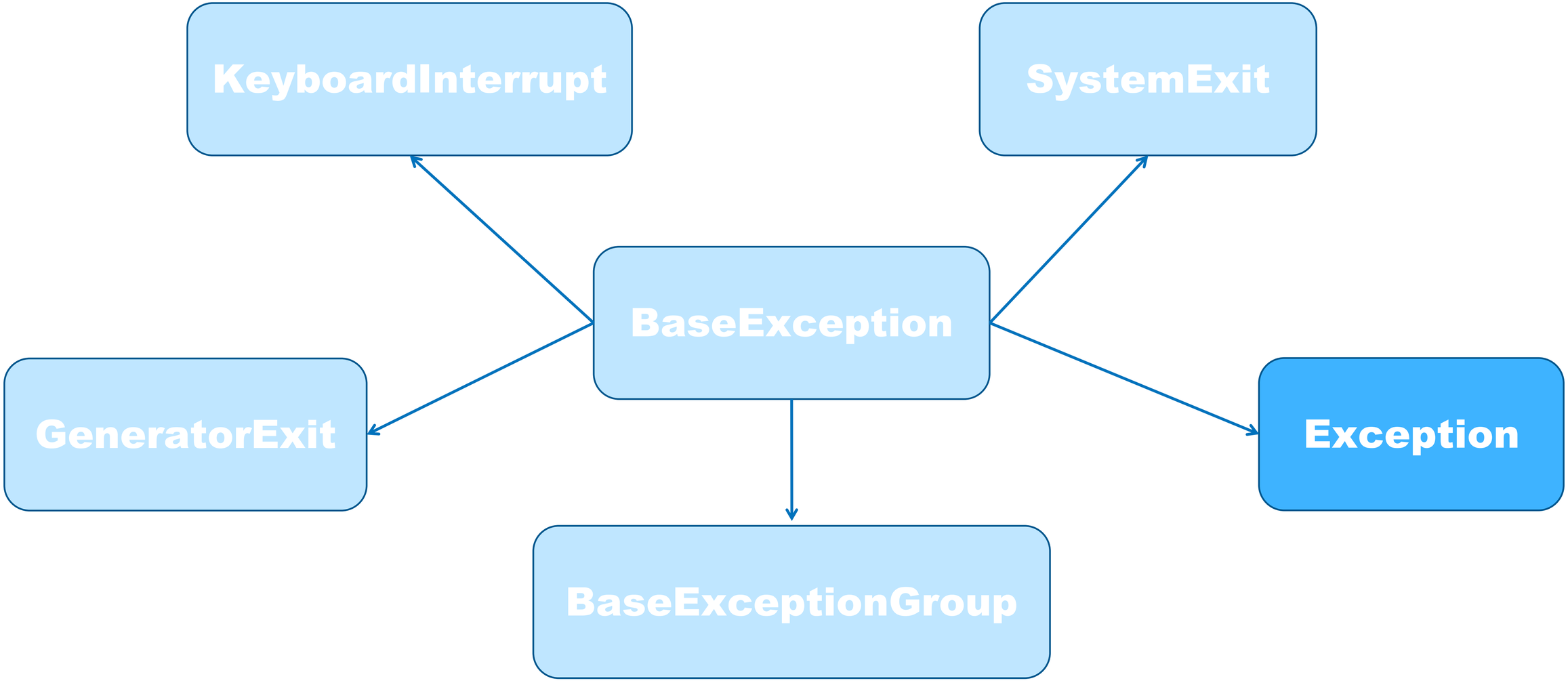
```
>>> # пример синтаксической ошибки
>>> num = 5
>>> if num % 2 == 1 print("number is odd")
...
SyntaxError: invalid syntax

>>> # пример логической ошибки
>>> division = num / 0
...
ZeroDivisionError: division by zero
```

Иерархия исключений



Иерархия исключений



try-except

```
while True:
    try:
        ...
        ...

    except ValueError:
        ...
        ...
```

try-блок

except-блок /
exception-handler

try-except: пример

```
# input -> "string"
while True:
    try:
        input_value = input("Enter a number: ")
        number = int(input_value)  # ValueError
        print(f"{number = }")      # не выполнится
        break                      # не выполнится
    except ValueError:
        # выполнится
        print(f"input is not valid: {input_value}")
# выполнится как только выйдем из цикла
print("successfully read number")
```

try-except: пример

```
# input -> 42
while True:
    try:
        input_value = input("Enter a number: ")
        number = int(input_value)  # OK
        print(f"{number = }")      # выполнится
        break                      # выполнится
    except ValueError:
        # не выполнится
        print(f"input is not valid: {input_value}")
# выполнится
print("successfully read number")
```

Непойманные исключения

```
# input -> 5
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # OK
    result = 10 / number       # OK
    print(f"{result = }")
except ValueError:
    print(f"input is not valid: {input_value}")

result = 2.0
```

Непойманные исключения

```
# input -> "five"
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # FAIL
    result = 10 / number       # не будет выполнено
    print(f"{result = }")
except ValueError:
    print(f"input is not valid: {input_value}")

input is not valid: five
```

Непойманные исключения

```
# input -> 0
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # OK
    result = 10 / number       # FAIL
    print(f"{result = }")
except ValueError:
    print(f"input is not valid: {input_value}")


...
ZeroDivisionError: division by zero
```

Общий обработчик

```
# input -> "five"
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # FAIL
    result = 10 / number       # не будет выполнено
    print(f"{result = }")
except (ValueError, ZeroDivisionError):
    print(f"input is not valid: {input_value}")

input is not valid: five
```

Общий обработчик

```
# input -> 0
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # OK
    result = 10 / number       # FAIL
    print(f"{result = }")
except (ValueError, ZeroDivisionError):
    print(f"input is not valid: {input_value}")

input is not valid: 0
```

Несколько обработчиков

```
# input -> "five"
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # FAIL
    result = 10 / number       # не будет выполнено
    print(f"{result = }")
except ValueError:
    print(f"input is not valid: {input_value}")
except ZeroDivisionError:
    print("0 as input is forbidden")

input is not valid: five
```

Несколько обработчиков

```
# input -> 0
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # OK
    result = 10 / number       # FAIL
    print(f"{result = }")
except ValueError:
    print(f"input is not valid: {input_value}")
except ZeroDivisionError:
    print("0 as input is forbidden")
```

```
0 as input is forbidden
```

Напоминание

```
>>> print(ValueError.__mro__)
(
    <class 'ValueError'>,
    <class 'Exception'>,
    <class 'BaseException'>,
    <class 'object'>,
)
```

Порядок выбора обработчика

```
# input -> "five"
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # FAIL
    result = 10 / number       # не будет выполнено
    print(f"{result = }")

except ValueError:  # используемый обработчик
    print(f"input is not valid: {input_value}")

except Exception as exception:
    print(f"general handler catch
{type(exception).__name__}")

input is not valid: five
```

Порядок выбора обработчика

```
# input -> 0
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # OK
    result = 10 / number       # FAIL
    print(f"{result = }")

except ValueError:
    print(f"input is not valid: {input_value}")

except Exception as exception: # используемый обработчик
    print(f"general handler catch
{type(exception).__name__}")
```

```
general handler catch ZeroDivisionError
```

Неверный порядок обработчиков

```
# input -> "five"
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # FAIL
    result = 10 / number       # не будет выполнено
    print(f"{result = }")

except Exception as exception: # используемый обработчик
    print(f"general handler catch
{type(exception).__name__}")

except ValueError:
    print(f"input is not valid: {input_value}")
```

```
general handler catch ValueError
```

try-except-else

```
# input -> "five"
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # FAIL

except ValueError:
    print(f"input is not valid:
{input_value}")

else:
    print(f"{number = }")

input is not valid: five
```

try-except-else

```
# input -> 42
try:
    input_value = input("Enter a number: ")
    number = int(input_value)  # OK

except ValueError:
    print(f"input is not valid:
{input_value}")

else:
    print(f"{number = }")

number = 42
```

raise

```
# возбуждение с явным инстанцированием
>>> raise ValueError("invalid value")
...
ValueError: invalid value

# возбуждение без явного инстанцирования
>>> raise ValueError
...
ValueError:
```

re-raise

```
try:
    raise ValueError("invalid value")

except ValueError as exception:
    print(f"exception info: {exception}")
    raise

exception info: invalid value
ValueError: invalid value
```

Цепочка исключений

```
def connect_to_db() -> None:
    raise ConnectionError("fail to connect")

try:
    connect_to_db()

except ConnectionError:
    raise RuntimeError("transaction failed")
```

```
...
ConnectionError: fail to connect
...
RuntimeError: transaction failed
```

raise from

```
def connect_to_db() -> None:
    raise ConnectionError("fail to connect")

try:
    connect_to_db()

except ConnectionError as exc:
    raise RuntimeError("transaction failed") from exc
```

```
...
ConnectionError: fail to connect
...
RuntimeError: transaction failed
```

raise from None

```
def connect_to_db() -> None:
    raise ConnectionError("fail to connect")

try:
    connect_to_db()

except ConnectionError:
    raise RuntimeError("transaction failed") from None

...
RuntimeError: transaction failed
```

Пользовательские исключения

```
class MyException(Exception):
    pass

try:
    raise MyException

except Exception as exc:
    print(f"catch exception: {type(exc).__name__}")
```

```
catch exception: MyException
```

Механизм распространения ошибок

```
def raise_() -> None:
    print("before raise")
    raise Exception("exc from raise_")
    print("after raise")

def call_raise() -> None:
    print("before calling raise_")
    raise_()
    print("after calling raise_")
```

Механизм распространения ошибок

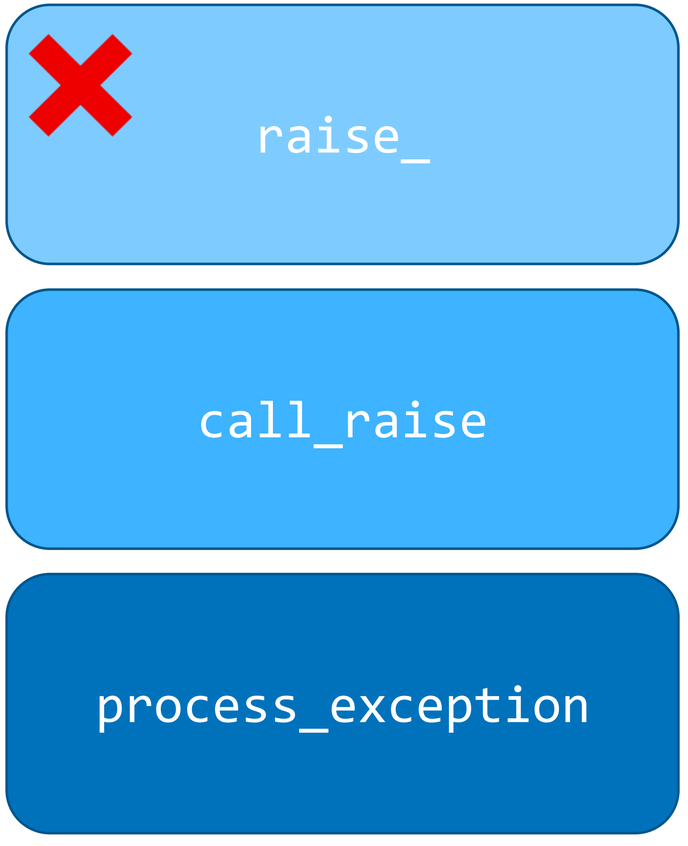
```
def process_exception() -> None:
    try:
        print("before exception raising")
        call_raise()
        print("after exception raising")

    except Exception as exc:
        print(f"process exception:
{exc}")
```

Механизм распространения ошибок

```
>>> process_exception()
before exception raising
before calling raise_
before raise
>>> raise Exception(...)
```

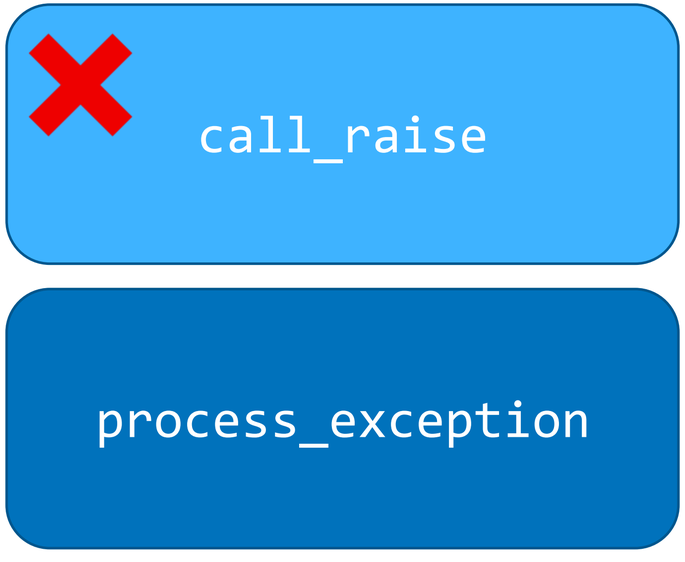
Стек вызовов



Механизм распространения ошибок

```
>>> process_exception()
before exception raising
before calling raise_
before raise
>>> raise Exception(...)
```

Стек вызовов



Механизм распространения ошибок

```
>>> process_exception()
before exception raising
before calling raise_
before raise
>>> raise Exception(...)
process exception: exc from raise_
```

Стек вызовов

process_exception

Контекстные менеджеры

Захват и освобождение ресурсов

```
>>> # захват ресурса
>>> file = open("file.txt", mode="w")
>>> file.write("Hello, World!")

>>> # освобождение ресурса
>>> file.close()
```

Проблема

```
>>> # захват ресурса
>>> file = open("file.txt", mode="w")
>>> file.write("Hello, World!")  # FAIL

>>> # освобождение ресурса не произойдет
>>> file.close()
```

try-finally

```
try:
    file = open("file.txt", mode="w")
    file.write("Hello, World!")  # OK
    print("succesfully write to file")

finally:
    file.close()
    print("successfully close file")
```

```
succesfully write to file
successfully close file
```

try-finally

```
try:
    file = open("file.txt", mode="r")
    file.write("Hello, World!")  # FAIL
    print("succesfully write to file")

finally:
    file.close()
    print("successfully close file")

successfully close file
...
UnsupportedOperation: not writable
```

try-except-finally

```
try:
    file = open("file.txt", mode="r")
    file.write("Hello, World!")
    print("succesfully write to file")

except Exception as exc:
    print(exc)

finally:
    file.close()
    print("successfully close file")
```

```
not writable
successfully close file
```

try-except-else-finally

```
try:
    # выполняется всегда
    pass
except Exception:
    # выполняется при ошибке в try-блоке
    pass
else:
    # выполняется, если в try-блоке нет ошибок
    pass
finally:
    # выполняется всегда
    pass
```

with

```
# файл гарантированно будет закрыт
# что бы ни произошло
with open("test.txt", "w") as file:
    file.write("Hello, World!")
```

Множественный with

```
with (
    open("source.txt", "r") as file_read,
    open("sink.txt", "w") as file_write,
):
    readen_data = file_read.read()
    file_write.write(readen_data)
```

Протокол контекстного менеджера

```
class MyContextManager:
    def __enter__(self) -> None:
        print("call __enter__")

    def __exit__(self, exc_type, exc_value, exc_tb) -> None:
        print("call __exit__")

>>> with MyContextManager():
...     pass

call __enter__
call __exit__
```

Протокол контекстного менеджера

```
class MyContextManager:
    def __enter__(self) -> None:
        print("call __enter__")

    def __exit__(self, exc_type, exc_value, exc_tb) -> None:
        print("call __exit__")

>>> with MyContextManager():
...     raise Exception

call __enter__
call __exit__
...
Exception:
```

Значения из __enter__

```
class MyContextManager:
    def __enter__(self) -> str:
        print("call __enter__")
        return "value from __enter__"

    def __exit__(self, exc_type, exc_value, exc_tb) -> None:
        print("call __exit__")

>>> with MyContextManager() as enter_value:
...     print(enter_value)

call __enter__
value from __enter__
call __exit__
```

Обработка исключений в __exit__

```
class MyContextManager:
    def __enter__(self) -> None:
        print("call __enter__")

    def __exit__(self, exc_type, exc_value, exc_tb) -> bool:
        print("call __exit__")
        if isinstance(exc_value, ValueError):
            return True

>>> with MyContextManager():
...     raise ValueError

call __enter__
call __exit__
```

Семинар