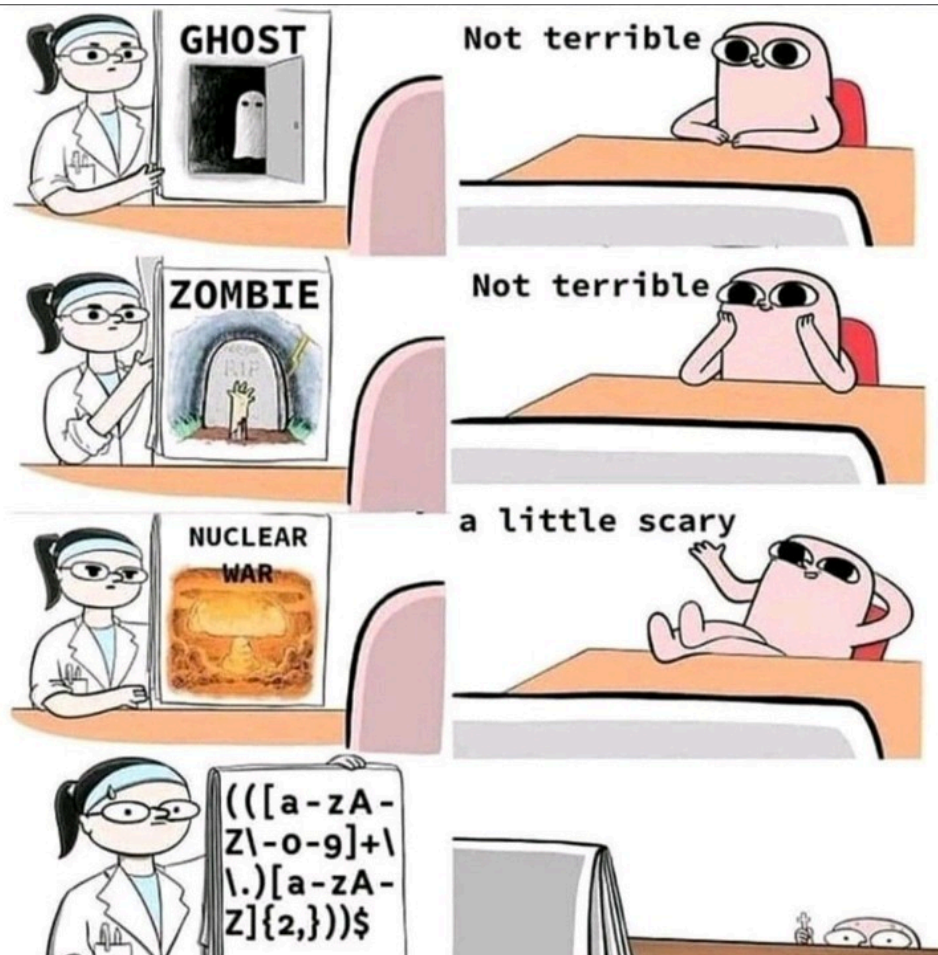


Regulární výrazy

PVA2 Programování a vývoj aplikací

Obsah

1. Obsah
2. Co jsou to regulární výrazy?
3. Použití regulárních výrazů
4. Příklady použití
5. Pomocníci
6. Základní syntaxe
7. Kvantifikátory
8. Zástupné znaky
9. Sekvence znaků
10. Skupiny znaků
11. Negace
12. Logický OR (Alternativa)
13. Regex prakticky
14. Postup při tvorbě regulárního výrazu
15. Vizualizace regulárních výrazů
16. Časté chyby při tvorbě regexů
17. Regex v Pythonu
18. Regulární výrazy v Pythonu
19. Příklady použití v Pythonu
20. Příklad `re.sub()`
21. `re.fullmatch()`
22. `re.search()`
23. Word Boundaries



Co jsou to regulární výrazy?

- Regulární výrazy jsou způsob, jak popsat vzory v textu
- Konstrukce vychází z teorie formálních jazyků a překladačů.
- Jsou to řetězce, které popisují, jaký text hledáme
- Používají se v různých programovacích jazycích a nástrojích
- Například v JavaScriptu, Pythonu, PHP, nebo v textových editorech
- Základní syntax stejná ve většině jazycích, ale mohou se lišit v detailu
- Jsou velmi mocným nástrojem pro práci s textem
- Jsou také velmi těžké na pochopení a psaní
- Většina programátorů je neumí správně používat

Použití regulárních výrazů

- Vyhledávání textu
- Nahrazování textu
- Validace vstupů
 - Například validace emailu, hesla, telefonního čísla, atd.
- Extrahování dat
- Filtraci dat

Příklady použití

- Vyhledání všech emailů v textu
- Vyhledání všech URL v textu
- Vyhledání všech telefonních čísel v textu
- Kontrola, zda je heslo dostatečně silné
- Kontrola, zda je email validní
- Kontrola, zda je telefonní číslo validní

Pomocníci

- Regex101
- RegExr

Základní syntaxe

- V syntaxi regulárních výrazů má každý znak svůj speciální význam.
- Základní znaky se hledají přesně tak, jak jsou napsány.
- Kvantifikátory: `*` `+` `?` `{n}` `{n,}` `{n,m}`
- Skupiny: `()` `[]`
- Rozsahy: `a-z` `A-Z` `0-9`
- Speciální znaky: `.` `^` `$` `*` `+` `?` `\` `|`

Kvantifikátory

- Znak kvantifikátoru určuje, kolikrát se má znak před ním opakovat.
- Kvantifikátory se vždy vztahují k jednomu znaku nebo skupině znaků.
- `*` - nula nebo více opakování
- `+` - jedno nebo více opakování
- `?` - nula nebo jedno opakování
- `{n}` - přesně n opakování
- `{n,}` - n nebo více opakování
- `{n,m}` - od n do m opakování
- `*?` - hledá co nejmenší možné opakování

Příklady

```
1  a* - a, aa, aaa, aaaa, ...
2  a+ - aa, aaa, aaaa, ...
3  a? - a, aa
4  a{3} - aaa
5  a{3,} - aaa, aaaa, aaaaa, ...
6  a{3,5} - aaa, aaaa, aaaaa
7  a*? - a, aa, aaa, aaaa, ...
```

Zástupné znaky

- Zástupné znaky se používají pro hledání libovolného znaku nebo skupiny znaků.
- `.` - libovolný znak
- `[]` - libovolný znak z dané množiny
- `[^]` - libovolný znak, který není v dané množině
- `()` - skupina znaků
- `|` - logický OR, pravý `Alt` + `w`
- `\` - escape znak
- `^` - začátek řádku, pravý `alt` + `3`
- `$` - konec řádku, pravý `alt` + `u`

Příklady

```
1  a.c - abc, acc, a1c, a#c, ...
2  [a-z] - libovolný znak z a-z
3  [^a-z] - libovolný znak, který není z a-z
4  ^a - začíná na a
5  a$ - končí na a
6  ^a.*b$ - řádek začíná na a a končí na b
7  [a-z0-9] - libovolný znak z a-z nebo 0-9
8  [a-z]{3} - tři znaky z a-z
9  [a-z]{3,5} - tři až pět znaků z a-z
10 a|b - buď znak a nebo znak b
```

Sekvence znaků

- Sekvence znaků se používají pro hledání konkrétního řetězce znaků.
- Sekvence znaků se hledají přesně tak, jak jsou napsány.

```
1 abc - abc
2 abc{3} - abccc
3 abc{3,5} - abccc, abccccc, abccccc
```

Množiny znaků

- 1 `[abc]` - a nebo b nebo c
- 2 `[a-z]` - libovolný znak z a-z
- 3 `[A-Z]` - libovolný znak z A-Z
- 4 `[0-9]` - libovolný znak z 0-9
- 5 `[a-zA-Z]` - libovolný znak z a-z nebo A-Z
- 6 `[a-zA-Z0-9]` - libovolný znak z a-z, A-Z nebo 0-9

- 1 `\d` - libovolné číslo (0-9) `[0-9]`
- 2 `\D` - libovolný znak, který není číslo (0-9) `[^0-9]`
- 3 `\w` - libovolné písmeno nebo číslo (a-z, A-Z, 0-9)
- 4 `\W` - libovolný znak, který není písmeno nebo číslo (`^a-z, A-Z, 0-9`)
- 5 `\s` - libovolný bílý znak (mezera, tabulátor, nový řádek)
- 6 `\S` - libovolný znak, který není bílý znak

Skupiny znaků

- Skupiny znaků se používají pro hledání více znaků najednou.
- Skupiny se vždy uzavírají do závorek `()`.
- Skupiny mohou obsahovat libovolné znaky nebo zástupné znaky.

```
1 (a|b) - a nebo b
2 (a|b|c) - a nebo b nebo c
3 (a|b|c){3} - tři znaky a, b nebo c
4 (a|b|c){3,5} - tři až pět znaků a, b nebo c
5 ([A-E]|[0-5]){1} - buď znak z A-E nebo z 0-5
6 ([a-z]|[^0-9]){3} - tři znaky z a-z nebo znaky, které nejsou z 0-9
```

Negace

- Negace se používá pro hledání znaků, které nejsou v dané množině.
- Negace se zapisuje pomocí `^` na začátku množiny znaků.

```
1  [^a-z] - libovolný znak, který není z a-z  
2  [^0-9] - libovolný znak, který není z 0-9
```

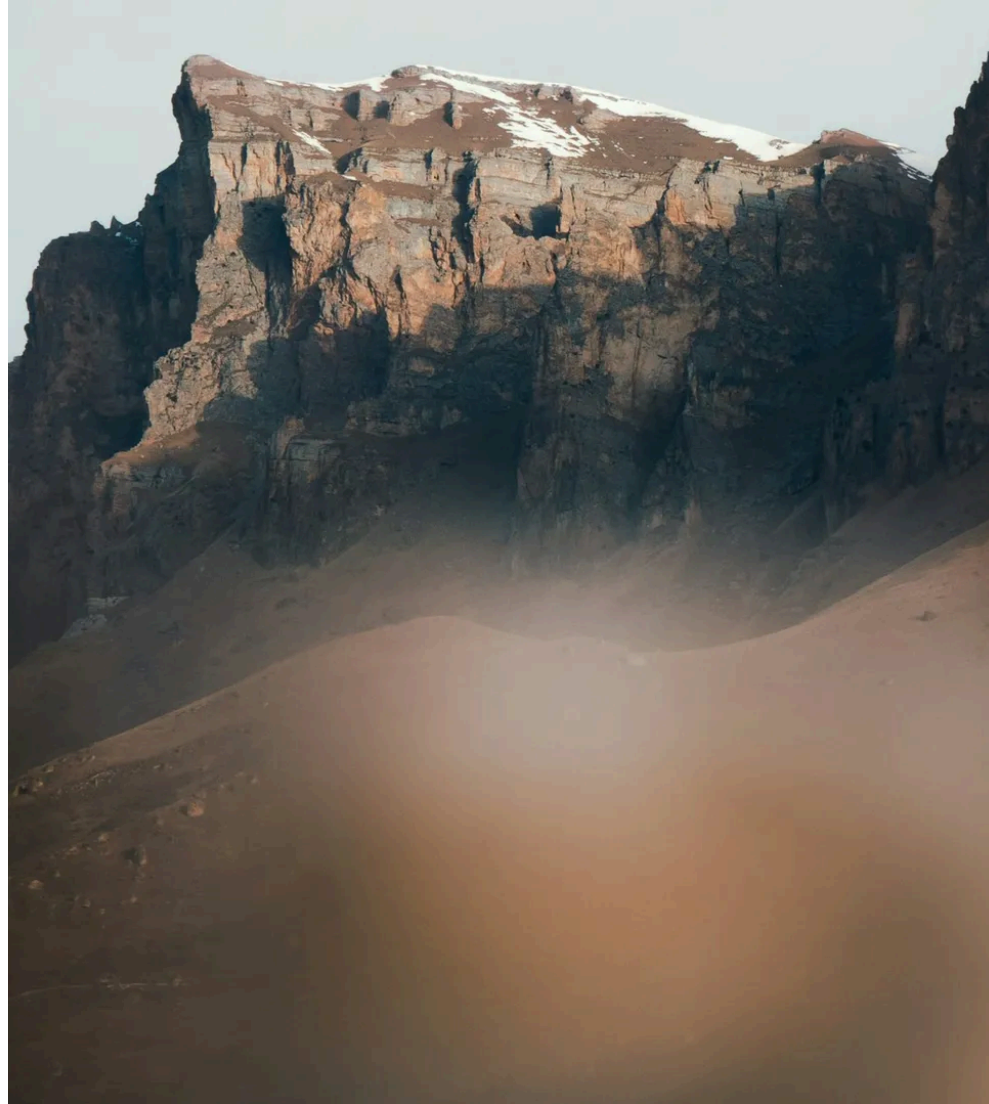
Pozor! Negace se používá pouze uvnitř množiny znaků `[]`. Neplést s začínáním řádku `^`.

Logický OR (Alternativa)

- Alternativa se používá pro hledání více možností najednou.
- Zapisujeme pomocí svislítky `|` mezi možnostmi.

```
1  a|b - buď znak a nebo znak b
2  a|A - buď znak a nebo znak A
3  abc|def - buď abc nebo def
4  [a-z]|[^0-9] - buď znak z a-z nebo znak, který není z 0-9
```

Regex prakticky



Postup při tvorbě regulárního výrazu

1. Definujte, co hledáte.

- Např. "Hledám všechna slova, která začínají na `a` a končí na `z`."

2. Rozdělte problém na části.

- Hledání slova: `[a-zA-Z]+`
- Začíná na `a` : `^a`
- Končí na `z` : `z$`

3. Postupně testujte jednotlivé části.

- Začněte s jednoduchým výrazem a postupně přidávejte další pravidla.

4. Použijte nástroje pro testování.

- Doporučené nástroje: Regex101 nebo RegExr.

Vizualizace regulárních výrazů

Např. pro regulární výraz `^a.*z$` :

1. `^` : Začátek řádku.
2. `a` : První znak musí být `a` .
3. `.*` : Jakékoliv znaky (nula nebo více opakování).
4. `z$` : Poslední znak musí být `z` .

::right::

Příklad

1 `^a.*z$`

Text	Výsledek
abc	x

Časté chyby při tvorbě regexů

1. Zapomenutý escape znak:

- Např. hledáte tečku (`.`), ale bez escapování se jedná o zástupný znak.
- Řešení: Použijte `\.`.

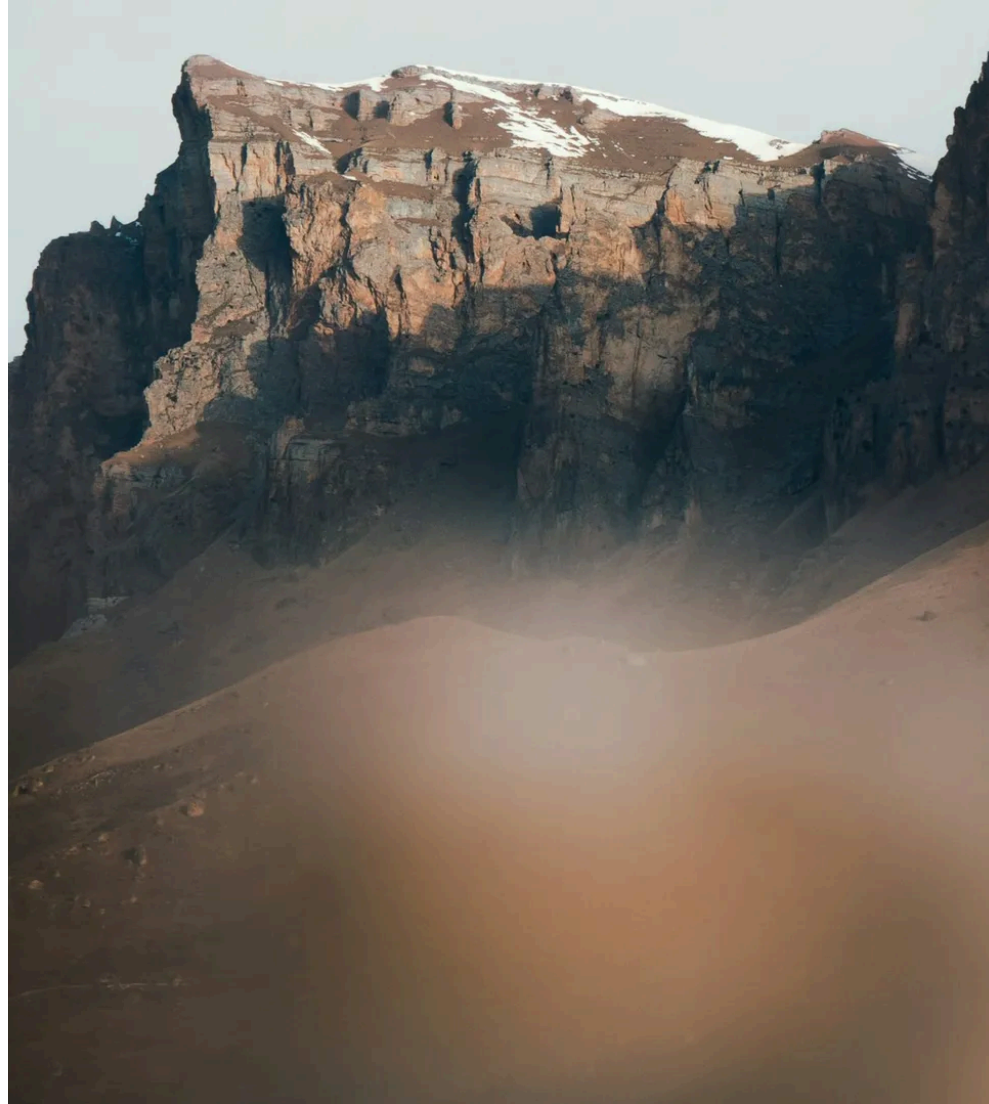
2. Příliš obecný výraz:

- Např. `.*` může zachytit více, než potřebujete.
- Řešení: Zúžit pomocí kvantifikátorů, např. `{1, 5}`.

3. Nepoužité kotvy (`^` , `$`):

- Pokud nehledáte od začátku nebo do konce řádku, může dojít k nechtěným výsledkům.
- Řešení: Použít kotvy tam, kde jsou nutné.

Regex v Pythonu



Regulární výrazy v Pythonu

- Python má zabudovanou knihovnu `re` pro práci s regulárními výrazy.
- Knihovna obsahuje několik základních funkcí pro práci s regulárními výrazy.
- Regulární výrazy se vždy zapisují jako raw stringy (označené předponou `r`).
- `re._metoda_( _pattern_, _zpracováváný text_ )`
- Nejčastěji používané metody:
 - `re.search()`
 - `re.sub()`
 - `re.match()`
 - `re.findall()`

Příklady použití v Pythonu

```
1  import re
2
3  # Hledání textu
4  re.search(r'abc', 'abcde') # <re.Match object; span=(0, 3), match='abc'>
5
6  # Nahrazení textu
7  re.sub(r'abc', 'def', 'abcde') # 'defde'
8
9  # Validace vstupu
10 re.match(r'[a-z]+', 'abc') # <re.Match object; span=(0, 3), match='abc'>
11
12 # Extrahování dat
13 re.findall(r'[a-z]+', 'abc def ghi') # ['abc', 'def', 'ghi']
14
15 # Filtraci dat
16 re.findall(r'[a-z]{3}', 'abc def ghi') # ['abc', 'def', 'ghi']
```

Příklad `re.sub()`

```
1  import re
2
3  # Pro pattern vždy používejte raw stringy (označené předponou r),
4  # abyste se vyhnuli escapování zpětným lomítkem.
5
6  text = "Python je skvělý jazyk. Python je velmi populární."
7  pattern = r'Python'
8  replacement = "PHP"
9
10 new_text = re.sub(pattern, replacement, text)
11 print(new_text) # "PHP je skvělý jazyk. PHP je velmi populární."
```

re.fullmatch()

```
1 import re
2
3 # regularni vyraz
4 regex = r"[a-zA-Z]{2,10}"
5
6 text = "Jmeno"
7
8 if(re.fullmatch(regex, text)):
9     print("Splňuje")
10 else:
11     print("Nesplňuje")
```


re.search()

```
1  import re
2
3  texty = ["ps","pes","pse","poe","prase","poklice"]
4  vyrazy = [ r"p[ars]e", r"p[ars]*e", r"p[^ars]e" ]
5
6  for text in texty:
7      for vyraz in vyrazy:
8          if re.search(vyraz, text):
9              vysledek = "ano"
10         else:
11             vysledek = "ne"
12
13     print(text + " splňuje " + vyraz + ": " + vysledek)
```

Word Boundaries

- Word boundaries se používají pro hledání slov.
- zapisují se pomocí `\b` na začátku nebo na konci slova.
- Když je umístěno mezi dvěma znaky, kontroluje, zda se nachází na hranici slova (tj. mezi slovem a ne-slovem).

```
1  \babc\b - abc
2  \babc\b - abc, abcde, abcdef, ...
```

```
1  # Najde slovo "python" pouze jako samostatné slovo,
2  # nikoli jako součást jiného slova (např. "pythonic").
3  r"\bpython\b"
4
5  # Najde třímístná čísla jako samostatná slova (např. "123", ale ne "1234").
6  # Oba zápisy jsou ekvivalentní
7  r"\b\d{3}\b"
8  r"\b[0-9]{3}\b"
```

Děkuji za pozornost

Otázky?

Repository / Prezentace