Riferimento Rapido Espressioni Regolari

Pattern, quantificatori, gruppi, lookahead, flag

Pattern Base

Metacaratteri

`.`	Qualsiasi carattere (tranne newline)
`^`	Inizio della stringa / riga
`$`	Fine della stringa / riga
`*`	0 o più del precedente
`+`	1 o più del precedente
`?`	0 o 1 del precedente (opzionale)
`\`	Escape del metacarattere

Corrispondenza Letterale

```
hello     # matches "hello" exactly
a.c       # matches "abc", "alc", "a-c", etc.
\.txt     # matches literal ".txt"
```

Classi di Caratteri

Espressioni tra Parentesi

`[abc]`	Corrisponde a a, b o c
`[^abc]`	Qualsiasi tranne a, b, c
`[a-z]`	Lettera minuscola
`[A-Z]`	Lettera maiuscola
`[0-9]`	Cifra
`[a-zA-Z0-9]`	Alfanumerico

Classi Abbreviate

`\d`	Cifra `[0-9]`
`\D`	Non cifra `[^0-9]`
`\w`	Carattere di parola `[a-zA-Z0-9_]`
`\W`	Non carattere di parola
`\s`	Spazio bianco `[ \t\n\r\f]`
`\S`	Non spazio bianco

Quantificatori

Quantificatori Greedy

`*`	0 o più (greedy)
`+`	1 o più (greedy)
`?`	0 o 1 (greedy)
`{n}`	Esattamente n volte
`{n,}`	n o più volte
`{n,m}`	Da n a m volte

Quantificatori Lazy

`*?`	0 o più (lazy / non-greedy)
`+?`	1 o più (lazy)
`??`	0 o 1 (lazy)
`{n,m}?`	Da n a m (lazy)

I quantificatori lazy corrispondono al minor numero di caratteri possibile

Greedy vs Lazy

```
<.+>      # greedy: "<b>bold</b>"
<.+?>     # lazy:   "<b>"
```

Ancore

`^`	Inizio stringa (o riga con flag `m`)
`$`	Fine stringa (o riga con flag `m`)
`\b`	Confine di parola
`\B`	Non confine di parola
`\A`	Inizio stringa (non influenzato da `m`)
`\Z`	Fine stringa (non influenzato da `m`)

Esempi di Ancoraggio

```
^Hello        # starts with "Hello"
world$        # ends with "world"
\bword\b      # "word" as whole word
\Bword\B      # "word" inside another word
```

Gruppi e Alternanza

Gruppi di Cattura

```
(abc)         # capture group: match "abc"
(a|b|c)       # alternation: a or b or c
(cat|dog)     # match "cat" or "dog"
(\d{3})-(\d{4})  # groups: "123-4567"
```

Tipi di Gruppo

`(pattern)`	Gruppo di cattura
`(?:pattern)`	Gruppo non di cattura
`(?P<name>pat)`	Gruppo con nome (Python)
`(?<name>pat)`	Gruppo con nome (JS, .NET)
`\1 \2`	Backreference al gruppo 1, 2
`a|b`	Alternanza: a o b

Lookahead e Lookbehind

`(?=pattern)`	Lookahead positivo
`(?!pattern)`	Lookahead negativo
`(?<=pattern)`	Lookbehind positivo
`(?<!pattern)`	Lookbehind negativo

Esempi Lookaround

```
\d+(?= USD)     # digits followed by " USD"
\d+(?! USD)     # digits NOT followed by " USD"
(?<=\$)\d+      # digits preceded by "$"
(?<!\$)\d+      # digits NOT preceded by "$"
```

I lookaround corrispondono a una posizione senza consumare caratteri

Pattern Comuni

`\d{1,3}(\.\d{1,3}){3}`	Indirizzo IPv4 (base)
`[\w.+-]+@[\w-]+\.[\w.]+`	Email (base)
`https?://[\w./\-?&#=]+`	URL (base)
`\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}`	Numero di telefono US
`\d{4}-\d{2}-\d{2}`	Data (AAAA-MM-GG)
`#?[0-9a-fA-F]{6}`	Codice colore esadecimale

Questi sono pattern semplificati; in produzione potrebbe servire una validazione più rigorosa

Flag

`g`	Global: trova tutte le corrispondenze, non solo la prima
`i`	Case-insensitive
`m`	Multilinea: `^` / `$` corrispondono ai confini di riga
`s`	Dotall: `.` corrisponde anche ai newline
`x`	Verbose: ignora spazi bianchi, permette commenti
`u`	Unicode: supporto Unicode completo

Uso dei Flag per Linguaggio

```
/pattern/gi           # JavaScript
re.compile(r"pat", re.I | re.M)  # Python
grep -iE "pattern"    # grep (extended)
```