



Politechnika
Wrocławska

Języki Skryptowe

Elementy *The Python Standard Library*: logowanie,
konfigurowanie, przetwarzanie wyrażeń regularnych.
Tworzenie tekstowych interfejsów użytkownika.

Marcin Jodłowiec

27 marca 2024

Agenda

- Praca z logami
- Praca z wyrażeniami regularnymi
- Interfejsy w trybie tekstowym
 - Interfejsy linii komend (CLI)
 - Tekstowe interfejsy użytkownika (TUI)
- Praca z plikami konfiguracyjnymi
- Inne moduły

Praca z logami: moduł **logging** I

- ▶ Logowanie oznacza śledzenie zdarzeń, podczas działania programu.
- ▶ Zdarzenie można opisać wiadomością tekstową, która może zawierać opcjonalnie zmienne dane.
- ▶ Zdarzenia mają różny poziom istotności.

Poziomy istotności modułu **logging** i ich semantyka

- ▶ **DEBUG** – szczegółowa informacja, charakter diagnostyczny, deweloperski.
- ▶ **INFO** – potwierdzenie, że sprawy dzieją się tak, jak powinny.
- ▶ **WARNING** – Wskazanie nieoczekiwanego zdarzenia, prognoza problemu w przyszłości.
- ▶ **ERROR** – Pewne działania programu, z powodu błędu, nie mogły zostać wykonane, ale program jest wciąż w stabilnym stanie.
- ▶ **CRITICAL** – Poważny błąd wskazujący na to, że program może być niezdolny do kontynuacji działania.

Praca z logami: moduł `logging` II

- ▶ Moduł `logging` dostarcza funkcji do logowania zdarzeń o konkretnym poziomie istotności:
 - ▶ `debug()`, `info()`, `warning()`, `error()`, `critical()`

Zadanie	Sposób rozwiązania
Wyświetlenie wyniku programu (np. obliczeń) na wyjście standardowe	<code>print()</code>
Raportowanie o zdarzeniach występujących podczas standardowego działania programu.	<code>logging.info()</code> , <code>logging.debug()</code>
Ostrzeżenie o zdarzeniu	<code>logging.warning()</code>
Zgłoszenie błędu w celu jego obsługi	wyrzucenie wyjątku
Zgłoszenie wychwyconego błędu bez przerywania działania programu	<code>logging.error()</code> , <code>logging.exception()</code> , <code>logging.critical()</code> w zależności od rodzaju błędu

Proste przykłady

- ▶ Logi na poziomie DEBUG oraz INFO nie są widoczne.

```
1 >>> import logging
2 >>> logging.debug("To sa informacje diagnostyczne")
3 >>> logging.info("Na szczescie wszystko jest w porzadku :)")
4 >>> logging.warning("Uwaga!")
5 WARNING:root:Uwaga!
```

- ▶ Konfiguracja poziomu logowania

```
1 >>> logging.basicConfig(encoding='utf-8', level=logging.DEBUG)
2 >>> logging.debug("Informacja diagnostyczna")
3 DEBUG:root:Informacja diagnostyczna
```

- ▶ Logowanie do pliku

```
1 >>> import logging
2 >>> logging.basicConfig(filename='my.log', encoding = 'utf-8', level=logging.INFO)
3 >>> logging.info("Ta informacja idzie do pliku")
4 >>> logging.critical("TA TEZ!")
5 >>> exit()
6
7 cat my.log
8 INFO:root:Ta informacja idzie do pliku
9 CRITICAL:root:TA TEZ!
```

Customizacja logów

- ▶ Atrybuty struktury **LogRecord**
 - ▶ asctime, filename, funcName, message, levelname, ...
- ▶ Zmiana formatu

```
1 >>> import logging
2 >>> logging.basicConfig(format='%%(asctime)s\t%(levelname)s: %(message)s', level=logging
    .DEBUG)
3 >>> logging.debug("Informacja diagnostyczna!")
4 2023-03-29 02:15:46.941 DEBUG:Informacja diagnostyczna!
```

- ▶ logger, formatter, StreamHandler i FileHandler – API do złożonej obsługi logowania

```
1 import logging,sys
2
3 logging.basicConfig(level=logging.DEBUG)
4
5 console_handler = logging.StreamHandler(sys.stderr)
6 console_handler.setLevel(logging.ERROR)
7 formatter = logging.Formatter('%(levelname)s: %(message)s')
8 console_handler.setFormatter(formatter)
9
10 file_handler = logging.FileHandler('logs.log')
11 file_handler.setLevel(logging.DEBUG)
12 formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
13 file_handler.setFormatter(formatter)
14
15 logger = logging.getLogger(__name__)
16 logger.addHandler(console_handler)
17 logger.addHandler(file_handler)
```

Praca z wyrażeniami regularnymi I

- ▶ Wyrażenia regularne (ang. *regular expressions* – regex, regex patterns)
- ▶ Symboliczna notacja do definiowania języków regularnych.
- ▶ Dopasowywanie ciągów znaków do wzorca.
- ▶ Przetwarzanie wyrażeń regularnych w Pythonie: moduł `re`
- ▶ `re` kompiluje wyrażenia do postaci bajtowej i uruchamia silnik napisany w C
- ▶ znaki specjalne: `. ^ $ * + ? [] \ | ()`
- ▶ surowe ciągi znaków: `r"(\w+)"` vs `'(\w+)'`

Definiowanie wyrażeń regularnych: Proste wzorce

- ▶ symbole **alfanumeryczne** dopasowują same siebie
- ▶ niektóre alfanumeryczne symbole zyskują specjalne znaczenie, gdy są poprzedzone backslashem (\)
- ▶ **znaki interpunkcyjne** – odwrotnie – poprzedzone backslashem dopasowują same siebie
- ▶ backslash jest dopasowany przez potwórzony backslash (\\)

Składnia wyrażeń regularnych I

Element	Znaczenie
<code>.</code>	Dopasowuje dowolny pojedynczy znak (oprócz <code>\n</code>).
<code>^</code>	Dopasowuje początek łańcucha.
<code>\$</code>	Dopasowuje koniec łańcucha.
<code>*</code>	Kwantyfikator: zachłannie dopasowuje zero lub więcej wystąpień poprzedniego wyrażenia.
<code>+</code>	Kwantyfikator: zachłannie dopasowuje jeden lub więcej wystąpień poprzedniego wyrażenia.
<code>?</code>	Kwantyfikator: zachłannie dopasowuje zero lub jedno wystąpienie poprzedniego wyrażenia.
<code>*?</code> , <code>+?</code> , <code>??</code>	Niezachłanne wersje <code>*</code> , <code>+</code> , <code>?</code> .

Składnia wyrażeń regularnych II

$\{m\}$	Dopasowuje dokładnie m wystąpień poprzedniego wyrażenia.
$\{m,n\}$	Dopasowuje między m a n wystąpień poprzedniego wyrażenia.
$\{m,n\}?$	Niezachłanna wersja poprzedniego wyrażenia.
$[...]$	Dopasowuje dowolny znak ze zbioru w nawiasach kwadratowych.
$[^...]$	Dopasowuje dowolny znak, który nie zawiera się w nawiasie po symbolu $^$
$ $	Dopasowuje albo wyrażenie poprzedzające albo następujące po symbolu.
$(...)$	Dopasowuje wyrażenie w nawiasie i wskazuje na grupę.

Składnia wyrażeń regularnych III

(?aiLmsux:)	Wbudowany w wyrażenie sposób ustawienia opcjonalnej flagi (a,i,L,m,s,u,x)
(?:...)	Działa jak (...), ale nie tworzy grupy.
(?P<id>...)	Działa jak (...), ale dodatkowo nazywa grupę <id>
(?P=<id>...)	Dopasowuje z grupą o nazwie <id>
(?=...)	Asercja wyprzedzająca (ang. <i>lookahead assertion</i>) – dopasowuje następne wyrażenie, ale nie konsumuje żadnej części łańcucha.
(?!...)	Negatywna asercja wyprzedzająca – dopasowuje, gdy po wyrażeniu nie następuje wyrażenie określone jako „...”

Składnia wyrażeń regularnych IV

<code>(?<=...)</code>	Asercja wsteczna (ang. <i>lookbehind assertion</i>) – dopasowuje, jeśli obecna pozycja w ciągu jest poprzedzona dopasowaniem wyrażenia „...” i kończy się w obecnej pozycji.
<code>\<num></code>	Dopasowuje zawartość grupy o numerze <code><num></code>
<code>\A</code>	Dopasowuje pusty ciąg znaków, lecz tylko na początku całego łańcucha
<code>\b</code>	Dopasowuje pusty ciąg znaków, lecz tylko na początku lub końcu słowa
<code>\B</code>	Dopasowuje pusty ciąg znaków, ale nie na początku ani końcu słowa.
<code>\d</code>	Dopasowuje jedną cyfrę (0-9)
<code>\D</code>	Dopasowuje jeden znak, który nie jest cyfrą.

Składnia wyrażeń regularnych V

<code>\s</code>	Dopasowuje biały znak (np. spację nową linię, tabulację, etc.)
<code>\S</code>	Dopasowuje jeden znak, który nie jest białym znakiem
<code>\w</code>	Dopasowuje jeden znak alfanumeryczny
<code>\W</code>	Dopasowuje jeden niealfanumeryczny znak.
<code>\Z</code>	Dopasowuje pusty łańcuch znaków, ale tylko na końcu całego łańcucha.

Składnia wyrażeń regularnych VI

Flagi

- ▶ A lub ASCII – wykorzystuje ASCII
- ▶ U lub UNICODE – wykorzystuje UNICODE
- ▶ L lub LOCALE – wykorzystuje LOCALE do określenia znaków (np. \w, \W)
- ▶ M lub MULTILINE – symbole ^ i \$ określają początek i koniec linii
- ▶ S lub DOTALL – symbol kropki dopasowuje każdy znak, łącznie ze znakiem nowej linii
- ▶ I lub IGNORECASE – tryb dopasowywania niewrażliwy na wielkość liter
- ▶ X lub VERBOSE – białe znaki we wzorcu są ignorowane .

Przykłady

- ▶ Wyrażenie dopasowujące ciągi znaków Andrzej, Stefan i Dżesika:

```
r'(Andrzej|Stefan|Dżesika)'
```

- ▶ Wyrażenie dopasowujące imię i nazwisko:

```
r'^[A-Z][a-z]+s[A-Z][a-z]+$'
```

- ▶ Wyrażenie dopasowujące imię i nazwisko poprzedzone prefiksem:

```
r'(?<=Imię i nazwisko:\s)[A-Z][a-z]+\s[A-Z][a-z]+''
```

- ▶ Wyrażenie dopasowujące adres e-mail:

```
r'^[a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z]{2,}$'
```

- ▶ wyrażenie dopasowujące nazwy plików z rozszerzeniem .txt:

```
r'\w+(?=\.txt)'
```

Przetwarzanie i wykorzystywanie wyrażeń regularnych I

► Dopasowywanie do wzorca (`re.match`)

```
1 >>> pattern = r'([a-zA-Z]+)\s\1'  
2 >>> re.match(pattern, "he he")  
3 <re.Match object; span=(0, 5), match='he he'>
```

► Grupowanie (`Match.group()`, `Match.groups()`)

```
1 >>> hehe.groups()  
2 ('he',)  
3  
4 >>> pattern = r'([a-zA-Z]+)\s\1(\1\1)'  
5 >>> hohohoho = re.match(pattern, "ho hohoho")  
6 >>> hohohoho.group(1)  
7 'ho'  
8 >>> hohohoho.group(2)  
9 'hoho'
```

► Kompilowanie (`re.compile`) – tworzenie obiektu wyrażenia regularnego

```
1 >>> pattern = r'[a-zA-Z]+'
```

```
2 >>> compiled_pattern = re.compile(pattern)
```

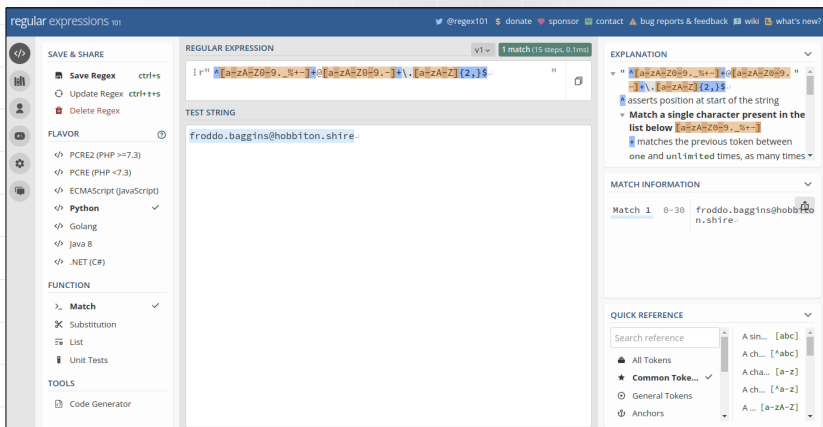

► Wyszukiwanie (`search`, `findall`)

```
1 >>> r1 = re.compile(r'box')
2 >>> r1.match('inbox')
3 >>> r1 = re.compile(r'box')
4 >>> r1.match('inbox')
5 >>> r1.search('inbox')
6 <re.Match object; span=(2, 5), match='box'>
7 >>> r2 = re.compile(r'\w{2}')
8 >>> r2.findall("Sprzedam opła.")
9 ['Sp', 'rz', 'ed', 'am', 'op', 'la']
```

Debugowanie wyrażeń regularnych

Regex 101

- ▶ <https://pythex.org/>
- ▶ <https://regex101.com>



The screenshot shows the regex101.com interface. The regular expression being tested is `! r" [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ "`. The test string is `frodo.baggins@hobbiton.shire.`. The interface shows a match for the entire string. The explanation section highlights the `[a-zA-Z0-9._%+-]+` part, stating it matches the previous token between one and unlimited times. The match information section shows the match starting at index 0 and ending at index 30. The quick reference section provides a search reference for the match.

regular expressions 101 twitter @regex101 donate sponsor contact bug reports & feedback wiki what's new?

SAVE & SHARE

- Save Regex `ctrl+s`
- Update Regex `ctrl+u+s`
- Delete Regex

FLAVOR

- PCRE2 (PHP >=7.3)
- PCRE (PHP <7.3)
- ECMAScript (JavaScript)
- Python ✓
- Golang
- Java 8
- .NET (C#)

FUNCTION

- Match ✓
- Substitution
- List
- Unit Tests

TOOLS

- Code Generator

REGULAR EXPRESSION `v1` `1 match (15 steps, 0.1ms)`

`! r" [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ "`

TEST STRING

`frodo.baggins@hobbiton.shire.`

EXPLANATION

- `[a-zA-Z0-9._%+-]+` asserts position at start of the string
- `[a-zA-Z0-9._%+-]+` Match a single character present in the list below `[a-zA-Z0-9._%+-]`
- `[a-zA-Z0-9._%+-]+` matches the previous token between one and unlimited times, as many times ✓

MATCH INFORMATION

Match 1 0-30 `frodo.baggins@hobbiton.shire.`

QUICK REFERENCE

Search reference

- All Tokens
- Common Tokens
- General Tokens
- anchors

A sin... [abc]
A ch... [^abc]
A cha... [a-z]
A ch... [^a-z]
A ... [a-zA-Z]

Interfejsy w trybie tekstowym

- ▶ interfejsy linii komend – Command-line interface (CLI)
 - ▶ Wprowadzanie poleceń z klawiatury
 - ▶ Łatwa automatyzacja
 - ▶ Pobieranie danych poprzez parametry linii komend
 - ▶ Narzędzia warte uwagi:
 - ▶ `argparse`
 - ▶ `click`
 - ▶ `docopt`
 - ▶ `python-fire`
- ▶ tekstowe interfejsy użytkownika – Text-based user interface (TUI)
 - ▶ Wykorzystanie trybu tekstowego (terminala) do symulowania elementów graficznych, takich jak okna, pola do wprowadzania, przyciski.
 - ▶ Narzędzia:
 - ▶ `ncurses` – wrapper na bibliotekę w C, trudne w użyciu
 - ▶ `prompt_toolkit`
 - ▶ `asciimatics`
 - ▶ `blessed`

argparse I

- moduł biblioteki standardowej dostarczający intuicyjny parser `sys.argv`

```
1 import argparse
2
3 parser = argparse.ArgumentParser(description='Przykładowy program z argumentami')
4
5 parser.add_argument('arg1', help='Pierwszy arg. pozycyjny')
6 parser.add_argument('--arg2', help='Drugi arg. opcjonalny')
7 parser.add_argument('--arg3', required=True, help='Trzeci arg. opcjonalny wymagany')
8 parser.add_argument('-v', '--verbose', action='store_true', help='Włącz tryb rozgadany')
9
10 args = parser.parse_args()
11
12 print(f'Argument 1: {args.arg1}')
13 if args.arg2:
14     print(f'Argument 2: {args.arg2}')
15 print(f'Argument 3: {args.arg3}')
16
17 if args.verbose:
18     print('Tryb rozgadany włączony')
```

- wywołanie

```
1 python program.py argument --arg2 value --arg3 required_value -v
```

► Podkomendy

```
1 import argparse
2
3 # Tworzenie parsera argumentów
4 parser = argparse.ArgumentParser(description='Przykładowy program z podkomendami')
5
6 subparser1 = parser.add_subparsers(title='subcommand1', description='Podkomenda 1',
7                                   dest='subcommand1')
8
9 subparser1.add_parser('cmd1', help='Polecenie 1')
10
11 subparser2 = parser.add_subparsers(title='subcommand2', description='Podkomenda 2',
12                                   dest='subcommand2')
13
14 parser_cmd2 = subparser2.add_parser('cmd2', help='Polecenie 2')
15 parser_cmd2.add_argument('--arg', help='Opcjonalny argument')
16
17 # Parsowanie argumentów
18 args = parser.parse_args()
```

```
1 python program.py subcommand2 cmd2 --arg value
```

docopt 1

- ▶ **moduł zewnętrzny**, pip install docopt
- ▶ Tworzenie interfejsu polega na napisaniu dokumentacyjnego łańcucha, który jest parsowany przez **docopt** i pozwala na wygenerowanie struktury słownikowej.

```
1 """Naval Fate.
2
3 Usage:
4     naval_fate.py ship new <name>...
5     naval_fate.py ship <name> move <x> <y> [--speed=<kn>]
6     naval_fate.py ship shoot <x> <y>
7     naval_fate.py mine (set|remove) <x> <y> [--moored | --drifting]
8     naval_fate.py (-h | --help)
9     naval_fate.py --version
10
11 Options:
12     -h --help    Show this screen.
13     --version    Show version.
14     --speed=<kn> Speed in knots [default: 10].
15     --moored     Moored (anchored) mine.
16     --drifting   Drifting mine.
17
18 """
19 from docopt import docopt
20
```

```
21  
22 if __name__ == '__main__':  
23     arguments = docopt(__doc__, version='Naval Fate 2.0')  
24     print(arguments)
```

► Wywołanie

```
1 python naval_fate.py ship my_ship move 10 20 --speed=1024  
2     {'--drifting': False,  
3      '--help': False,  
4      '--moored': False,  
5      '--speed': '1024',  
6      '--version': False,  
7      '<name>': ['my_ship'],  
8      '<x>': '10',  
9      '<y>': '20',  
10     'mine': False,  
11     'move': True,  
12     'new': False,  
13     'remove': False,  
14     'set': False,  
15     'ship': True,  
16     'shoot': False}
```

click I

- ▶ **moduł zewnętrzny**, pip install click
- ▶ Tworzenie CLI w stylu dekorowania elementów kodu.
- ▶ Nie tylko parsing, także zarządzanie przekierowaniem zebranych danych do odpowiednich funkcji.

```
1 import click
2
3 @click.group()
4 def cli():
5     pass
6
7 @cli.command()
8 @click.option('--greeting', default='Hello', help='The greeting to use.')
9 @click.option('--uppercase/--no-uppercase', default=False, help='Convert name to uppercase.')
10 @click.argument('name')
11 def greet(name, greeting, uppercase):
12     """Greet NAME."""
13     if uppercase:
14         name = name.upper()
15     click.echo(f'{greeting}, {name}!')
16
17 @cli.command()
18 @click.option('--farewell', default='Goodbye', help='The farewell to use.')
19 @click.option('--uppercase/--no-uppercase', default=False, help='Convert name to uppercase.')
20 @click.argument('name')
21 @click.argument('count', type=int)
```


click II

```
22 def goodbye(name, count, farewell, uppercase):
23     """Say goodbye to NAME COUNT times."""
24     if uppercase:
25         name = name.upper()
26         for i in range(count):
27             click.echo(f'{farewell}, {name}!')
28
29 if __name__ == '__main__':
30     cli()
```

► Wywołanie

```
1 python cli.py greet
2 Usage: cli.py greet [OPTIONS] NAME
3 Try 'cli.py greet --help' for help.
4
5 Error: Missing argument 'NAME'.
```

Tekstowe interfejsy użytkownika (TUI) I

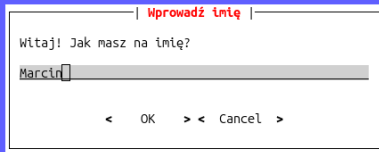
Przykład: prompt_toolkit

- ▶ Zaprojektowany jako narzędzie do ułatwiania wprowadzania danych, rozwinął się do złożonego, wysokopoziomowego systemu tworzenia aplikacji w trybie tekstowym.
- ▶ Dokumentacja modułu
- ▶ Przykład użycia

```
1 from prompt_toolkit.shortcuts import button_dialog, input_dialog
2
3 def greet():
4     name = input_dialog(
5         title='Wprowadź imię',
6         text='Witaj! Jak masz na imię?'
7     ).run()
8
9     button_dialog(
10        title='Hello',
11        text=f'Hello, {name}!',
12        buttons=[('OK', None)],
13    ).run()
14
15
16 if __name__ == '__main__':
17     greet()
```

Tekstowe interfejsy użytkownika (TUI) II

Przykład: prompt_toolkit



Wprowadź imię

Witaj! Jak masz na imię?

Marcin

< OK > < Cancel >

Tekstowe interfejsy użytkownika (TUI) III

Przykład: prompt_toolkit



Praca z plikami konfiguracyjnymi I

Moduł `configparser`

- ▶ Moduł `ConfigParser` dostarcza narzędzi do pracy z prostymi plikami konfiguracyjnymi (podobnymi do *.ini)
- ▶ Struktura pliku – sekcje z parami klucz-wartość

```
1 [general]
2 debug = true
3 log_file = /var/log/myapp.log
4
5 [database]
6 host = localhost
7 port = 5432
8 name = my_database
9 user = my_user
10 password = my_password
```

Praca z plikami konfiguracyjnymi II

Moduł configparser

- ▶ obiekty `ConfigParser()` pozwalają czytać konfigurację z pliku

```
1 import configparser
2 config = configparser.ConfigParser()
3 config.read('config.ini')
4
5 debug = config.getboolean('general', 'debug')
6 log_file = config.get('general', 'log_file')
7
8 db_name = config.get('database', 'name')
```

- ▶ Wartości domyślne

```
1 >>> config.get("LogLevel", "DEBUG")
2 >>> "DEBUG"
```

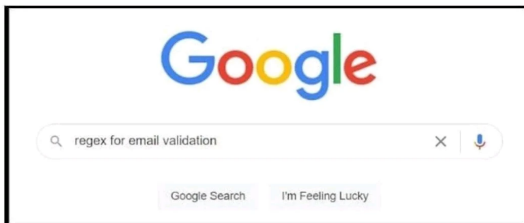
Inne przydatne moduły biblioteki standardowej Pythona

- ▶ `math` – moduł zawierający stałe i funkcje matematyczne
- ▶ `cmath` – funkcje matematyczne dla liczb zespolonych
- ▶ `fraction` – operacje na ułamkach
- ▶ `decimal` – szybkie, dobrze zaokrąglane operacje na liczbach rzeczywistych
- ▶ `random` – moduł do generowania liczb losowych
- ▶ `statistics` – powszechnie używane statystyki
- ▶ `pprint` – czytelne wypisywanie strukturdanych
- ▶ `getpass` – pobierania hasła od użytkownika bez wyświetlania
- ▶ ...
- ▶ ...

Więcej w [The Python Standard Library](#)

To już wszystko na dziś!

DAY1 OF PROGRAMMING



10 YEARS OF PROGRAMMING

