Lesson 27 - Console App with real DB

Повторение

1. Теория из прошлого урока
2. Разбор домашнего задания

Legend

try-except

В Python try-except используется для обработки исключений и предотвращения сбоев в работе программы при возникновении ошибок.

```
try:
    # Код, который может вызвать исключение
    x = 1 / 0
except ZeroDivisionError:  # Обработка конкретного исключения
    print("Ошибка: деление на ноль!")
```

Использование нескольких except
Можно обрабатывать разные исключения отдельно:

```
try:
    x = int("abc")  # Ошибка ValueError
except ZeroDivisionError:
    print("Ошибка: деление на ноль!")
except ValueError:
    print("Ошибка: невозможно преобразовать строку в число!")
```

except Exception as e
Позволяет получить сам объект ошибки:

```
try:
    x = 1 / 0
except Exception as e:
    print(f"Произошла ошибка: {e}")
```

else – выполняется, если ошибок не было

```
try:
    print("Всё работает!")
except:
    print("Ошибка!")
else:
    print("Ошибок нет, продолжаем выполнение.")
```

---

finally – выполняется в любом случае

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Ошибка!")
finally:
    print("Этот блок выполняется всегда.")
```

Генерация ошибок через raise

Можно вручную вызвать исключение:

```
raise ValueError("Ошибка значения!")
```

Если хотите обработать ошибку:

```
try:
    raise ValueError("Ошибка значения!")
except ValueError as e:
    print(f"Перехвачено: {e}")e
```

При работе с БД (например, psycopg2):

```
import psycopg2

try:
    conn = psycopg2.connect("dbname=test user=postgres password=secret")
```

```
except psycopg2.OperationalError as e:
    print(f"Ошибка подключения к БД: {e}")
```

Пример с обработкой случая, когда запись не найдена:

```
import psycopg2

try:
    # Подключение к базе данных
    conn = psycopg2.connect("dbname=test user=postgres password=secret")
    cursor = conn.cursor()

    # Выполнение SQL-запроса
    cursor.execute("SELECT * FROM tasks WHERE task_id = %s", (1,))
    result = cursor.fetchone()  # Получаем одну запись

    if result is None:
        print("Запись не найдена!")
    else:
        print("Задача найдена:", result)

except psycopg2.Error as e:
    print("Ошибка при работе с БД:", e)
finally:
    # Закрытие соединения
    if conn:
        cursor.close()
        conn.close()
```

Пример CRUD запросов к БД

```
import psycopg2
from psycopg2 import sql


class Task:
    def __init__(self, task_id):
        self.__task_id = task_id
        self.__title = None
        self.__description = None
        self.__deadline = None
        self.__priority = None
        self.__is_done = False
```

```
    @property
    def task_id(self):
        return self.__task_id

    @property
    def title(self):
        return self.__title

    @title.setter
    def title(self, title):
        if len(title) == 0:
            print('Заголовок не может быть пустым')
        elif len(title) > 16:
            print("Слишком длинный заголовок (макс. 16 символов)")
        else:
            self.__title = title

    @property
    def description(self):
        return self.__description

    @description.setter
    def description(self, description):
        if len(description) == 0:
            print('Описание не может быть пустым')
        elif len(description) > 64:
            print("Слишком длинное описание (макс. 64 символа")
        else:
            self.__description = description

    @property
    def deadline(self):
        return self.__deadline

    @deadline.setter
    def deadline(self, deadline):
        if len(deadline) == 0:
            print('Дедлайн не может быть пустым')
        elif len(deadline) != 10:
            print("Не правильный формат даты (DD-MM-YYYY)")
        else:
            self.__deadline = deadline

    @property
    def priority(self):
        return self.__priority
```

```
    @priority.setter
    def priority(self, priority):
        values = ["низкий", "средний", "высокий"]
        if priority not in values:
            print("Некорректное значение приоритета")
        else:
            self.__priority = priority

    @property
    def is_done(self):
        return self.__is_done

    @is_done.setter
    def is_done(self, is_done):
        if is_done is True and self.__is_done is True:
            print("Задача уже выполнена!!!")
        elif is_done is False and self.__is_done is False:
            print("Задача и так выполнена!!!!")
        else:
            self.__is_done = is_done


class TaskRepository:
    def __init__(self, db_config):
        self.db_config = db_config

    def connect(self):
        try:
            conn = psycopg2.connect(**self.db_config)
            return conn
        except psycopg2.Error as e:
            print(f"Ошибка при подключении к базе данных: {e}")
            return None

    def create_task(self, task):
        try:
            conn = self.connect()
            if conn is None:
                return
            cursor = conn.cursor()
            query = """
                INSERT INTO tasks (title, description, deadline, priority,
is_done)
                VALUES (%s, %s, %s, %s, %s) RETURNING task_id;
            """
```

```
            cursor.execute(query, (task.title, task.description,
task.deadline, task.priority, task.is_done))
            task_id = cursor.fetchone()[0]
            conn.commit()
            cursor.close()
            conn.close()
            return task_id
        except psycopg2.Error as e:
            print(f"Ошибка при добавлении задачи: {e}")
        finally:
            if conn:
                conn.close()

    def get_task(self, task_id):
        try:
            conn = self.connect()
            if conn is None:
                return
            cursor = conn.cursor()
            query = "SELECT * FROM tasks WHERE task_id = %s;"
            cursor.execute(query, (task_id,))
            result = cursor.fetchone()
            if result is None:
                print(f"Задача с ID {task_id} не найдена")
            else:
                task = Task(result[0])
                task.title = result[1]
                task.description = result[2]
                task.deadline = result[3]
                task.priority = result[4]
                task.is_done = result[5]
                cursor.close()
                conn.close()
                return task
        except psycopg2.Error as e:
            print(f"Ошибка при получении задачи: {e}")
        finally:
            if conn:
                conn.close()

    def update_task(self, task):
        try:
            conn = self.connect()
            if conn is None:
                return
            cursor = conn.cursor()
```

```
            query = """
                UPDATE tasks
                SET title = %s, description = %s, deadline = %s, priority =
%s, is_done = %s
                WHERE task_id = %s;
            """
            cursor.execute(query, (task.title, task.description,
task.deadline, task.priority, task.is_done, task.task_id))
            conn.commit()
            cursor.close()
            conn.close()
        except psycopg2.Error as e:
            print(f"Ошибка при обновлении задачи: {e}")
        finally:
            if conn:
                conn.close()

    def delete_task(self, task_id):
        try:
            conn = self.connect()
            if conn is None:
                return
            cursor = conn.cursor()
            query = "DELETE FROM tasks WHERE task_id = %s;"
            cursor.execute(query, (task_id,))
            conn.commit()
            cursor.close()
            conn.close()
        except psycopg2.Error as e:
            print(f"Ошибка при удалении задачи: {e}")
        finally:
            if conn:
                conn.close()


# Пример использования
db_config = {
    'dbname': 'your_database',
    'user': 'your_user',
    'password': 'your_password',
    'host': 'localhost',
    'port': '5432'
}

repo = TaskRepository(db_config)
```

```
# Создание задачи
task = Task(1)
task.title = "Задача 1"
task.description = "Описание задачи 1"
task.deadline = "31-12-2025"
task.priority = "средний"
task.is_done = False

task_id = repo.create_task(task)
print(f"Задача с ID {task_id} успешно добавлена.")

# Получение задачи по ID
task_from_db = repo.get_task(task_id)
if task_from_db:
    print(f"Задача: {task_from_db.title}, {task_from_db.description},
{task_from_db.deadline}, {task_from_db.priority}, Статус: {'Выполнено' if
task_from_db.is_done else 'Не выполнено'}")

# Обновление задачи
task_from_db.title = "Обновленная задача"
task_from_db.description = "Обновленное описание"
repo.update_task(task_from_db)

# Удаление задачи
repo.delete_task(task_id)
```

ДЗ-27

```
```