Конспект лекції (10)

▼ Вступ в логування

В Python існують різні способи логування подій (logging events). Модуль `logging` є стандартним засобом для логування в Python. Ось декілька прикладів, як ви можете використовувати його для реєстрації подій:

```
import logging

# Налаштування конфігурації логування
logging.basicConfig(filename='example.log', level=logging.D

# Логування подій різного рівня (DEBUG, INFO, WARNING, ERRO
logging.debug('Це повідомлення рівня DEBUG')
logging.info('Це повідомлення рівня INFO')
logging.warning('Це повідомлення рівня WARNING')
logging.error('Це повідомлення рівня ERROR')
logging.critical('Це повідомлення рівня CRITICAL')
```

У цьому прикладі логи будуть зберігатися в файлі "example.log". Ви можете змінити рівень логування за допомогою параметра `level`, і тільки повідомлення з обраного рівня і вище будуть записані.

Якщо ви хочете логувати події в консоль, можете використовувати `StreamHandler`:

```
import logging

# Налаштування конфігурації логування
logging.basicConfig(level=logging.DEBUG, format='%(asctime)

# Додавання обробника для виводу в консоль
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.DEBUG)
logging.getLogger('').addHandler(console_handler)

# Логування подій різного рівня
```

```
logging.debug('Це повідомлення рівня DEBUG')
logging.info('Це повідомлення рівня INFO')
logging.warning('Це повідомлення рівня WARNING')
logging.error('Це повідомлення рівня ERROR')
logging.critical('Це повідомлення рівня CRITICAL')
```

В цьому випадку логи будуть виводитися в консоль. Ви можете налаштувати обробник так, як вам потрібно, і додавати його до логера за допомогою `addHandler`.

▼ Конфігурація логеру

Конфігурація логера може бути виконана різними способами. Одним із підходів є використання файлу конфігурації, який містить параметри налаштувань логування. Інший підхід - використання коду для налаштування логера.

Використання Файлу Конфігурації:

Створіть файл конфігурації, наприклад, `logging_config.ini`:

```
[loggers]
keys=root,sampleLogger

[handlers]
keys=consoleHandler,fileHandler

[formatters]
keys=sampleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_sampleLogger]
level=DEBUG
handlers=fileHandler
qualname=sampleLogger
propagate=0

[handler_consoleHandler]
```

```
class=StreamHandler
level=DEBUG
formatter=sampleFormatter
args=(sys.stdout,)

[handler_fileHandler]
class=FileHandler
level=DEBUG
formatter=sampleFormatter
args=('example.log',)

[formatter_sampleFormatter]
format=%(asctime)s - %(levelname)s - %(message)s
datefmt=%Y-%m-%d %H:%M:%S
```

Тепер використайте цей файл для конфігурації логера в коді Python:

```
import logging
import logging.config

logging.config.fileConfig('logging_config.ini')

# Використання логера
logger = logging.getLogger('sampleLogger')

logger.debug('Це повідомлення рівня DEBUG')
logger.info('Це повідомлення рівня INFO')
```

Використання Коду для Конфігурації:

```
import logging

# Створення конфігурації
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s - %(levelname)s - %
                    handlers=[
                        logging.StreamHandler(),  # Виведен
                        logging.FileHandler('example.log')
```

```
                    ])

# Використання логера
logger = logging.getLogger(__name__)

logger.debug('Це повідомлення рівня DEBUG')
logger.info('Це повідомлення рівня INFO')
```

Обирайте підхід, який краще відповідає вашим потребам. Файл конфігурації може бути зручним для великих проектів, де вам потрібно легко змінювати параметри логування без зміни коду. В інших випадках, конфігурація в коді може бути досить зручною.

▼ Обробники-захоплювачі повідомлень

▼ Захват у файл

Щоб створити обробник для запису в файл за допомогою модуля `logging`, вам слід використовувати клас `FileHandler`. Ось приклад, як ви можете це зробити:

```
import logging

# Створення логера
logger = logging.getLogger(__name__)

# Налаштування рівня логування
logger.setLevel(logging.DEBUG)

# Створення обробника для запису в файл
file_handler = logging.FileHandler('logfile.txt')

# Налаштування рівня логування для обробника
file_handler.setLevel(logging.DEBUG)

# Створення форматера для обробника
formatter = logging.Formatter('%(asctime)s - %(levelname
file_handler.setFormatter(formatter)

# Додавання обробника до логера
```

```
logger.addHandler(file_handler)

# Логування подій
logger.debug('Це повідомлення рівня DEBUG')
logger.info('Це повідомлення рівня INFO')
logger.warning('Це повідомлення рівня WARNING')
logger.error('Це повідомлення рівня ERROR')
logger.critical('Це повідомлення рівня CRITICAL')
```

У цьому прикладі створюється обробник `FileHandler`, який записує логи у файл з ім'ям "logfile.txt". Форматтер визначає, яким чином буде виглядати кожне повідомлення в файлі.

Важливо пам'ятати, що краще закривати обробник після використання, щоб гарантувати коректне завершення запису в файл:

```
file_handler.close()
```

Це може бути важливим, особливо якщо ви використовуєте логер в довгостроковому процесі.

▼ Захват stdout

Щоб створити обробник для виводу в `stdout` (стандартний вивід), ви можете використовувати клас `StreamHandler` з модуля `logging`. Ось приклад:

```
import logging

# Створення логера
logger = logging.getLogger(__name__)

# Налаштування рівня логування
logger.setLevel(logging.DEBUG)

# Створення обробника для виводу в stdout
console_handler = logging.StreamHandler()

# Налаштування рівня логування для обробника
console_handler.setLevel(logging.DEBUG)
```

```
# Створення форматера для обробника
formatter = logging.Formatter('%(asctime)s - %(levelname
console_handler.setFormatter(formatter)

# Додавання обробника до логера
logger.addHandler(console_handler)

# Логування подій
logger.debug('Це повідомлення рівня DEBUG')
logger.info('Це повідомлення рівня INFO')
logger.warning('Це повідомлення рівня WARNING')
logger.error('Це повідомлення рівня ERROR')
logger.critical('Це повідомлення рівня CRITICAL')
```

У цьому прикладі створюється обробник `StreamHandler`, який виводить логи на стандартний вивід (`stdout`). Ви можете налаштувати рівень логування, вибрати формат повідомлень та додати цей обробник до логера.

Якщо вам не потрібно виводити повідомлення з певного рівня, ви можете змінити рівень логування для обробника. Наприклад, якщо ви хочете виводити тільки повідомлення рівня INFO і вище, встановіть `console_handler.setLevel(logging.INFO)`.

▼ Одночасне використання StreamHandler та FileHandler

Ви можете одночасно використовувати обидва обробники (`StreamHandler` і `FileHandler`) у логері для виводу інформації як в консоль, так і в файл. Ось приклад:

```
import logging

# Створення логера
logger = logging.getLogger(__name__)

# Налаштування рівня логування
logger.setLevel(logging.DEBUG)

# Створення обробника для виводу в stdout (консоль)
```

```
console_handler = logging.StreamHandler()

# Створення обробника для запису в файл
file_handler = logging.FileHandler('logfile.txt')

# Налаштування рівня логування для обробників
console_handler.setLevel(logging.DEBUG)
file_handler.setLevel(logging.DEBUG)

# Створення форматера для обробників
formatter = logging.Formatter('%(asctime)s - %(levelname

# Налаштування форматера для обробників
console_handler.setFormatter(formatter)
file_handler.setFormatter(formatter)

# Додавання обробників до логера
logger.addHandler(console_handler)
logger.addHandler(file_handler)

# Логування подій
logger.debug('Це повідомлення рівня DEBUG')
logger.info('Це повідомлення рівня INFO')
logger.warning('Це повідомлення рівня WARNING')
logger.error('Це повідомлення рівня ERROR')
logger.critical('Це повідомлення рівня CRITICAL')
```

У цьому прикладі обидва обробники (`console_handler` і `file_handler`) додаються до логера, і повідомлення будуть виводитися як у консоль, так і у файл.

Цей підхід дозволяє вам налаштовувати вивід логів для різних цілей, таких як відладка в консолі та зберігання повідомлень в файлі для подальшого аналізу чи архівації.

▼ Можливості форматування логеру

Форматування логера визначає те, які інформаційні елементи включаються у кожне повідомлення логу. Ви можете використовувати ключові слова

форматування для додавання різних атрибутів, таких як час логування, рівень логування, текст повідомлення та інші. Ось пояснення деяких з них:

- **%(asctime)s**: Додає час логування у форматі "рік-місяць-день година:хвилина:секунда,мілісекунда".
- **%(levelname)s**: Додає рівень логування (наприклад, DEBUG, INFO, WARNING, ERROR, CRITICAL).
- **%(message)s**: Додає текстове повідомлення логу.
- **%(filename)s**: Додає ім'я файлу, з якого було викликано логування.
- **%(funcName)s**: Додає ім'я функції, з якої було викликано логування.
- **%(lineno)d**: Додає номер рядка у файлі, з якого було викликано логування.
- **%(name)s**: Додає ім'я логера.

Ви можете використовувати ці ключові слова у своєму форматі для створення власного шаблону логування. Ось приклад, як встановити форматування для логера:

```
import logging

# Створення логера
logger = logging.getLogger(__name__)

# Налаштування рівня логування
logger.setLevel(logging.DEBUG)

# Створення обробника для виводу в stdout (консоль)
console_handler = logging.StreamHandler()

# Налаштування рівня логування для обробника
console_handler.setLevel(logging.DEBUG)

# Створення форматера для обробника
formatter = logging.Formatter('%(asctime)s - %(levelname)s

# Налаштування форматера для обробника
console_handler.setFormatter(formatter)
```

```
# Додавання обробника до логера
logger.addHandler(console_handler)

# Логування подій
logger.debug('Це повідомлення рівня DEBUG')
logger.info('Це повідомлення рівня INFO')
```

В цьому прикладі `%` перед ключовим словом вказує на використання цього ключового слова для форматування. Можете змінити порядок, додавати або видаляти ключові слова, щоб відповідати вашим потребам форматування логування.

▼ Аналіз логів

Аналіз логів - це процес обробки і вивчення інформації, яка міститься в лог-файлах, щоб винести корисні висновки або виявити проблеми. Логи можуть містити інформацію про помилки, винятки, події та стан системи. Аналіз логів є важливою частиною відлагодження програм, виявлення проблем безпеки та моніторингу додатків.

Ось деякі загальні підходи до аналізу логів:

1. **Пошук помилок та винятків:** Аналізуйте логи на предмет повідомлень про помилки та винятки. Шукайте знаки несправностей у вихідних даних.
2. **Моніторинг продуктивності:** Вивчайте логи, щоб виявити області, де продуктивність може бути покращена. Це може включати виявлення довгих запитів, великої кількості помилкових запитів та інше.
3. **Виявлення вторгнень:** Перевіряйте логи на незвичайні події або активність, яка може свідчити про вторгнення або інші безпекові проблеми.
4. **Моніторинг використання ресурсів:** Вивчайте логи, щоб визначити, як використовуються ресурси, такі як пам'ять, процесор та інші.
5. **Відслідковування користувацької активності:** Якщо логи містять інформацію про користувацьку активність, вивчайте їх для зрозуміння того, як користувачі взаємодіють з системою.
6. **Відладка:** Використовуйте логи для відлагодження додатку. Детальні повідомлення про стан системи можуть допомогти зрозуміти, що сталося в той час, коли виникла проблема.

Щоб виконати аналіз логів, ви можете використовувати різні інструменти та техніки:

- **ELK Stack (Elasticsearch, Logstash, Kibana):** Потужна платформа для збору, обробки та візуалізації логів.
- **Splunk:** Інструмент для пошуку, моніторингу та аналізу даних.
- **Графіки та дашборди:** Створюйте графіки та дашборди для візуалізації та аналізу логів.
- **Регулярні вирази:** Використовуйте регулярні вирази для ефективного пошуку та вилучення інформації з лог-файлів.
- **Системи збору метрик:** Використовуйте власні системи для збору та відображення метрик, які можуть бути корисними для аналізу логів.

При аналізі логів важливо мати зрозуміння структури та форматування лог-повідомлень у вашому конкретному додатку чи середовищі.

А**наліз логів unittest**

Аналіз логів в рамках модуля `unittest` в основному відбувається під час виконання тестів. `unittest` може генерувати лог-повідомлення про кожен тест, якщо ви встановите
рівень логування на потрібний. Зазвичай це робиться через вбудований модуль
`logging`.

▼ Строгі твердження assert

В мові програмування Python твердження `assert` використовується для перевірки істинності виразу. Якщо вираз є `False`, тобто не відповідає очікуваному стану, то виконання програми призупиняється, і викидається виключення `AssertionError`.

Синтаксис твердження `assert` виглядає наступним чином:

```
assert вираз, [повідомлення_про_помилку]
```

- `вираз` - це умова, яка повинна бути істинною.
- `повідомлення_про_помилку` (необов'язково) - це рядок, який буде виведено у випадку, якщо твердження виявиться хибним. Це дає можливість зазначити більш детальне повідомлення про те, що пішло не так.

Приклади використання твердження `assert`:

```
x = -5
assert x > 0, f"x=={x}, але повинно бути додатнім числом"
```

У цьому прикладі, якщо значення `x` не є додатнім числом, виконається твердження про помилку і виведеться повідомлення `"повинно бути додатнім числом"`.

Твердження `assert` часто використовується для перевірки попередніх умов, допомагаючи забезпечити коректність програми та розробляти її в тестовому режимі.

Важливо враховувати, що в релізному коді використання тверджень `assert` може бути відключено за допомогою флага `-O` при запуску програми (наприклад, `python -O script.py`).