Винятки: Теорія та Концепції

1.1. Що таке Винятки (Exceptions)?

У програмуванні **виняток (exception)** — це подія, яка виникає під час виконання програми і порушує нормальний потік інструкцій. Це не синтаксична помилка (яку виявляє інтерпретатор до запуску коду), а **помилка часу виконання (runtime error)**.

Коли виникає виняток, Python "піднімає" (raises) його, і якщо цей виняток не оброблено, програма зупиняється і виводить повідомлення про помилку (traceback).

Приклад:

```
# Це синтаксична помилка - програма не запуститься
print("Привіт"

# Це виняток - програма запуститься, але впаде під час виконання
print(10 / 0) # ZeroDivisionError
my_list = [1, 2]
print(my_list[5]) # IndexError
```

1.2. Навіщо обробляти Винятки?

Обробка винятків є критично важливою для створення надійних, стабільних та зручних для користувача програм, особливо у веб-розробці:

- **Стабільність програми:** Запобігає неочікуваним збоям програми. Замість "падіння" програми, ви можете коректно обробити помилку та дозволити програмі продовжити роботу.
- **Зручний користувацький інтерфейс:** Замість незрозумілих повідомлень про помилки та трасування стека, користувач отримує зрозуміле повідомлення про те, що сталося, або дружню сторінку помилки (наприклад, 404 Not Found, 500 Internal Server Error).
- **Безпека:** Необроблені винятки можуть розкривати конфіденційну інформацію про внутрішню структуру вашої програми (наприклад, шляхи до файлів, деталі бази даних), що може бути використано зловмисниками.
- **Підтримка та налагодження:** Централізована обробка винятків дозволяє логувати помилки, що значно спрощує їх пошук та виправлення.
- **Коректне керування ресурсами:** Навіть якщо виникає помилка, ви можете

гарантувати закриття файлів, з'єднань з базою даних тощо.

1.3. Ієрархія Винятків та їхні Типи

У Python всі винятки є класами, які успадковуються від базового класу BaseException. Більшість вбудованих винятків успадковуються від класу Exception.

Схема: Спрощена ієрархія вбудованих винятків у Python.

Деякі поширені вбудовані типи винятків:

- **SyntaxError**: Виникає, коли парсер Python виявляє синтаксичну помилку. (Виняток, але зазвичай виявляється до виконання).
- **TypeError**: Операція або функція застосована до об'єкта невідповідного типу.
  - *Приклад:* len(123), 5 + "рядок"
- **ValueError**: Операція або функція застосована до об'єкта правильного типу, але з невідповідним значенням.
  - *Приклад:* int("abc"), math.sqrt(-1)
- **NameError**: Змінна або функція не знайдена.
  - *Приклад:* print(undefined_variable)
- **IndexError**: Індекс послідовності (списку, кортежу) знаходиться поза допустимим діапазоном.
  - *Приклад:* my_list[10] для my_list = [1, 2]
- **KeyError**: Ключ не знайдено у словнику.
  - *Приклад:* my_dict['non_existent_key'] для my_dict = {'a': 1}
- **ZeroDivisionError**: Ділення або операція за модулем на нуль.
  - *Приклад:* 10 / 0
- **FileNotFoundError**: Файл або директорія не знайдена.
  - *Приклад:* open("non_existent_file.txt")
- **IOError**: Загальна помилка вводу/виводу. (Часто FileNotFoundError є підкласом IOError).
- **AttributeError**: Спроба доступу до неіснуючого атрибута або методу об'єкта.
  - *Приклад:* my_object.non_existent_method()
- **ImportError**: Не вдалося імпортувати модуль або ім'я з модуля.
  - *Приклад:* import non_existent_module
- **TimeoutError**: Операція перевищила встановлений час очікування (особливо актуально для мережевих запитів).
  - *Приклад:* При спробі підключитися до зовнішнього API, який не відповідає.
- **ConnectionRefusedError**: Спроба підключення була відхилена (наприклад, сервер не працює).
- **HTTPException (з веб-фреймворків):** Специфічні для веб-розробки винятки,

що відповідають HTTP-статусам (наприклад, 404 Not Found, 400 Bad Request, 500 Internal Server Error).

1.4. Блоки обробки Винятків: try, except, else, finally

Для обробки винятків у Python використовується конструкція try-except-else-finally.

- **try блок:**
  - Містить код, який може викликати виняток.
  - Якщо виняток виникає всередині try блоку, виконання цього блоку негайно припиняється, і Python шукає відповідний except блок.
  - Якщо виняток не виникає, except блоки пропускаються.
- **except блок:**
  - Визначає, який тип винятку він буде обробляти.
  - Ви можете вказати конкретний тип винятку (наприклад, except ValueError:), кілька типів (наприклад, except (TypeError, ValueError):), або не вказувати тип взагалі (для обробки будь-якого винятку, але це зазвичай не рекомендується).
  - Можна захопити об'єкт винятку за допомогою as e (наприклад, except ValueError as e:), щоб отримати доступ до деталей помилки.
- **else блок (необов'язковий):**
  - Виконується, якщо код у try блоці завершився **без винятків**.
  - Корисно для коду, який повинен виконуватися лише тоді, коли try блок був успішним.
- **finally блок (необов'язковий):**
  - Виконується **завжди**, незалежно від того, чи виник виняток у try блоці, чи був він оброблений, чи ні.
  - Ідеально підходить для операцій очищення, таких як закриття файлів, з'єднань з базою даних, звільнення ресурсів.

Загальна структура:

```
try:
    # Код, який може викликати виняток
    # (наприклад, операції з файлами, мережеві запити, обчислення)
except SpecificException1:
    # Обробка SpecificException1
except SpecificException2 as e:
    # Обробка SpecificException2, доступ до об'єкта винятку 'e'
except (SpecificException3, SpecificException4):
```

```
    # Обробка кількох винятків одного типу
except Exception as e: # Захоплює будь-який інший виняток, що
успадковується від Exception
    # Загальна обробка інших винятків
    # (завжди розташовуйте загальніші винятки нижче конкретних)
else:
    # Цей код виконується, якщо винятків у try блоці НЕ було
finally:
    # Цей код виконується завжди, незалежно від винятку
    # (наприклад, закриття файлів, з'єднань)
```

1.5. Генерування Винятків (raise)

Ви можете викликати винятки вручну за допомогою оператора raise. Це корисно, коли виявляєте невідповідні умови, які ваша функція не може обробити.

```
def process_age(age):
    if not isinstance(age, int):
        raise TypeError("Вік має бути цілим числом.")
    if age < 0 or age > 120:
        raise ValueError("Вік має бути в діапазоні від 0 до 120.")
    print(f"Вік оброблено: {age}")

try:
    process_age("двадцять")
except TypeError as e:
    print(f"Помилка типу: {e}")

try:
    process_age(200)
except ValueError as e:
    print(f"Помилка значення: {e}")
```

Ви також можете використовувати raise без аргументів у except блоці, щоб повторно викликати виняток після його часткової обробки. Це корисно для логування помилки, а потім передачі її далі по стеку викликів.

```
try:
    # Деякий код, що викликає помилку
    result = 10 / 0
except ZeroDivisionError as e:
    print(f"Виникла помилка ділення на нуль: {e}")
    raise # Повторно викликає той самий виняток
```

1.6. Створення Власних (Кастомних) Винятків

Ви можете створювати власні типи винятків, успадковуючи їх від Exception (або іншого відповідного вбудованого винятку). Це дозволяє вам створювати більш специфічні та зрозумілі помилки, які відображають логіку вашого додатку.

Коли це корисно:

- Для виділення специфічних помилок бізнес-логіки (наприклад, InvalidUserDataError, ProductNotFoundError).
- Для покращення читабельності коду та полегшення обробки конкретних сценаріїв.

Приклад:

```
class InvalidInputError(Exception):
    """Виняток, що виникає при некоректних вхідних даних."""
    def __init__(self, message="Некоректні вхідні дані."):
        self.message = message
        super().__init__(self.message)

class UserNotFoundError(Exception):
    """Виняток, що виникає, коли користувача не знайдено."""
    def __init__(self, user_id, message="Користувача не знайдено."):
        self.user_id = user_id
        self.message = f"{message} ID: {user_id}"
        super().__init__(self.message)

def get_user_by_id(user_id):
    if user_id < 0:
        raise InvalidInputError("ID користувача не може бути
від'ємним.")
    if user_id != 100: # Імітуємо, що користувач з ID 100 існує
```

```
        raise UserNotFoundError(user_id)
    return {"id": user_id, "name": "Тестовий Користувач"}

try:
    get_user_by_id(-5)
except InvalidInputError as e:
    print(f"Помилка вхідних даних: {e}")

try:
    get_user_by_id(99)
except UserNotFoundError as e:
    print(f"Помилка пошуку користувача: {e}")
    print(f"Спроба знайти ID: {e.user_id}")
```

Практичні Приклади Винятків

```
import os
import json
import requests # Для імітації мережевих запитів
import time
import logging
```

```
# Налаштування базового логування
logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')
```

Приклад 1: Базова обробка ZeroDivisionError

Простий приклад використання try-except.

```
def safe_divide(numerator, denominator):
    try:
        result = numerator / denominator
        print(f"Результат ділення: {result}")
    except ZeroDivisionError:
        print("Помилка: Спроба ділення на нуль!")
    except TypeError: # Можна додати обробку інших помилок
        print("Помилка: Некоректні типи аргументів для ділення.")
    print("Продовження виконання програми після ділення.")

safe_divide(10, 2)
safe_divide(10, 0)
safe_divide("abc", 5) # Викличе TypeError
```

Приклад 2: Обробка кількох типів винятків

Демонструє обробку різних винятків у різних `except` блоках.

```
def get_element_from_list(data_list, index):
    try:
        value = data_list[index]
        print(f"Значення за індексом {index}: {value}")
    except IndexError:
        print(f"Помилка: Індекс {index} виходить за межі списку.")
    except TypeError:
        print("Помилка: Індекс має бути цілим числом.")
    except Exception as e: # Захоплює будь-який інший виняток
        print(f"Виникла неочікувана помилка: {e}")

my_data_list = [10, 20, 30]
get_element_from_list(my_data_list, 1)
get_element_from_list(my_data_list, 5) # IndexError
get_element_from_list(my_data_list, "два") # TypeError
get_element_from_list(None, 0) # AttributeError (обробиться загальним
Exception)
```

Приклад 3: Використання `else` та `finally`

Демонструє, коли виконуються блоки `else` та `finally`.

```
def process_file_operation(filename, mode):
    file_handle = None # Ініціалізуємо змінну
    try:
        file_handle = open(filename, mode)
        content = file_handle.read()
        print(f"Файл '{filename}' успішно прочитано. Зміст:
'{content[:20]}...'")
    except FileNotFoundError:
```

```
        print(f"Помилка: Файл '{filename}' не знайдено.")
    except IOError as e:
        print(f"Помилка вводу/виводу при роботі з файлом
'{filename}': {e}")
    except Exception as e:
        print(f"Виникла неочікувана помилка: {e}")
    else: # Виконується, якщо винятків НЕ було
        print(f"Операція з файлом '{filename}' завершилася успішно
(без винятків).")
    finally: # Виконується завжди
        if file_handle:
            file_handle.close()
            print(f"Файл '{filename}' закрито (з `finally`).")
        else:
            print(f"Дескриптор файлу для '{filename}' не був
відкритий.")

# Створюємо тестовий файл
test_file_name = "test_data.txt"
with open(test_file_name, 'w') as f:
    f.write("Це тестовий вміст файлу.")

process_file_operation(test_file_name, 'r') # Успішне виконання
process_file_operation("non_existent_file.txt", 'r') #
FileNotFoundError
process_file_operation(test_file_name, 'invalid_mode') # ValueError
(обробиться загальним Exception)

# Очищення
if os.path.exists(test_file_name):
    os.remove(test_file_name)
```

Приклад 4: Генерування винятків (`raise`)

Демонструє, як викликати винятки вручну та повторно.

```python
def validate_email(email):
    if "@" not in email or "." not in email:
        raise ValueError("Некоректний формат email-адреси.")
    if len(email) < 5:
        raise ValueError("Email-адреса занадто коротка.")
    print(f"Email '{email}' валідний.")

try:
    validate_email("test@.com")
except ValueError as e:
    print(f"Помилка валідації: {e}")

try:
    validate_email("a@b.c")
except ValueError as e:
    print(f"Помилка валідації: {e}")

try:
    validate_email("valid.email@example.com")
except ValueError as e:
    print(f"Цей email мав бути валідним, але виникла помилка: {e}")

# Приклад `raise` для повторного виклику
def process_data_with_logging(data):
    try:
        if not isinstance(data, list):
            raise TypeError("Дані мають бути списком.")
        if not data:
            raise ValueError("Список даних не може бути порожнім.")

        # Імітація обробки даних
        result = sum(data) / len(data)
        print(f"Дані оброблено, середнє: {result}")
    except (TypeError, ValueError) as e:
        logging.error(f"Помилка при обробці даних: {e}")
        raise # Повторно викликаємо виняток, щоб він був оброблений
вище
```

```
print("\nТестування `raise` для повторного виклику:")
try:
    process_data_with_logging("не список")
except TypeError:
    print("Перехоплено TypeError після повторного виклику.")
except ValueError:
    print("Перехоплено ValueError після повторного виклику.")

try:
    process_data_with_logging([])
except TypeError:
    print("Перехоплено TypeError після повторного виклику.")
except ValueError:
    print("Перехоплено ValueError після повторного виклику.")
```

Приклад 5: Створення та використання власного винятку (Custom Exception)

Демонструє, як визначити та викликати власні винятки.

```
class InvalidUserDataError(Exception):
    """Виняток, що виникає при некоректних даних користувача."""
    def __init__(self, field, value, message="Некоректні дані
користувача."):
        self.field = field
        self.value = value
        self.message = f"{message} Поле: '{field}', Значення:
'{value}'"
        super().__init__(self.message)

class DatabaseConnectionError(Exception):
    """Виняток, що виникає при проблемах з підключенням до БД."""
    def __init__(self, db_name, message="Помилка підключення до бази
даних."):
```

```
        self.db_name = db_name
        self.message = f"{message} БД: '{db_name}'"
        super().__init__(self.message)

def create_new_user(username, password, email):
    if not username or len(username) < 3:
        raise InvalidUserDataError("username", username, "Ім'я
користувача має бути не менше 3 символів.")
    if not password or len(password) < 8:
        raise InvalidUserDataError("password", "********", "Пароль
має бути не менше 8 символів.")
    if "@" not in email:
        raise InvalidUserDataError("email", email, "Некоректний
формат email.")

    # Імітація збереження в БД (може викликати
DatabaseConnectionError)
    if random.random() < 0.1: # 10% шанс на помилку БД
        raise DatabaseConnectionError("main_db")

    print(f"Користувач '{username}' успішно створений.")
    return {"username": username, "email": email}

import random

print("\nТестування створення користувача:")
for i in range(3):
    try:
        if i == 0:
            create_new_user("i", "pass", "test@example.com") #
Невалідний username
        elif i == 1:
            create_new_user("valid_user", "short",
"test@example.com") # Невалідний password
        elif i == 2:
            create_new_user("valid_user", "long_secure_pass",
"invalid-email") # Невалідний email
        else:
```

```
            create_new_user("valid_user", "long_secure_pass",
"valid@example.com") # Валідний, але може бути помилка БД
    except InvalidUserDataError as e:
        print(f"Помилка валідації даних користувача: {e.message}")
        print(f"  Поле: {e.field}, Значення: {e.value}")
    except DatabaseConnectionError as e:
        print(f"Помилка підключення до БД: {e.message}")
    except Exception as e:
        print(f"Неочікувана помилка: {e}")
```

Приклад 6: Обробка помилок в імітованому веб-запиті (Invalid Input)

Як обробляти некоректні дані, що надходять від користувача через веб-форму/API.

```
def parse_json_request(json_string):
    """Імітує парсинг JSON з HTTP-запиту."""
    try:
        data = json.loads(json_string)
        return data
    except json.JSONDecodeError as e:
        # У веб-фреймворку тут можна повернути 400 Bad Request
        raise ValueError(f"Некоректний формат JSON: {e}")

def process_order_request(request_body):
    """Імітує обробку замовлення з веб-запиту."""
    try:
        data = parse_json_request(request_body)

        # Перевірка наявності та типу полів
        product_id = data.get("product_id")
        quantity = data.get("quantity")
```

```
        if not isinstance(product_id, int) or product_id <= 0:
            raise ValueError("Поле 'product_id' має бути додатним
цілим числом.")
        if not isinstance(quantity, int) or quantity <= 0:
            raise ValueError("Поле 'quantity' має бути додатним цілим
числом.")

        # Імітація логіки замовлення
        print(f"Замовлення отримано: Product ID={product_id},
Quantity={quantity}")
        return {"status": "success", "order_id": random.randint(1000,
9999)}

    except ValueError as e:
        logging.warning(f"Помилка валідації запиту: {e}")
        # У веб-фреймворку тут можна повернути 400 Bad Request
        return {"status": "error", "message": str(e)}
    except Exception as e:
        logging.exception("Неочікувана помилка при обробці
замовлення.")
        # У веб-фреймворку тут можна повернути 500 Internal Server
Error
        return {"status": "error", "message": "Внутрішня помилка
сервера."}

# Тестування
print("\nВалідний запит:")
valid_request = '{"product_id": 101, "quantity": 5}'
response = process_order_request(valid_request)
print(f"Відповідь: {response}")

print("\nНекоректний JSON:")
invalid_json_request = '{"product_id": 101, "quantity": 5,' #
Неправильний JSON
response = process_order_request(invalid_json_request)
print(f"Відповідь: {response}")

print("\nНекоректний тип даних (quantity):")
```

```
invalid_data_type_request = '{"product_id": 101, "quantity": "пять"}'
response = process_order_request(invalid_data_type_request)
print(f"Відповідь: {response}")

print("\nВідсутнє поле (product_id):")
missing_field_request = '{"quantity": 2}'
response = process_order_request(missing_field_request)
print(f"Відповідь: {response}")
```

Приклад 7: Обробка помилок зовнішнього API (Timeout/Connection Error)

Імітація виклику зовнішнього API та обробка мережевих помилок.

```
# Імітація зовнішнього API
def mock_external_api(endpoint):
    """
    Імітує виклик зовнішнього API.
    Може викликати ConnectionError, Timeout, або повернути 404/500.
    """
    if "timeout" in endpoint:
        print("Імітація таймауту...")
        raise requests.exceptions.Timeout("API request timed out.")
    if "connection_error" in endpoint:
        print("Імітація помилки з'єднання...")
        raise requests.exceptions.ConnectionError("Failed to
establish a new connection.")
    if "404" in endpoint:
        print("Імітація 404 Not Found...")
        response = requests.Response()
        response.status_code = 404
        response._content = b'{"detail": "Not Found"}'
        raise requests.exceptions.HTTPError(response=response)
    if "500" in endpoint:
        print("Імітація 500 Internal Server Error...")
```

```
        response = requests.Response()
        response.status_code = 500
        response._content = b'{"detail": "Internal Server Error"}'
        raise requests.exceptions.HTTPError(response=response)

    print(f"Успішний виклик API для {endpoint}.")
    return {"status": "success", "data": f"Дані з {endpoint}"}

def fetch_data_from_api(endpoint):
    """Виконує запит до зовнішнього API з обробкою помилок."""
    try:
        # У реальному коді: response =
requests.get(f"https://api.example.com/{endpoint}", timeout=5)
        # response.raise_for_status() # Викличе HTTPError для 4xx/5xx
статусів

        data = mock_external_api(endpoint) # Використовуємо нашу
імітацію
        return data
    except requests.exceptions.Timeout:
        logging.error(f"Таймаут при запиті до API: {endpoint}")
        return {"status": "error", "message": "Сервіс недоступний
(таймаут)."}
    except requests.exceptions.ConnectionError:
        logging.error(f"Помилка з'єднання з API: {endpoint}")
        return {"status": "error", "message": "Не вдалося
підключитися до сервісу."}
    except requests.exceptions.HTTPError as e:
        status_code = e.response.status_code if e.response else "N/A"
        logging.error(f"HTTP-помилка від API ({status_code}):
{endpoint} - {e.response.text if e.response else 'No response'}")
        if status_code == 404:
            return {"status": "error", "message": "Ресурс не
знайдено."}
        elif status_code == 500:
            return {"status": "error", "message": "Внутрішня помилка
зовнішнього сервісу."}
        else:
```

```
            return {"status": "error", "message": f"Невідома
HTTP-помилка ({status_code})."}
    except Exception as e:
        logging.exception(f"Неочікувана помилка при роботі з API:
{endpoint}")
        return {"status": "error", "message": "Неочікувана помилка
при роботі з API."}

# Тестування
print("\nУспішний запит:")
print(fetch_data_from_api("users/1"))

print("\nЗапит з таймаутом:")
print(fetch_data_from_api("users/timeout"))

print("\nЗапит з помилкою з'єднання:")
print(fetch_data_from_api("users/connection_error"))

print("\nЗапит з 404 помилкою:")
print(fetch_data_from_api("users/404"))

print("\nЗапит з 500 помилкою:")
print(fetch_data_from_api("users/500"))
```