EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

jihomoravský kraj

SPRÁVA A DOHLED NAD POČÍTAČOVOU SÍTÍ

Regex

Metodický list

Autor: Giovanni Guadagno, Metodik: Bc. Jaroslav Tihlařík

Recenzenti: doc. Ing. Jaroslav Dočkal, CSc.

Rok vydání: 2023

Obsah

Cíle

- Student dokáže za pomocí otevřených zdrojů vytvořit Regex

Dovednosti

- Student analyzuje vzorek, na který má být regex aplikován
- Dle formátu vzorku vymyslí a napíše regex
- Regex je schopen aplikovat v praxi
- Student je schopen Regex debuggovat, hledat v něm chyby

Pracovní prostředí

Úlohu lze realizovat v prostředí Cylab JCEKB

Pro práci budeme potřebovat následující:

- Zdroj řetězců, pro které se bude dělat Regex
- Python3 – knihovna re
- Server regex101.com nebo obdobný
- Různé libovolné otevřené zdroje
- Součástí úlohy NENÍ výuková prezentace – je vhodné provádět výklad přímo v Regex editoru

Průběh výuky

1 Příprava

```
\s          Match whitespace
\S          Match non-whitespace
\d          Match číslice
\D          Match všeho kromě číslice
\w          Match písmen a číslic
\W          Match všeho kromě písmen a číslic
[xyz]       Match znaků x, y, z (Boolean OR)
[^xyz]      Match všeho kromě x, y, z (Boolean OR)
[A-Z]       Match znaků v rozmezí A-Z (case sensitive)
[^a-z]      Match všeho kromě znaků v rozmezí A-Z (case sensitive)
.           Match libovolného znaku
..          Match libovolných dvou znaků
?           Match nulakrát nebo jednou (např. \w?)
*           Match nulakrát nebo vícekrát
+           Match jednou nebo vícekrát
{5}         Match přesně pětkrát (např. \d{5})
{1,3}       Match jednou až třikrát (např. \{1,3} - vhodné pro IP adresy)
^           Začátek stringu
$           Konec stringu
(...)       Match & capture (z chytnutého řetězce se stane token, možno dále pracovat)
(?:...)     Match, ale ne capture (token se nevytvoří) - non-capturing token
a|b         Match a anebo b (není klasický Boolean OR)
\           Escape(např. \( bude se závorkou pracovat jako s jiným znakem)
```

Výše je „Regex cheat sheet“. Dále například služba regex101.com nebo obdobná pro tvorbu Regexu. Jako poslední nachystat zdroje.

2 První Regex

2.1 Doména

Zkusme nejprve udělat Regex pro doménu naší školy – cichnovabrno.cz. Víme, že je složena ze dvou řetězců složených z písmen a oddělených tečkou.

```
cichnovabrno.cz
\w+\.\w+
```

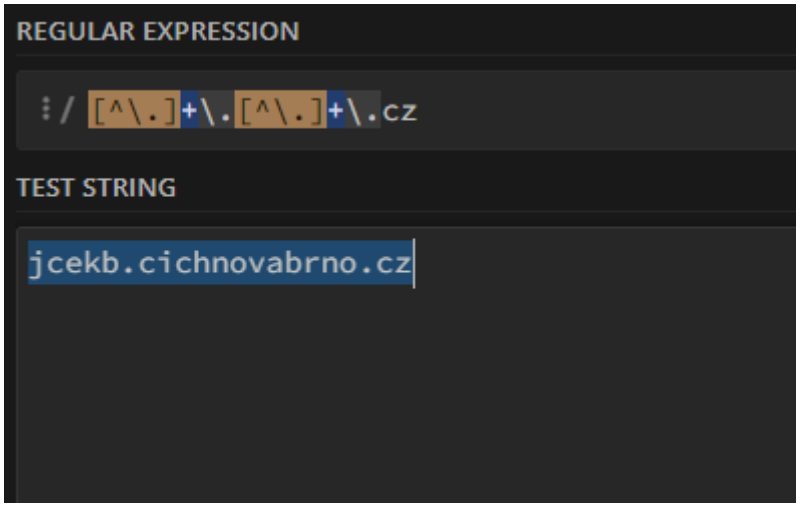
Takový Regex je zcela možno použít. Ale co když se v názvu objeví pomlčka? Zkusme na to jít jinak; předpokládajme, že aktuálně nemáme žádnou doménu nižšího řádu a vždy bude česká, a tak bude adresa vždy ve tvaru *něco* tečka *cz*. Použijme následující Regex:

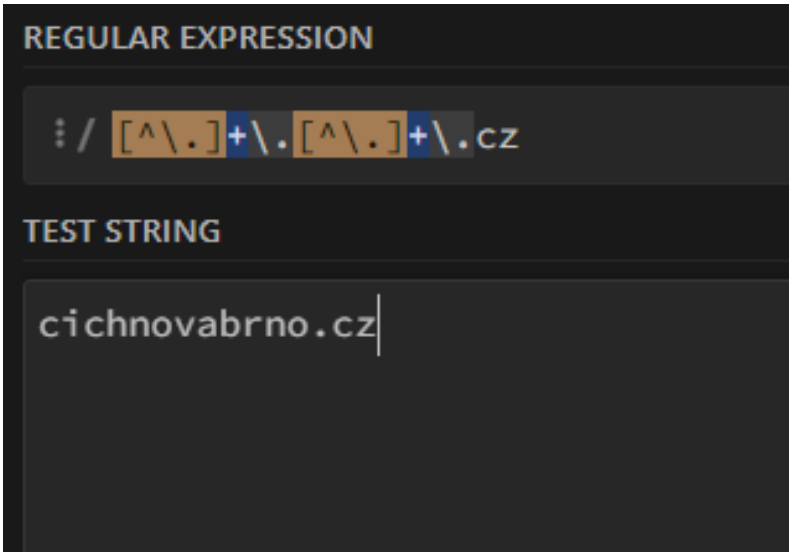
```
cichnovabrno.cz
[\.]+\.cz
```

Tento regex bude chytat vše, než narazí na tečku. Následně musí následovat tečka a za ní řetězec cz. Předpokládejme nyní, že budeme mít jednu subdoménu – vytvořme regex i pro tento případ

```
jcekb.cichnovabrno.cz
[^\.]+\.[^\.]+\.cz
```

To nebylo zas tak těžké – Regex matchuje do první tečky, následuje první tečka, potom match do druhé tečky, druhá tečka a následuje cz. Ale co když bude subdoména absentovat? V tom případě by nám výše uvedený Regex nefungoval

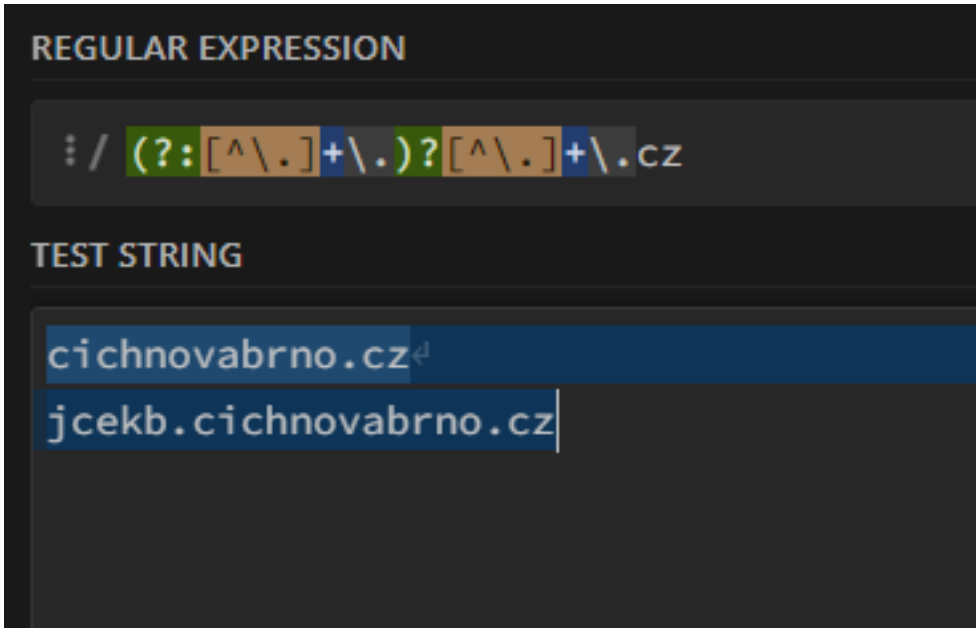




Musíme tedy zajistit, že subdoména někdy může absentovat – půjdeme na to přes kvantifikátor „nula nebo jedna“ – otazník. A vzhledem k tomu, že zatím s daty nepracujeme (group – do závorek), ponecháme subdoménu jako non-group token s kvantifikátorem 0 nebo 1.

```
jcekb.cichnovabrno.cz
(?:[^\.]+\.)?[^\.]+\.cz
```

Jak vidíme na obrázku níže, Regex matchuje obě varianty – jak se subdoménou jcekb, tak i bez ní

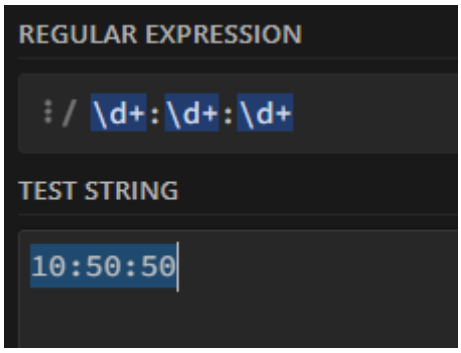


2.2 Čas

Předpokládejme, že máme čas ve formátu hh:mm:ss a máme jej matchnout. To se jeví jako poměrně jednoduché:

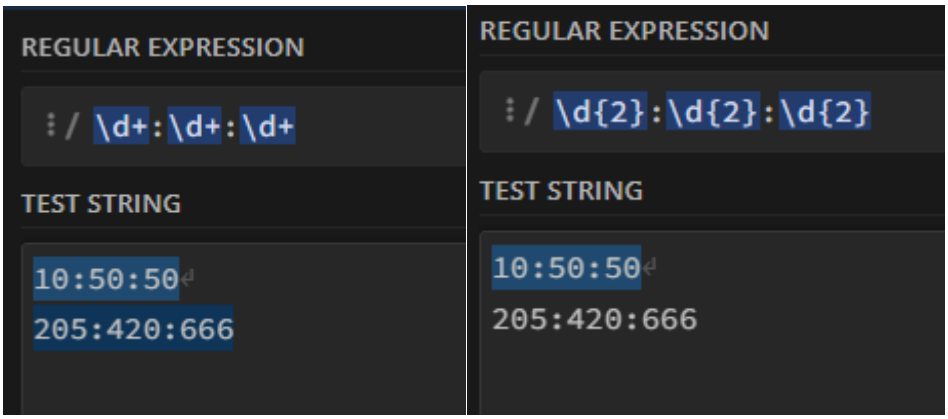
```
 10:50:50
\d+:\d+:\d+
```

A skutečně – Regex funguje

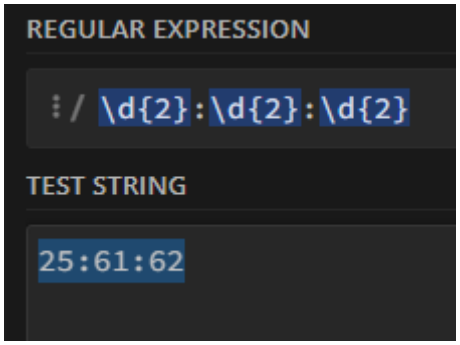


Co když ale budeme mít data, kde se vyskytuje například tento řetězec – 205:420:666. Zcela jistě to čas není. Co ale an to náš regex? ANO – match. Zkusme jej tedy upravit

Z výchozího textu víme, že chceme čas ve formátu hh:mm:ss – každá pozice má mít dvě číslice. Použijme místo kvantifikátoru + pevný kvantifikátor, a sice 2. Vkládáme jej do složených závorek.

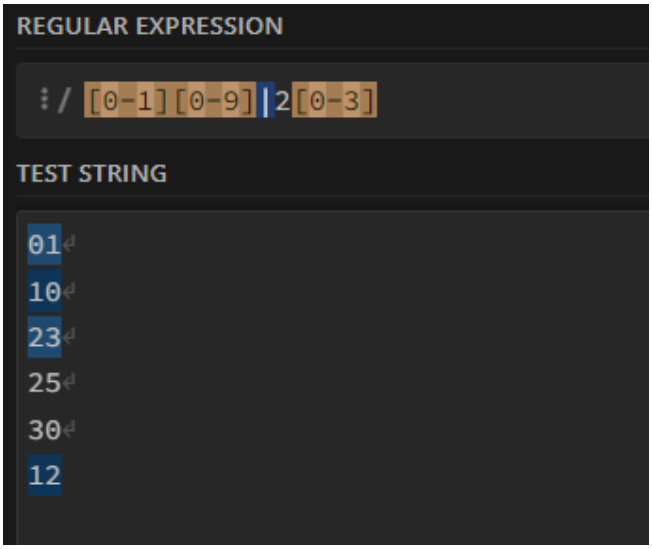


Toto vypadá lépe. Ale chybí ještě jedna věc k ošetření – co když dostaneme řetězec 25:61:62? Zcela jistě to čas není. Avšak co na to náš regex? Ano – MATCH.



Zkusme to omezit a začněme zleva. Na pozici hodin jsme v rozsahu od 00 do 24. Kombinujeme tedy nulu, jedničku, dvojku s čísly od 0 do 9. Ale co když začneme dvojkou? V tom případě ke dvojce můžeme přidat pouze čísla 0–3. Rozdělme tedy pozici hodin na dva případy – když začínáme jedničkou nebo nulou a když dvojkou:

```
10
[0-1][0-9]|2[0-3]
```



Tento zápis tedy říká, že matchne řetězec, jehož prvním znakem je rozmezí nula až jedna a druhým znakem nula až devět, NEBO řetězec, jehož první znak je dva a druhý nula až tři. Na pozici minut i sekund je to jednodušší. Jejich prvním znakem může být cokoli v rozmezí nula až pět, druhým cokoli v rozmezí nula až devět. Dejme to tedy vše dohromady.

Regex definující pozici hodin nejprve „zabalme“ do non-group tokenu, a to z toho důvodu, že tím nadefinujeme obsah, pro který platí operátor |.

```
10:50:50
[0-1][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]
```

3 Použití v pythonu

Zkusme nyní automaticky kontrolovat, zda textový řetězec, který zadáme, odpovídá našemu předdefinovanému Regexu. Využijeme na to jazyk Python.

Nejprve naimportujeme knihovnu re. Tato knihovna obsahuje spoustu funkcí [1], my však využijeme funkci re.match.

Tato funkce přijímá dva povinné argumenty, a sice samotný Regex a potom string, se kterým jej má porovnávat. Kód tady může vypadat takto:

```
import re

vstup = input("Zadejte retezec: ")

if re.match("(?:[^\.]+\.)?[^\.]+\.cz",vstup):
       print("Match!")
else:
    print("Bohuzel...")
```

Program od uživatele načte řetězec. Porovná jej s předem definovaným patternem. Jestliže regex řetězec matchuje, program vypíše Match!, v opačném případě Bohužel…

4 Regex pro Linux audit logy

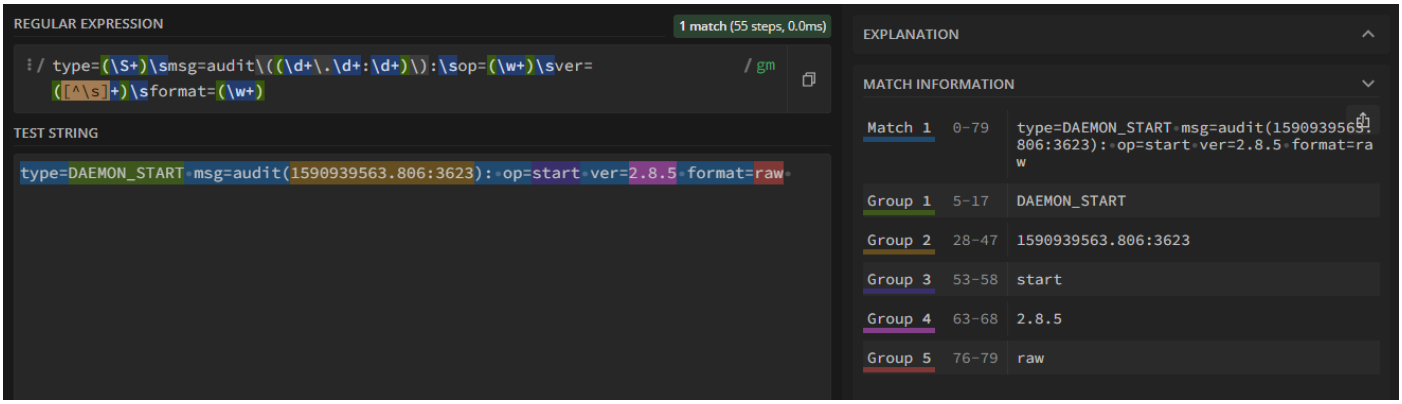
Vzorky jsou přílohou tohoto cvičení

```
type=DAEMON_START msg=audit(1590939563.806:3623): op=start ver=2.8.5 format=raw kernel=3.10.0-1062.el7.x86_64 auid=4294967295 pid=656 uid=0 ses=4294967295 subj=system_u:system_r:auditd_t:s0 res=success
```

Log je krásně strukturovaný – tedy Regex by neměl být tak složitý. Pevně definujme „proměnnou“, regexové tokeny používejme na hodnoty, například type=(\S+). Mezeru můžeme napsat jako mezeru, nebo tokenem pro whitepsace znak \s.

Ačkoli v první ukázce hodin byl počet číslic přesně definovaný, vzhledem k jinak pevnému formátu logu zde již není nutné číslice definovat pevně – ponechme to tedy následovně: msg=audit\((\d+\.\d+:\d+)\)

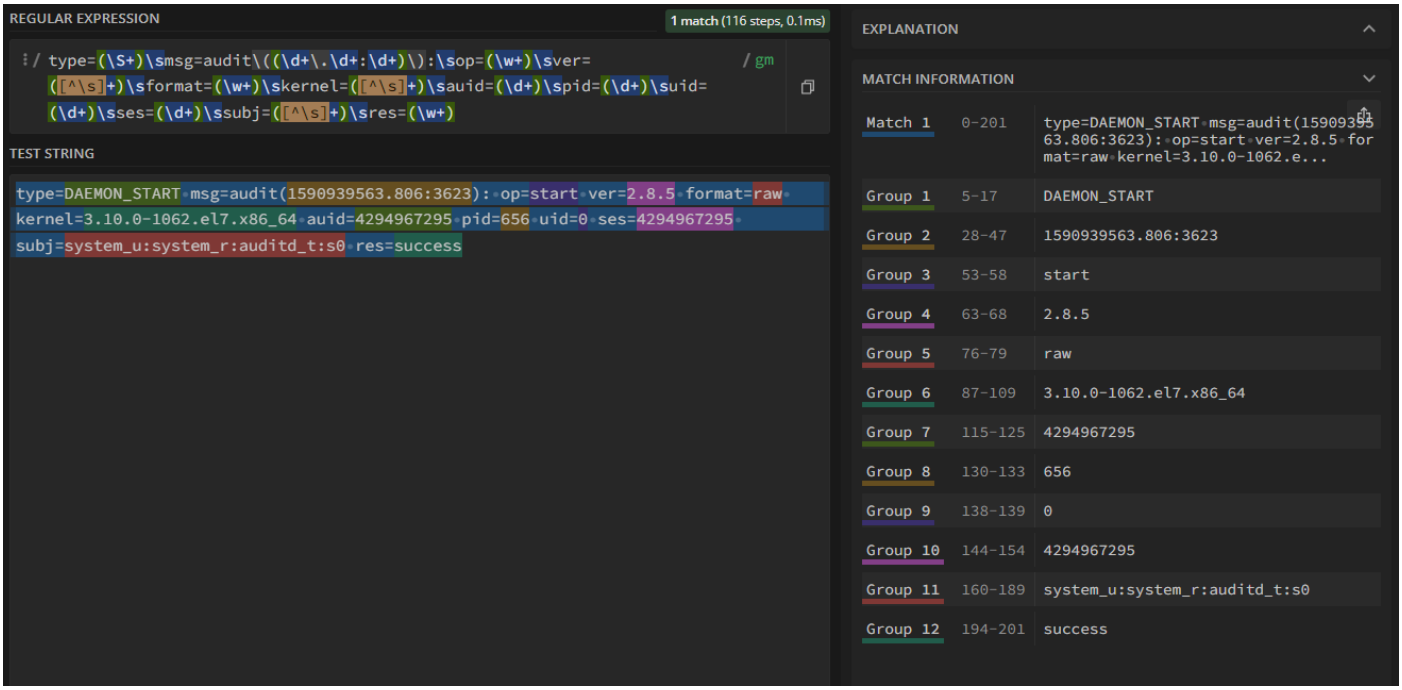
Op bude vždy nabývat hodnot takových, které nebudou číslené: op=(\w+), dále ver bude poněkud složitější – máme mezi čísly tečky, ale víme, že hodnota končí před mezerou, definujme to tedy následovně: ver=([^\s]+) – tedy match, dokud nenarazí na whitespace, tedy mezeru. Dále format bude taktéž nabývat hodnot takových, které neobsahují číslo: format=(\w+).



Pokračujme. U kernel nastává stejná situace jako u ver – je tu spousta různých znaků, ale víme, že jsou ohraničeny mezerou: kernel=([^\s]+), auid bude vždy číslo, tedy auid=(\d+), pid bude také číslo, proto pid=(\d+), uid také, tedy uid=(\d+), stejně tak ses=(\d+). Následuje subj, opakuje se situace jako u kernel, tedy subj=([^\s]+), poslední hodnotou je res. Jelikož v tomto případě nabývá hodnoty success, je pravděpodobné, že v opačném případě bude nabývat hodnoty například failure – víme, že to nebude hodnota obsahující číslo, a proto res=(\w+).

A tímto máme regex pro tento vzor logu hotov:

```
type=(\S+)\smsg=audit\((\d+\.\d+:\d+)\):\sop=(\w+)\sver=([^\s]+)\sformat=(\w+)\skernel=([^\s]+)\sauid=(\d+)\spid=(\d+)\suid=(\d+)\sses=(\d+)\ssubj=([^\s]+)\sres=(\w+)
```



Pozn. – používat všude proměnná=hodnota jako proměnná=([^\s]+), tedy po proměnné a rovnítku následuje hodnota, která se má matchovat do prvního whitespace znaku, je sice možné, avšak známe-li pattern logu a víme, co a na kterém místě standardně obsahuje (například po pid budou vždy čísla, po format vždy písmena etc.), je vhodné používat více deterministický zápis, a to z důvodu rizika nesprávného matche.

Na zbylé dva logy jsou Regexy níže:

```
type=CONFIG_CHANGE msg=audit(1590939564.067:5): audit_backlog_limit=8192 old=64 auid=4294967295
ses=4294967295 subj=system_u:system_r:unconfined_service_t:s0 res=1

type=(\S+)\smsg=audit\((\d+\.\d+:\d+)\):\saudit_backlog_limit=(\d+)\sold=(\d+)\sauid=(\d+)\sses=
(\d+)\ssubj=([^\s]+)\sres=(\d+)
```

```
type=SYSCALL msg=audit(1590939573.096:39): arch=c000003e syscall=175 success=yes exit=0
a0=1a6d340 a1=1d85 a2=41a96e a3=1a69300 items=0 ppid=821 pid=822 auid=4294967295 uid=0 gid=0
euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="modprobe"
exe="/usr/bin/kmod" subj=system_u:system_r:insmod_t:s0 key=(null)

type=(\S+)\smsg=audit\((\d+\.\d+:\d+)\):\sarch=(\S+)\ssyscall=(\d+)\ssuccess=(\w+)\sexit=(\d+)\s
a0=(\S+)\sa1=(\S+)\sa2=(\S+)\sa3=(\S+)\sitems=(\d+)\sppid=(\d+)\spid=(\d+)\sauid=(\d+)\suid=(\d+
)\sgid=(\d+)\seuid=(\d+)\ssuid=(\d+)\sfsuid=(\d+)\segid=(\d+)\ssgid=(\d+)\sfsgid=(\d+)\stty=([^\
s]+)\sses=(\d+)\scomm="([^"]+)"\sexe="([^"]+)"\ssubj=([^\s]+)\skey=([^\s]+)
```