Конспект лекції (8)

Що таке виключення

Виключення - помилки, що виникають в процесі роботи програми. Наприклад, якщо ми хочему ділити на 0 - ми отримаємо ZeroDivisionError

```
# Приклад логічної помилки
def divide(a, b):
    result = a / b
    return result

divide(10, 0)
```

Ми можемо оброблювати ці виключення, щоб программа не завершувала свою роботу з помилкою. У наступному блоці розберемо наступний приклад коду:

```
# Приклад виключення
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Помилка ділення на нуль: {e}")
```

Блок `try-except-finally`

`try`

- Ми використовуємо `try` для того, щоб обернути кусок кода, в якому ми чекаємо помилку
- Він може складатись з 1 чи більше команд та строк
- У цьому блоці ми перевіряємо на виключення команду `result = 10 / 0`
- За блоком `try` **обов'язково** повинен йти блок `except`

```
try:
    result = 10 / 0
```

▼ `except`

- `except` використовується для того, щоб сказати, що ми будемо робити, якщо в блоці `try` буде виключення
- Він може складатись з 1 чи більше команд та строк
- Ми можемо вказати виключення (як на лістінгу коду нижче), яке ми хочемо ловити, чи ловити усі виключення за допомогою `except :`

```
except ZeroDivisionError as e:
    print(f"Помилка ділення на нуль: {e}")
```

- `except` -ів може бути скільки завгодно, але виконається перший з них, який підпадає під умову
- Ось приклад, в якому використовується кілька виняткових ситуацій в одному блоку `try-except`:

```
def divide_numbers(a, b):
    try:
        result = a / b
        # Виклик виключення для демонстрації
        if b == 0:
            raise ZeroDivisionError("Ділення на нуль не допускається.")
        elif b < 0:
            raise ValueError("Дільник повинен бути додатнім числом.")
    except ZeroDivisionError as zde:
        print(f"Помилка ділення на нуль: {zde}")
    except ValueError as ve:
        print(f"Помилка значення: {ve}")
    except Exception as e:
        print(f"Інша помилка: {e}")
    else:
        print(f"Результат: {result}")
    finally:
        print("Цей блок виконається завжди")

# Виклик функції з різними аргументами
divide_numbers(10, 2)     # Результат: 5.0
divide_numbers(10, 0)     # Помилка ділення на нуль: division by zero
divide_numbers(10, -2)    # Помилка значення: Дільник повинен бути додатнім чис
лом.
```

У цьому прикладі:

1. Функція `divide_numbers` проводить ділення двох чисел.

2. Блок `try` визначає, якщо дільник (`b`) дорівнює 0, то генерується виключення `ZeroDivisionError`, а якщо `b` менше 0, то генерується виключення `ValueError`.
3. Блок `except` обробляє обидві виняткові ситуації окремо.
4. Блок `else` виконується, якщо виключення не виникло.
5. Блок `finally` виконується завжди, незалежно від того, чи сталася помилка чи ні.

▼ Створюємо власне виключення

Це виключення буде виникати, коли користувач вводить число більше заданого ліміту.

```
class TooLargeValueError(Exception):
    def __init__(self, value, limit):
        self.value = value
        self.limit = limit
        message = f"Значення {value} перевищує ліміт {limit}"
        super().__init__(message)

# Приклад використання власного виключення
try:
    limit = 100
    user_input = int(input("Введіть число: "))

    if user_input > limit:
        raise TooLargeValueError(user_input, limit)
    else:
        print("Дякую! Ви ввели припустиме значення.")
except TooLargeValueError as e:
    print(f"Помилка: {e}")
except ValueError:
    print("Помилка: Будь ласка, введіть ціле число.")
```

У цьому прикладі, якщо користувач вводить число, яке перевищує ліміт (в даному випадку, 100), ми викликаємо виключення `TooLargeValueError` із відповідним повідомленням. У випадку, якщо виникає власне виключення, ми ловимо його в блоку `except TooLargeValueError` та виводимо відповідне повідомлення.

▼ Конструкція `with`

Блок with дозволяє нам працювати з ресурсами, та не боятись, що ми їх заблокуємо. Напевно, в усіх була ситуація, коли ви хочете видалити файл, а

система пише, що він використовується іншим процесом. Щоб уникнути таких ситуацій в пайтоні - використовуємо блок with:

```
with open("example.txt", "r") as file:
    content = file.read()
```

Це значить, що ми відчиняємо файл “example.txt” для читання, потім зчитуємо його в змінну content, і коли ми закінчуємо все, що є в блоку with - операційна система отримує сигнал про те, що файл “example.txt” вже вільний. Цей запис ідентичен запису нижче:

```
file = None
try:
    # Відкриття файлу для читання
    file = open("example.txt", "r")

    # Операції змістом файлу
    content = file.read()
except:
    print(f"Виникла помилка: {e}")
finally:
    # Закриття файлу у блоку finally, щоб гарантувати його виклик навіть якщо виник
ає помилка
    if file is not None:
        file.close()
```

Тут ми викликаємо file.close у блоку finally, щоб гарантувати, що і при успішній роботі програми і при помилці ми віддамо наш файл назад до операційної системи.

Домашне завдання