IT CAREER HUB

Урок 25.

Исключения: обработка ошибок

Исключения

Исключения (exceptions) – это события, возникающие во время выполнения программы, которые сигнализируют об ошибочной ситуации, но могут быть обработаны, чтобы программа продолжила выполнение.

Примеры исключений

Исключение	**Причина возникновения**	**Пример возникновения**
`ZeroDivisionError`	деление на ноль	`print(10 / 0)`
`ValueError`	ошибка значения	`int("abc")`
`KeyError`	обращение к несуществующему ключу словаря	`info = {"a": 1}` `print(info["b"])`

Зачем нужна обработка исключений?

- **Повышение стабильности программы:** Позволяет программе продолжать работу после возникновения ошибки.
- **Улучшение пользовательского опыта:** Предоставляет понятные сообщения об ошибках вместо стандартного сообщения от Python.
- **Отладка:** Помогает обнаруживать и исправлять ошибки в коде на ранних этапах разработки.

Как Python реагирует на ошибки?

Когда в программе возникает ошибка:

1. Python создаёт объект исключения, который содержит:
    - Тип исключения (например, `ValueError`, `TypeError`)
    - Сообщение об ошибке, описывающее проблему
    - Трассировку (`traceback`), указывающую, где ошибка произошла
2. Если исключение не обработано, программа завершает выполнение, выводя трейсбэк (информацию об ошибке).

Пример

```python
print(10 / 0)  # Ошибка: ZeroDivisionError
```

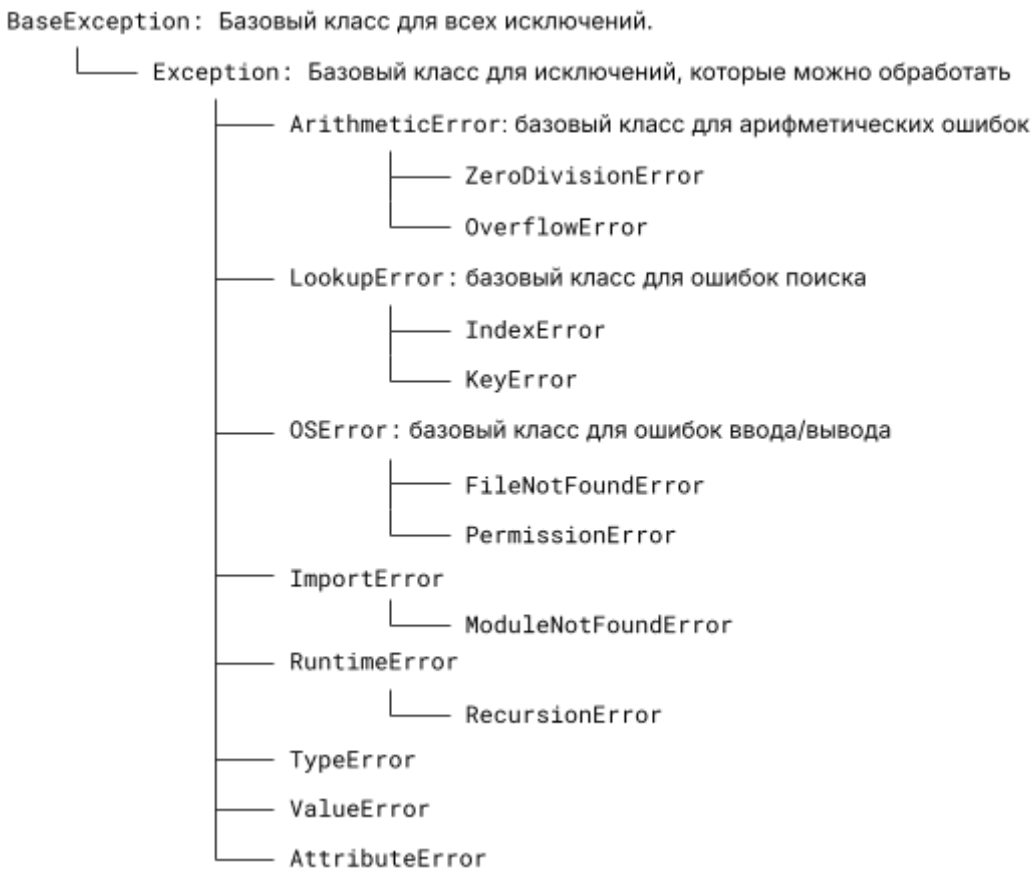
```
Traceback (most recent call last):
  File "/home/tanya/PycharmProjects/pythonProgramItch/_notes/test.py", line 1, in <module>
    print(10 / 0)
          ~~~^~~
ZeroDivisionError: division by zero

Process finished with exit code 1
```

Иерархия исключений

В Python все исключения происходят от базового класса `BaseException`, однако для обработки ошибок используется иерархия, начинающаяся с `Exception`.

Для изучения всех встроенных исключений и их иерархии в Python можно обратиться к Официальной документации Python: Исключения

Основные ветви иерархии:



Также есть ветви, которые наследуются от базового класса `BaseException`, но не обрабатываются как обычные исключения: `SystemExit`, `KeyboardInterrupt` и `GeneratorExit`.

IT CAREER HUB

Обработка исключений

Обработка исключений позволяет перехватывать ошибки, возникающие во время выполнения программы, чтобы предотвратить её аварийное завершение. Для этого используется конструкция `try-except`.

IT CAREER HUB

Конструкция try-except

`try-except` – это базовый механизм обработки исключений, который позволяет перехватывать ошибки во время выполнения программы и предотвращать её аварийное завершение.

Синтаксис

```python
try:
    # Код, который может вызвать исключение
except ExceptionType:
    # Код, который выполнится в случае исключения
```

`ExceptionType` – это тип исключения, который вы хотите обработать (например, `ZeroDivisionError`, `ValueError` и т.д.).

Пример

```python
try:
    result = 10 / 0  # Возникает ZeroDivisionError
    # Следующий код не достижим при ошибке
    print("Деление выполнено успешно!")
    print(result)
except ZeroDivisionError:
    print("Ошибка: деление на ноль!")
```

Как Python выбирает обработчик исключений?

1. Python проверяет обработчики except сверху вниз. Когда возникает исключение, Python идёт по списку обработчиков и выбирает первый подходящий. Первый совпавший блок except выполняется, а остальные игнорируются.

Python

```
try:
    result = 10 / 0  # Возникает ZeroDivisionError
except ValueError:
    print("Некорректное значение!")  # Пропускается, так как ошибка не ValueError
except ZeroDivisionError:
    print("Ошибка деления на ноль!")  # Этот блок выполнится
```

2. Если нет подходящего обработчика, программа завершает работу с ошибкой.

Python

```
try:
    result = 10 / 0  # Возникает ZeroDivisionError
except IndexError:
    print("Индекс вне диапазона!")  # Пропускается
except KeyError:
    print("Ключ не существует!")  # Пропускается
```

3. Общий обработчик `Exception` перехватывает все типы исключений, если они не были обработаны ранее.

Python

```
try:
    result = 10 / 0  # Возникает ZeroDivisionError
except IndexError:
    print("Индекс вне диапазона!")  # Пропускается
except KeyError:
    print("Ключ не существует!")  # Пропускается
except Exception:
    print("Ошибка обработки!")  # Выполнится, так как `Exception` перехватывает `ZeroDivisionError`
```

4. `Exception` нужно ставить в конце.
    - Если `except Exception` будет раньше, он перехватит все исключения, не дав сработать более специфичным обработчикам.

Python

```
try:
    result = 10 / 0
except Exception:  # Перехватывает всё, обработчики ниже не сработают
    print("Ошибка обработки!")
except ZeroDivisionError:
    print("Ошибка деления на ноль!")  # Этот код недостижим
```

IT CAREER HUB

Обработка нескольких исключений в одном блоке

Python позволяет обрабатывать несколько типов исключений в одном блоке `except`, используя кортеж из типов исключений.

Синтаксис

```python
try:
    # Код, который может вызвать исключение
except (ExceptionType1, ExceptionType2):
    # Код, который выполнится в случае любого из указанных исключений
```

Пример

```python
while True:
    try:
        user_input = input("Введите ненулевое число: ")
        result = 10 / int(user_input)
        print(f"Результат: {result}")
        break  # Выход из цикла, если ошибок не было
    except (ZeroDivisionError, ValueError):
        # Просим пользователя повторить ввод
        print("Ошибка! Введите корректное ненулевое число.")
```

IT CAREER HUB

Сообщение об ошибке

Для уточнения причины исключения можно присвоить объект исключения переменной.

Python

```python
try:
    # Попытка преобразования строки в число
    number = int("abc")
except (ZeroDivisionError, ValueError) as e:
    # Выводим текст исключения
    print(f"Произошла ошибка: {e}")
```

IT CAREER HUB

Задание для закрепления 1

Какой результат будет выведен при выполнении следующего кода?

Python

```python
try:
    x = 1 / 0
except Exception:
    print("Общее исключение")
except ZeroDivisionError:
    print("Ошибка деления на ноль")
```

a) Ошибка деления на ноль
b) Общее исключение
c) Ошибка деления на ноль, затем общее исключение
d) Ошибка выполнения

Посмотреть ответ

IT CAREER HUB

Конструкция try-except-else

`else` в конструкции `try-except` используется для выполнения кода, который должен быть выполнен **только если исключение не возникло** в блоке `try`.

Синтаксис

```python
try:
    # Код, который может вызвать исключение
except ExceptionType:
    # Код, который выполнится при возникновении исключения
else:
    # Код, который выполнится, если исключение НЕ возникло
```

Пример

```python
try:
    # Преобразование строки в число
    number = int(input("Введите число: "))
except ValueError:
    # Обработка некорректного ввода
    print("Ошибка! Введите корректное число.")
else:
    # Выполняется только если исключения не было
    print(f"Вы ввели число: {number}")
```

Зачем использовать else?

Код в блоке `else` явно отделён от кода, который может вызвать исключение.

- Это улучшает читаемость программы.
- Позволяет избежать выполнения дополнительных операций в случае ошибки.

Включайте в блок `try` только **необходимый код**, который может вызвать исключение:

- Это улучшает читаемость.
- Упрощает отладку.
- Позволяет точнее обрабатывать исключения.

Пример: Использование else для разделения логики

Python

```python
try:
    # Проверка числа на чётность
    number = int(input("Введите число: "))
except ValueError:
    # Обработка некорректного ввода
    print("Ошибка! Введите корректное число.")
else:
    # Выполняется только если число успешно введено
    if number % 2 == 0:
        print(f"{number} – чётное число.")
    else:
        print(f"{number} – нечётное число.")
```

IT CAREER HUB

Конструкция try-except-finally

finally – это блок, который выполняется всегда, независимо от того, возникло ли исключение в блоке try.

Он используется для завершения операций, которые должны быть выполнены в любом случае (например, закрытие соединений, файлов, освобождение памяти).

Синтаксис

Python

```
try:
    # Код, который может вызвать исключение
except ExceptionType:
    # Код, который выполнится при возникновении исключения
finally:
    # Код, который выполнится всегда, независимо от исключений
```

Пример

Python

```
try:
    # Преобразуем строку в число
    number = int(input("Введите число: "))
    result = 10 / number
except ValueError:
    # Обработка некорректного ввода
    print("Ошибка! Введите корректное число.")
except ZeroDivisionError:
    # Обработка деления на ноль
    print("Ошибка! Деление на ноль.")
finally:
    # Этот код выполнится в любом случае
    print("Завершение программы.")
```

IT CAREER HUB

Конструкция try-except-else-finally

Эта конструкция объединяет все блоки обработки исключений и завершения:

- `try`: Выполняется код, который может вызвать исключение.
- `except`: Выполняется, если в блоке `try` возникает исключение.
- `else`: Выполняется, если в блоке `try` не возникло исключений.
- `finally`: Выполняется в любом случае, независимо от того, было исключение или нет.

Пример

Python

```python
try:
    # Попытка преобразовать строку в число и выполнить деление
    number = int(input("Введите число: "))
    result = 10 / number
except ValueError:
    # Обработка некорректного ввода
    print("Ошибка: введено некорректное значение.")
except ZeroDivisionError:
    # Обработка деления на ноль
    print("Ошибка: деление на ноль.")
else:
    # Выполняется только если исключений не было
    print(f"Результат деления: {result}")
finally:
    # Завершающие действия
    print("Программа завершена.")
```

Возбуждение исключений

Для явного возбуждения исключений в Python используется ключевое слово `raise`. Это даёт возможность разработчику вручную вызвать исключение в любой части программы, чтобы указать на возникшую проблему.

Синтаксис

```python
raise ExceptionType("Сообщение об ошибке")
```

- `ExceptionType` – тип исключения, который вы хотите вызвать (например, `ValueError`, `TypeError` или пользовательское исключение).
- "Сообщение об ошибке" – пояснение причины исключения (опционально).

Пример

```python
number = -1
if number < 0:
    raise ValueError("Число не может быть отрицательным")
```

Что делать дальше после raise?

Когда исключение возбуждается с помощью `raise`, программа останавливает выполнение, если это исключение не перехвачено.

После использования `raise`:

1. Либо перехватите исключение с помощью `try-except` и обработайте его.
2. Либо позвольте программе завершиться, чтобы пользователь увидел информацию об ошибке.

Пример

Python

```python
while True:
    try:
        # Ввод числа пользователем
        number = int(input("Введите положительное число: "))
        if number < 0:
            raise ValueError("Число не может быть отрицательным")
        print(f"Вы ввели корректное число: {number}")
        break  # Завершаем цикл, если число корректное
    except ValueError as e:
        # Обработка некорректного ввода
        print(f"Ошибка: {e}. Попробуйте снова.")
```

IT CAREER HUB

Задание для закрепления 2

1. **Что делает ключевое слово raise?**

a) Завершает выполнение программы
b) Создаёт новое исключение
c) Игнорирует исключение
d) Перехватывает ошибку

2. **Какой результат будет выведен при выполнении следующего кода?**

Python

```python
try:
    print("До исключения", end=" | ")
    raise ValueError("Ошибка!")
    print("После исключения", end=" | ")
except ValueError as e:
    print(f"Перехвачено исключение: {e}")
```

a) До исключения
b) До исключения | Ошибка!
c) До исключения | Перехвачено исключение: Ошибка!
d) До исключения | После исключения | Перехвачено исключение: Ошибка!

Посмотреть ответ

IT CAREER HUB

Логирование

Логирование — это процесс записи информации о работе программы, включая ошибки, предупреждения и отладочные сообщения.

В Python для этого используется встроенный модуль logging, который позволяет выводить сообщения разного уровня важности в консоль или файлы.

Основы логирования

Модуль logging поддерживает 5 уровней логирования (от менее серьёзных к более критичным):

Уровень	Метод	Описание
DEBUG	`logging.debug`	Отладочные сообщения (для диагностики программы).
INFO	`logging.info`	Общая информация о работе программы.
WARNING	`logging.warning`	Предупреждения о потенциальных проблемах.
ERROR	`logging.error`	Ошибки, но программа продолжает работать.
CRITICAL	`logging.critical`	Критические ошибки, после которых программа может завершиться.

Пример

```python
import logging

# Настройка логирования (по умолчанию уровень WARNING и выше)
logging.basicConfig()

# Вывод различных сообщений
logging.debug("Отладочное сообщение")
logging.info("Информационное сообщение")
logging.warning("Предупреждение!")
logging.error("Ошибка!")
logging.critical("Критическая ошибка!")
```

Вывод

```
/home/tanya/PycharmProjects/pythonPr
WARNING:root:Предупреждение!
ERROR:root:Ошибка!
CRITICAL:root:Критическая ошибка!

Process finished with exit code 0
```

Почему `DEBUG` и `INFO` не отобразились?

- По умолчанию `logging` показывает только сообщения `WARNING` и выше.
- Чтобы увидеть `DEBUG` и `INFO`, нужно задать `level=logging.DEBUG` в `basicConfig`.

```python
# Настройка логирования уровня DEBUG)

logging.basicConfig(level=logging.DEBUG)
```

IT CAREER HUB

Запись логов в файл

Можно настроить запись логов не только в консоль, но и в файл. Для этого используется параметр `filename`.

Пример

Python

```python
import logging

# Настройка логирования с записью в файл
logging.basicConfig(filename="app.log", level=logging.INFO)

logging.info("Программа запущена")
logging.warning("Низкий уровень памяти")
logging.error("Ошибка подключения к базе данных")
```

Использование логов в обработке исключений

Логирование часто используется для записи ошибок.

Пример

Python

```python
import logging

logging.basicConfig(filename="app.log", level=logging.ERROR)

try:
    result = 10 / 0
except ZeroDivisionError as e:
    logging.error(f"Ошибка: {e}")
```

Настройка формата логов

Модуль `logging` позволяет задавать собственный формат сообщений для логирования. Это делает логи более читаемыми и информативными.

Формат задаётся с помощью аргумента `format="..."` в `logging.basicConfig()`. Вот несколько ключевых плейсхолдеров, которые можно использовать.

Плейсхолдер	Описание
`%(asctime)s`	Время записи лога в формате `YYYY-MM-DD HH:MM:SS`
`%(levelname)s`	Уровень логирования (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`)
`%(filename)s`	Имя файла, в котором выполняется логирование
`%(lineno)d`	Номер строки кода, где был вызван лог
`%(message)s`	Текст самого лог-сообщения

IT CAREER HUB

Пример

Python

```python
import logging

# Настройка формата логов
logging.basicConfig(
    filename="app.log",
    format="%(asctime)s - %(filename)s - %(lineno)d - %(levelname)s -
%(message)s",
    level=logging.DEBUG
)

logging.debug("Это отладочное сообщение")
logging.info("Информационное сообщение")
logging.warning("Предупреждение")
logging.error("Ошибка")
logging.critical("Критическая ошибка")
```

IT CAREER HUB

Лучшие практики обработки исключений

1. Обрабатывайте **только ожидаемые исключения**.
    - Указывайте конкретные типы исключений, чтобы не перехватывать ненужные ошибки.
2. Минимизируйте код в блоке `try`.
    - Включайте только тот код, который может вызвать исключение.
3. Используйте `finally` для освобождения ресурсов.
    - Например, для закрытия файлов или соединений.
4. Логируйте ошибки.
    - Вместо простого вывода используйте модуль `logging` для записи ошибок.

IT CAREER HUB

Задания для закрепления 3

Что делает параметр `filename="app.log"` в `logging.basicConfig()`?

a) Определяет уровень логирования
b) Задаёт формат логов
c) Указывает, в какой файл записывать логи
d) Очищает файл перед записью

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
Результат выполнения кода	Ответ: b
Задания на закрепление 2	Вернуться к заданиям
1. Ключевое слово raise	Ответ: b
2. Результат выполнения кода	Ответ: c
Задания на закрепление 3	Вернуться к заданиям
Параметр filename="app.log"	Ответ: b

IT CAREER HUB

Практическая работа

1. Обработка ввода пользователя

Напишите программу, которая запрашивает у пользователя число и обрабатывает возможные ошибки ввода, пока не получат корректное число.

Пример вывода

```Python
Введите число: qwe

Ошибка: Введите корректное число.

Введите число: 12.5

Вы ввели число: 12.5
```

IT CAREER HUB

Решение

Python

```
def get_valid_number():
    while True:
        try:
            return float(input("Введите число: "))
        except ValueError:
            print("Ошибка: Введите корректное число.")

num = get_valid_number()
\print(f"Вы ввели число: {num}")
```

IT CAREER HUB

2. Проверка возраста

Напишите функцию, которая проверяет, что возраст пользователя не меньше 18 лет с использованием ошибок.

Пример вывода

Python

```
Введите возраст: 17
Ошибка: Возраст должен быть 18 лет и старше.
```

IT CAREER HUB

Решение

Python

```python
def check_age():
    age = int(input("Введите возраст: "))
    if age < 18:
        raise ValueError("Ошибка: Возраст должен быть 18 лет и старше.")
    print("Возраст принят.")

try:
    check_age()
except ValueError as e:

    print(e)
```