Wyrażenia regularne (regex)

- „sprytny filtr” / „wzorzec”, który mówi komputerowi:
„znajdź mi w tekście rzeczy, które wyglądają tak i tak”.

- oprócz dokładnego słowa, regex potrafi szukać **formatu**:
np. maili, numerów telefonu, kodów pocztowych, IP, dat, tagów HTML itd.

---

1. Praktyczne zastosowania

- **Wyszukiwanie fragmentów tekstu**
np. „znajdź wszystkie e-maile w wiadomości”.

- **Weryfikacja formatu (walidacja)**
np. „czy to wygląda jak poprawny numer telefonu / kod pocztowy / data”.

- **Zamiana tekstu**
np. „zamień wszystkie linki na [LINK]”.

- **Ekstrakcja danych (wyciąganie)**
np. „wyciągnij z tekstu IP i porty”.

- **Dzielenie tekstu (split)**
np. „podziel zdanie na części po przecinkach / kropkach”.

---

2. Python: biblioteka re (regex w praktyce)

- W Pythonie regex obsługuje standardowa biblioteka **re**.
- Daje funkcje do: szukania, dopasowania, wyciągania, podmiany, dzielenia.

Najważniejsze funkcje

- `re.search(pattern, text)`
szuka **pierwszego dopasowania gdziekolwiek** w tekście.

- `re.match(pattern, text)`
sprawdza dopasowanie **tylko od początku** tekstu.

- `re.findall(pattern, text)`
zwraca **listę wszystkich dopasowań**.

- `re.sub(pattern, repl, text)`
zamienia dopasowane fragmenty na repl.

- `re.split(pattern, text)`
dzieli tekst według wzorca.

Wzorzec

- **Pattern** to przepis: „jak ma wyglądać to, czego szukam”.

  - „mail” = **litery/cyfry + @ + domena + kropka + końcówka**
  - „IP” = **liczby + kropki + liczby + kropki + liczby + kropki + liczby**

Najważniejsze elementy regex (ściąga)

1) Zwykłe znaki

- `kot` dopasuje dokładnie: `"kot"`

2) Metaznaki (te najczęstsze)

- `.` — dowolny znak (poza enterem)
- `\d` — cyfra `0-9`
- `\w` — znak „słowowy” (litera/cyfra/_)
- `\s` — biały znak (spacja, tab, nowa linia)
- `\b…\b` — granica słowa (np. żeby `kot` nie łapał `kotlet`)
- `^` — początek tekstu/linii
- `$` — koniec tekstu/linii

3) Kwantyfikatory (ile razy ma się powtórzyć)

- `*` — zero lub więcej
- `+` — jeden lub więcej
- `?` — zero lub jeden
- `{m,n}` — od m do n razy (np. `{2,4}`)

Przykłady:

- `\d+` → „ciąg cyfr” (np. `12345`)
- `\d{2}` → dokładnie 2 cyfry (np. `07`)
- `\d{2,4}` → 2–4 cyfry (np. `12` albo `2024`)

4) Klasy znaków [...]

- `[abc]` — a lub b lub c
- `[a-z]` — małe litery
- `[0-9]` — cyfry
- `[^0-9]` — „wszystko oprócz cyfr” (daszek w środku = negacja)

5) Grupowanie i „albo”

- `( ... )` — grupa (traktujemy fragment jako całość)
- `|` — alternatywa (albo to, albo to)

Przykład:

- `(kot|pies)` dopasuje „kot” albo „pies”

6) Escape (uciekanie znaków specjalnych)

- Niektóre znaki mają znaczenie specjalne, np. `. ? ( ) [ ]`

Jeśli chcesz kropkę „dosłownie”, piszesz: `\.`

Flagi (tryby działania) — 4 najważniejsze

🟥 `re.IGNORECASE` / `re.I`
ignoruje wielkość liter (`abc` pasuje też do `ABC`)

🟥 `re.MULTILINE` / `re.M`
`^` i `$` działają na **każdą linię**, nie tylko cały tekst

🟥 `re.DOTALL` / `re.S`
kropka `.` dopasowuje też znak nowej linii

🟥 `re.VERBOSE` / `re.X`
pozwala pisać regex „ładnie”, z odstępami i komentarzami (czytelniej)

🟥 `re.ASCII` / `re.A`
wymusza tryb **ASCII** dla skrótów typu `\w`, `\d`, `\s` (np. `\w` dopasuje tylko a–z, A–Z, 0–9 i _, bez polskich znaków typu *ą*, *ł*)

Mini-przykłady

🔶 **Znajdź wszystkie liczby w tekście**
Wzorzec: `\d+`
Sens: „jeden lub więcej cyfr”

🔶 **Sprawdź czy tekst wygląda jak kod pocztowy PL `12-345`**
Wzorzec: `^\d{2}-\d{3}$`
Sens: start → 2 cyfry → myślnik → 3 cyfry → koniec

🔶 **Zamień wszystkie linki na `[LINK]`**
Wzorzec: `https?://\S+`
Sens: `http://` albo `https://`, potem ciąg znaków bez spacji

🔶 **Wyciągnij wszystkie adresy IP (prosto, „formatowo”)**
Wzorzec: `\b\d{1,3}(\.\d{1,3}){3}\b`
Sens: 1–3 cyfry i jeszcze 3 razy: kropka + 1–3 cyfry