giganci programowania

Regex - wyrażenia regularne

Wyrażenia regularne

Wyrażenia regularne (regex) to specjalne wzorce, które umożliwiają przeszukiwanie, modyfikowanie oraz analizowanie tekstu na podstawie zadanych reguł.

Zastosowania wyrażeń regularnych

- Wyszukiwanie tekstu: Regex pozwala na znalezienie fragmentów tekstu, które spełniają określony wzorzec, np. wszystkie adresy e-mail w tekście.
- Weryfikacja formatu danych: Można za pomocą regex sprawdzić, czy dany ciąg znaków pasuje do określonego formatu, np. czy numer telefonu lub data są poprawne.
- Zamiana tekstu: Regex ułatwia zamianę określonych fragmentów tekstu na inne, np. zamiana wszystkich adresów URL na odnośnik "LINK".
- Parsowanie i ekstrakcja danych: Regex umożliwia wydobywanie konkretnych fragmentów tekstu, takich jak numery telefonów, adresy IP czy kody pocztowe.
- Podział tekstu: Regex pozwala na dzielenie tekstu na części na podstawie wyrażeń, np. podział tekstu według interpunkcji.

Biblioteka re w Pythonie

re to standardowa biblioteka w Pythonie służąca do pracy z wyrażeniami regularnymi (regex). Wyrażenia regularne to narzędzie do przeszukiwania, dopasowywania i manipulacji tekstu na podstawie zdefiniowanych wzorców. Biblioteka re oferuje zestaw funkcji i metod, które pozwalają na wykonywanie takich operacji jak wyszukiwanie wzorców w tekście, ich zamiana, czy podział tekstu na części.

Wzorzec

- **Wzorzec** (ang. *pattern*) to specjalny ciąg znaków, który definiuje reguły wyszukiwania i dopasowywania fragmentów tekstu.

- **Budowa wzorca**
  Wzorzec jest tworzony za pomocą specjalnych symboli, znaków i metaznaków, które określają, co powinno być dopasowane w tekście. Może być to np. konkretny ciąg znaków, pewien zakres liter lub cyfr, liczba powtórzeń określonego fragmentu czy dowolny znak. Wzorzec definiuje, jakiego rodzaju dane chcemy znaleźć w tekście.

Kluczowe elementy wzorców

Klasa znaków definiuje zbiór znaków, spośród których każdy pojedynczy znak może być dopasowany.

- [a-z] - małe litery.
- [A-Z] - wielkie litery.
- [a-zA-Z] - małe i wielkie litery.
- [0-9] - cyfry.
- [abc] - litery “a”, “b”, “c”

Kluczowe elementy wzorców

Metaznaki mają specjalne znaczenie w wyrażeniach regularnych i są używane do określania bardziej złożonych wzorców. Oto kilka często stosowanych metaznaków:

- . (kropka) - dopasowuje dowolny znak z wyjątkiem znaku nowej linii.
- \d - dopasowuje dowolną cyfrę (0-9).
- \w - dopasowuje dowolny znak alfanumeryczny (litera, cyfra lub znak podkreślenia).
- \s - dopasowuje dowolny znak biały (spacja, tabulator, nowa linia).
- \b - dopasowuje granicę słowa
- ^ - dopasowuje początek ciągu lub linii.
- $ - dopasowuje koniec ciągu lub linii.

Kluczowe elementy wzorców

Kwantyfikatory pozwalają określić, ile razy dany fragment wzorca może wystąpić:

- * - dopasowuje zero lub więcej wystąpień poprzedniego znaku lub wyrażenia.
- + - dopasowuje jedno lub więcej wystąpień.
- ? - dopasowuje zero lub jedno wystąpienie.
- {m,n} - dopasowuje od m do n wystąpień.

Najpopularniejsze funkcje biblioteki re

- **re.match(pattern, string)**: Sprawdza, czy wzorzec pasuje na początku tekstu. Zwraca obiekt Match lub None.

```
sentence = "Ala ma kota"
result = re.match(r"Ala", "Ala ma kota")
print(f"Wyszukiwanie elementu Ala {result}")
```

- **re.search(pattern, string)**: Szuka pierwszego wystąpienia wzorca w tekście. Zwraca obiekt Match lub None jeśli nie znajdzie pasującego fragmentu.

```
napis = "Ala ma 123 jabłka"
result = re.search(r"\d+", napis)
print(f"Wyszukiwanie liczby zakończone, wykryto: {result}")
```

Najpopularniejsze funkcje biblioteki re

- **re.findall(pattern, string)**: Zwraca listę wszystkich fragmentów tekstu, które pasują do wzorca.

```
napis = "Ala i Kuba jadą na wycieCzkę do Warszawy"
result = re.findall(r"\b[A-Z][a-z]+", napis)
print(f"Wyszukiwanie wyrazów zaczynających się wielką literą zakończone, wykryto: {result}")
```

- **re.sub(pattern, repl, string)**: Zastępuje wszystkie wystąpienia wzorca w tekście na podany ciąg znaków repl.

```
text = "Valorant to najlepsza strzelanka na świecie"
pattern = "Valorant"
replacement = "CS"
new_text = re.sub(pattern, replacement, text)
print(new_text)
```