정규표현식 치트시트

데이브차일드의 정규표현식 치트시트

정규표현식(Regex)에 사용되는 기호, 범위, 그룹화, 어서션과 몇 가지 예시 패턴을 빠르게 참고할 수 있는 가이드.

- Anchors (앵커)
  - `^`: 문자열 시작, 또는 멀티라인 패턴에서 라인 시작
  - `\A`: 문자열 시작
  - `$`: 문자열 끝, 또는 멀티라인 패턴에서 라인 끝
  - `\Z`: 문자열 끝
  - `\b`: 단어 경계
  - `\B`: 단어 경계 아님
  - `\<`: 단어 시작
  - `\>`: 단어 끝
- Character Classes (문자 클래스)
  - `\c`: 제어 문자
  - `\s`: 공백 문자
  - `\S`: 공백 문자가 아님
  - `\d`: 숫자
  - `\D`: 숫자가 아님
  - `\w`: 단어 문자
  - `\W`: 단어 문자가 아님
  - `\x`: 16진수 숫자
  - `\O`: 8진수 숫자
- POSIX
  - `[:upper:]`: 대문자
  - `[:lower:]`: 소문자
  - `[:alpha:]`: 모든 문자 (알파벳)
  - `[:alnum:]`: 숫자와 문자
  - `[:digit:]`: 숫자
  - `[:xdigit:]`: 16진수 숫자
  - `[:punct:]`: 구두점
  - `[:blank:]`: 스페이스와 탭
  - `[:space:]`: 공백 문자
  - `[:cntrl:]`: 제어 문자
  - `[:graph:]`: 출력 가능한 문자
  - `[:print:]`: 출력 가능한 문자와 공백
  - `[:word:]`: 숫자, 문자 및 밑줄(_)
- Assertions (어서션)
  - `?=`: 긍정적 탐색
  - `?!`: 부정적 탐색
  - `?<=`: 긍정적 후방 탐색
  - `?<!` 또는 `?<!`: 부정적 후방 탐색
  - `?>`: 한번만 매칭되는 하위 표현식
  - `?()`: 조건 (if then)
  - `?()|`: 조건 (if then else)
  - `?#`: 주석
- Quantifiers (수량자)
  - `*`: 0개 이상
  - `{3}`: 정확히 3개
  - `+`: 1개 이상
  - `{3,}`: 3개 이상
  - `?`: 0개 또는 1개
  - `{3,5}`: 3,4 또는 5개
  - 수량자 뒤에 `?`를 붙이면 탐욕성이 사라짐 (최소 매칭)
- Escape Sequences (이스케이프 시퀀스)
  - `\`: 다음 문자를 이스케이프
  - `\Q`: 리터럴 시퀀스 시작
  - `\E`: 리터럴 시퀀스 종료
  - "이스케이핑"은 정규식의 특수 문자를 문자 그대로 처리하는 방법
- Common Metacharacters (일반 메타문자)
  - `^`, `[`, `.`, `$`, `{`, `*`, `(`, `\`, `+`, `)`, `|` ,`?`, `<`,`>`
  - 이스케이프 문자는 보통 `\`
- Special Characters (특수 문자)
  - `\n`: 줄 바꿈
  - `\r`: 캐리지 리턴
  - `\t`: 탭
  - `\v`: 수직 탭
  - `\f`: 폼 피드
  - `\xxx`: 8진수 문자
  - `\xhh`: 16진수 문자
- Groups and Ranges (그룹과 범위)
  - `.`: 줄 바꿈(n)을 제외한 모든 문자
  - `(a|b)`: a 또는 b
  - `(...)`: 그룹
  - `(?:...)`: 캡처하지 않는 그룹
  - `[abc]`: a 또는 b 또는 c
  - `[^abc]`: a 또는 b 또는 c가 아님
  - `[a-q]`: a부터 q까지 소문자
  - `[A-Q]`: A부터 Q까지 대문자
  - `[0-7]`: 0부터 7까지 숫자
  - `\x`: 하위 패턴 번호 x
  - (범위는 포함 범위)
- Pattern Modifiers (패턴 수정자):`*` 별표는 PCRE 패턴에서 지원하는 플래그임을 나타냅니다.
  - `g` : 전역 매칭 (Global match) — 함수 호출 옵션으로 사용되며, PCRE 패턴 내부 플래그가 아님
  - `i *` : 대소문자 구분 없음 (Case-insensitive) (`PCRE` 지원)
  - `m *` : 멀티라인 모드 (Multi-line) (`PCRE` 지원)
  - `s *` : 단일 라인(DOTALL) 모드 (`PCRE` 지원)
  - `x *` : 확장 모드 (Extended mode), 공백과 주석 무시 (`PCRE` 지원)
  - `e *` : 대체 결과 실행 (Evaluate replacement) — Perl, PHP preg_replace 등에서 지원 (`PCRE` 자체 수정자는 아님)
  - `U *` : 비탐욕적(ungreedy) 매칭 (`PCRE` 지원)
- String Replacement (문자열 치환)
  - `$n`: n번째 캡처 그룹
  - `$2`: 예시 `/^(abc(xyz))$/`에서 "xyz"
  - `$1`: 예시 `/^(?:abc)(xyz)$/`에서 "xyz"
  - `$'`: 매칭된 문자열 뒷 부분
  - `$+`: 마지막 매칭 그룹
  - `$&`: 전체 매칭 문자열
  - 일부 정규식 구현에서는 `$` 대신 `\` 사용

실용적인 정규표현식 패턴

이메일 주소 검증

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```

- 설명: 이메일 주소의 기본적인 형식을 검증
- 예시: `user@example.com`, `test.email+tag@domain.co.uk`

전화번호 검증

```
# 국제 전화번호 (선택적 국가코드)
^(\+?1[-.\s]?)?\(?([0-9]{3})\)?[-.\s]?([0-9]{3})[-.\s]?([0-9]{4})$

# 한국 전화번호
^(010|011|016|017|018|019)[-.\s]?[0-9]{3,4}[-.\s]?[0-9]{4}$
```

URL 검증

```
^https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*)$
```

- 설명: HTTP/HTTPS URL 형식 검증
- 예시: `https://www.example.com/path?query=value`

IP 주소 검증

```
# IPv4 주소
^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$

# IPv6 주소 (간단한 버전)
^(?:[0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}$
```

비밀번호 강도 검증

```
# 최소 8자, 대소문자, 숫자, 특수문자 포함
^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$

# 더 강력한 비밀번호 (최소 12자)
^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{12,}$
```

날짜 형식 검증

```
# YYYY-MM-DD 형식
^\d{4}-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])$

# MM/DD/YYYY 형식
^(0[1-9]|1[0-2])\/(0[1-9]|[12][0-9]|3[01])\/\d{4}$

# DD/MM/YYYY 형식
^(0[1-9]|[12][0-9]|3[01])\/(0[1-9]|1[0-2])\/\d{4}$
```

신용카드 번호 검증

```
# Visa (13, 16, 19자리, 4로 시작)
^4[0-9]{12}(?:[0-9]{3})?(?:[0-9]{3})?$

# Mastercard (16자리, 5로 시작하거나 2로 시작)
^(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}$

# American Express (15자리, 34 또는 37로 시작)
^3[47][0-9]{13}$
```

HTML 태그 제거

```
<[^>]*>
```

- 설명: HTML 태그를 모두 제거
- 예시: `<p>Hello <b>World</b></p>` → `Hello World`

파일 확장자 추출

```
\.([a-zA-Z0-9]+)$
```

- 설명: 파일명에서 확장자 부분을 캡처

- 예시: `document.pdf` → 그룹 1: `pdf`

중복 공백 제거

```
\s+
```

- 설명: 여러 개의 연속된 공백을 찾음
- 치환: 단일 공백으로 교체

한국어 텍스트 처리

```
# 한글만 추출
[가-힣]+

# 한글, 영문, 숫자만 허용
^[가-힣a-zA-Z0-9\s]+$

# 한국 주민등록번호 (간단한 형식만)
^\d{6}-[1-4]\d{6}$
```

로그 파일 파싱

```
# Apache 로그 형식
^(\S+) (\S+) (\S+) \[([\w:/]+\s[+
\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3})
(\d+) "([^"]*)" "([^"]*)"$

# 그룹 설명:
# 1: IP 주소
# 2: - (보통 사용되지 않음)
# 3: 사용자 ID
# 4: 타임스탬프
# 5: HTTP 메서드
# 6: 요청 URL
# 7: HTTP 버전
# 8: 상태 코드
# 9: 응답 크기
# 10: Referer
# 11: User-Agent
```

JSON 키 추출

```
"([^"]+)":\s*"([^"]*)"
```

- 설명: JSON에서 키-값 쌍을 추출
- 예시: `{"name": "John", "age": "30"}` → 그룹 1: `name`, 그룹 2: `John`

SQL 인젝션 방지 패턴

```
# 위험한 SQL 키워드 감지
(?i)(union|select|insert|update|delete|
drop|create|alter|exec|execute|script)

# SQL 주석 패턴
--.*$|/\*.*?\*/
```

프로그래밍 언어별 정규표현식

JavaScript

```javascript
// 이메일 검증
const emailRegex = /^[^\s@]+@[^\s@]+\.
[^\s@]+$/;
const isValidEmail =
emailRegex.test("user@example.com");

// 전화번호 포맷팅
const phoneRegex = /(\d{3})(\d{3})
(\d{4})/;
const formatted =
"1234567890".replace(phoneRegex, "($1)
$2-$3");

// URL 추출
const urlRegex = /https?:\/\/[^\s]+/g;
const urls = text.match(urlRegex);
```

Python

```python
import re

# 이메일 검증
email_pattern = r'^[a-zA-Z0-9._%+-]+@[a-
zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
is_valid = re.match(email_pattern,
"user@example.com")

# 전화번호 포맷팅
phone_pattern = r'(\d{3})(\d{3})(\d{4})'
formatted = re.sub(phone_pattern, r'(\1)
\2-\3', "1234567890")

# 모든 이메일 주소 찾기
emails = re.findall(r'\b[A-Za-z0-9._%
+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
text)
```

Java

```java
import java.util.regex.Pattern;
import java.util.regex.Matcher;

// 이메일 검증
Pattern emailPattern =
Pattern.compile("^[a-zA-Z0-9._%+-]+@[a-
zA-Z0-9.-]+\\.[a-zA-Z]{2,}$");
Matcher matcher =
emailPattern.matcher("user@example.com");
boolean isValid = matcher.matches();

// 전화번호 포맷팅
String phonePattern = "(\\d{3})(\\d{3})
(\\d{4})";
String formatted =
"1234567890".replaceAll(phonePattern,
"($1) $2-$3");
```

성능 최적화 팁

효율적인 패턴 작성

```
# 나쁜 예: 과도한 백트래킹
(.*)*

# 좋은 예: 구체적인 패턴
[a-zA-Z0-9]+

# 나쁜 예: 중첩된 선택적 그룹
(a(b(c(d)?)?)?)?

# 좋은 예: 단순한 패턴
[a-d]+
```

캐시 활용

```javascript
// 패턴을 미리 컴파일하여 재사용
const emailRegex = new RegExp('^[^\
\s@]+@[^\\s@]+\\.[^\\s@]+$');

function validateEmail(email) {
    return emailRegex.test(email);
}
```

앵커 사용

```
# 전체 문자열 매칭 (권장)
^pattern$

# 단어 경계 사용
\bword\b

# 라인 시작/끝
^start.*end$
```

디버깅 및 테스트

온라인 도구

- Regex101: https://regex101.com/
- Regexr: https://regexr.com/
- RegEx Pal: https://www.regexpal.com/

테스트 방법

```javascript
// 패턴 테스트 함수
function testRegex(pattern, testCases) {
    const regex = new RegExp(pattern);
    testCases.forEach(testCase ⇒ {
        const result =
regex.test(testCase.input);
        console.log(`${testCase.input}:
${result ? 'PASS' : 'FAIL'}`);
    });
}

// 사용 예제
testRegex('^[a-zA-Z0-9._%+-]+@[a-zA-
Z0-9.-]+\\.[a-zA-Z]{2,}$', [
    { input: 'user@example.com' },
    { input: 'invalid-email' },
    { input:
'test.email+tag@domain.co.uk' }
]);
```