Regular Expressions

> **정규표현식은 특정 문자열을 가리키는 패턴이다. 즉 이 패턴으로 원하는 문자열을 찾는다.** - 켄 톰슨

기본 패턴 찾기

- 클래스; []
- 단축문자; \d: 숫자, \D: 숫자 이외의 문자
- 임의의 문자; .: 아무문자(행의 끝에 오는 개행 문자는 제외)
- 그룹참조; ()
- 수량자; {숫자}: 해당 갯수, {최소숫자,최대숫자}: 최소숫자부터 최대숫자 갯수만큼 , ?: 하나 이하, +: 하나 이상, *: 0 이상
- 상수

정규표현식의 메타 문자			
메타 문자	**이름**	**코드 포인트**	**의미**
.	점	U+002E	임의의 문자를 찾음
****	역슬래시	U+005C	문자의 특수 의미를 없앰
|	세로줄	U+007C	선택
^	악센트 기호	U+005E	행의 시작을 나타내는 앵커
$	달러 기호	U+0024	행의 끝을 나타내는 앵커
?	물음표	U+003F	0번 또는 한 번을 나타내는 수량자
*****	별표	U+002A	0번 이상을 나타내는 수량자
+	덧셈 기호	U+002B	한 번 이상을 나타내는 수량자
[	왼쪽 대괄호	U+005B	문자 클래스 시작
]	오른쪽 대괄호	U+005D	문자 클래스 끝
{	왼쪽 중괄호	U+007B	수량자 또는 블록 시작
}	오른쪽 중괄호	U+007D	수량자 또는 블록 끝
(	왼쪽 소괄호	U+0028	그룹 시작
)	오른쪽 소괄호	U+0029	그룹 끝

숫자

- \d, [0-9]

숫자가 아닌 문자

- \D, [^0-9]; 공백, 구두점, 인용부호, 하이픈, 슬래시, 대괄호 같은 문자도 찾는다.
- \w; 모든 영문자, 숫자, _, 기타 스크립트 문자 = [a-zA-Z0-9_]
- \W; [^a-zA-Z0-9_]

단축 문자[1)]	
단축문자	**설명**
\a	벨 문자
[\b]	백스페이스 문자

단축 문자[1]	
단축문자	**설명**
\c x	제어 문자
\d	숫자
\D	숫자가 아닌 문자
\d xxx	문자의 10진수 값
\f	폼 피드 문자
\h	수평 공백
\H	수평 공백이 아닌 문자
\r	캐리지 리턴
\n	개행 문자
\0xxx	문자의 8진수 값
\s	공백 문자
\S	공백이 아닌 문자
\t	수평 탭 문자
\v	수직 탭 문자
\V	수직 탭이 아닌 문자
\w	영문자, 숫자, _, 기타 스크립트 문자
\W	영문자, 숫자, _, 기타 스크립트 문자를 제외한 문자
\0	널 문자
\x xx	문자의 16진수 값
\u xxx	문자의 유니코드 값

공백

- \s, [\t\n\r]

임의의 문자

- . (U+002E); 개행 문자[2]를 제외한 모든 문자
- \b; 단어의 경계

참조 그룹

- 찾고자 하는 내용을 ()로 감싼 후(참조) \$1, \$2 혹은 \1, \2와 같이 역참조

경계

- 위치 지정자 assertion
  - 행의 시작과 끝
  - 단어의 경계(두 종류)
  - 행의 시작과 끝
  - 문자열 상수를 나타내는 경계
- ^; 문맥에 따라 행이나 문자열 또는 문서 전체의 시작

- $; 행이나 문자열의 끝
- \b; 단어의 경계
- \B
- \<; 단어의 시작
- \>; 단어의 끝
- \A; ^의 기능처럼 해당 패턴이 행의 시작 위치에 나오는지 찾는다. PCRE(Perl Compatible Regular Expression)
- \Z, \z; 해당 패턴이 행의 끝에 나오는지
- \Q문자\E; 문자열을 상수로 지정

선택, 그룹, 역참조

- 그룹; 텍스트를 괄호로 묶은 것
  - 두 가지 이상의 패턴 중 하나를 선택할 때
  - 서브패턴을 만들 때
  - 나중에 역참조하기 위해 참조 그룹을 지정할 때
  - 수량자 같이 그룹으로 묶은 패턴에 어떤 작업을 적용할 때
  - 비참조 그룹을 사용할 때
  - 원자 그룹을 만들 때(고급과정)
- 선택 alternation; 찾고자 하는 패턴을 선택

정규표현식 옵션[3]		
옵션	**설명**	**지원**
(?d)	유닉스 행	자바
(?i)	대소문자 구분 없앰	PCRE, Perl, 자바
(iJ)	이름 반복 허용	PCRE*
(?m)	다중 행	PCRE, Perl, 자바
(?s)	한 행(dotall)	PCRE, Perl, 자바
(?u)	유니코드	자바
(?U)	욕심쟁이 모드 해제	PCRE
(?x)	공백과 코멘트는 무시	PCRE, Perl, 자바
(?-...)	옵션 기능 제거	PCRE

Perl 변경자(플래그)[4]	
변경자	**설명**
a	\d, \s, \w 및 ASCII 범위 내의 POSIX 문자
c	찾기 실패 후 현재 위치 유지
d	현재 플랫폼의 기본 설정 사용
g	global 모드 설정
i	대소문자 구분 없앰
l	현재 로케일 설정 사용
m	다중 행 문자열
p	찾은 문자열을 저장
s	문자열을 한 행으로 간주
u	유니코드 규칙 사용
x	공백과 코멘트 무시

- 서브패턴 ex) (the|The|THE), (t|T)h(e|eir)

- 그룹 참조와 역참조; \1, $1
- 그룹 이름 지정; ?<one>, ?<two>
- 그룹 이름으로 참조; $+{one}, $+{two}

그룹 이름 지정 구문	
구문	**설명**
(?<name>...)	그룹 이름 지정
(?name...)	또 다른 그룹 이름 지정
(?P<name>...)	파이썬에서 그룹 이름 지정
\k<name>	Perl에서 이름으로 참조
\k'name'	Perl에서 이름으로 참조
\g{name}	Perl에서 이름으로 참조
\k{name}	.NET에서 이름으로 참조
(?P=name)	파이썬에서 이름으로 참조

- 비참조 그룹; 역참조가 필요 없을 경우 ex) (?:the|The|THE), (?i)(?:the), (?:(?i)ther), (?i:the)
- 원자 그룹; 비참조 그룹 중 하나. 백트래킹 backtracking을 하는 정규표현식 엔진을 사용하는 경우, 이 그룹을 사용하면 정규표현식 전체는 아니더라도 원자 그룹에 해당하는 부분의 백트래킹 기능을 없앰.
  - ex) (?>the)

문자 클래스

- 대괄호식 bracketed expressions
- [a-z], [A-Z], [aeiou], [0-9], ...
- 부정 문자 클래스; [^문자]
- 합집합; [0-3[6-9]]
- 차집합; [a-z&&[^m-r]]

POSIX 문자 클래스

POSIX 문자 클래스	
문자 클래스	**설명**
[[:alnum:]]	영문자와 숫자
[[:alpha:]]	알파벳 문자(영문자)
[[:ascii:]]	ASCII 문자(모두 128)
[[:blank:]]	빈 문자
[[:ctrl:]]	제어 문자
[[:digit:]]	숫자
[[:graph:]]	그래픽 문자
[[:lower:]]	소문자
[[:print:]]	인쇄 가능한 문자
[[:punct:]]	구두점 문자
[[:space:]]	공백 문자
[[:upper:]]	대문자
[[:word:]]	단어
[[:xdigit:]]	16진수

유니코드와 기타 문자

- \u16진수코드; \u00e9, \xe9
- 8진수로 문자 찾기; \351(==\u00e9)
- 제어문자 찾기; /cx

수량자

- 욕심많고, 게으르고, 독점적인 수량자
- 욕심쟁이; 입력된 텍스트에서 가능한 많은 부분, 즉 전체 문자열을 가져온 다음 원하는 문자열을 찾아내려고 시도.
- 게으른(또는 느슨한) 수량자; 우선 문자열의 시작 위치부터 검색. 원하는 문자열을 찾을 때까지 하나씩 문자를 증가시키면서 찾다가, 계속 찾지 못하면 마지막으로 전체 텍스트를 검사. 일반 수량자 뒤에 물음표(?)를 붙인다.
- 독점적인 수량자; 전체 텍스트를 가져온 후, 찾고자 하는 문자열인지를 검사. 오직 한 번만 시도, 백트래킹은 하지 않음. 일반 수량자 뒤에 덧셈기호(+)를 붙인다.
- 클리니 스타(*)
- *, +, ?

기본 수량자	
구문	**설명**
?	0 또는 하나
+	하나 이상
*	0 또는 그 이상

게으른 수량자	
구문	**설명**
??	게으르게 0번 또는 한 번
+?	게으르게 한 번 이상
***?**	게으르게 0번 이상
{n}?	게으르게 n번
{n.}?	게으르게 n번 이상
{m,n}?	게으르게 m부터 n까지

독정적인 수량자	
구문	**설명**
?+	독점적으로 0번 또는 한 번
++	독점적으로 한 번 이상
***+**	독점적으로 0번 이상
{n}+	독점적으로 n번
{n.}+	독점적으로 n번 이상
{m,n}+	독점적으로 m에서 n까지

탐색

- lookaround
- 긍정형 전방 탐색(Positive lookaheads)

- 부정형 전방 탐색(Negative lookaheads)
- 긍정형 후방 탐색(Positive lookbehinds)
- 부정형 후방 탐색(Negative lookbehinds)

자주 쓰는 정규표현식

- 자음 모음을 포함한 모든 한글 검색; <alert type="info">[ㄱ-|5가-히ㅎ]+

</alert>

- 이메일 주소 확인; <alert type="info">^[a-zA-Z][\w-.%]+@([\w-]{2,63}\.)+[a-zA-Z]{2,4}$</alert>
    1. ^문장의 시작부터 $끝까지 체크
    2. 대소문자 구분 없이 a-z,0-9,.,-,_의 1개 이상
    3. @
    4. 대소문자 구분 없이 a-z,0-9,-,_가 2글자 이상 63자 이하의 글자가 오며 점(.)은 1개 이상
    5. 대소문자 구분 없이 a-z의 2개 이상 4개 이하
- IP 주소체크; <alert type="info">((25[0-5]|2[0-4][0-9]|[01]?[0-9[0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)</alert>
    1. 250~251 또는 200~249 또는 0~199 \. 정확히 3번
    2. 250-251 또는 200~249 또는 0~199 1번
- URL; <alert type="info">^https?://([\w-]+.)+(/[\w-./?&%=]*)?$</alert>
    1. ^문장의 시작부터 $끝까지 체크
    2. http로 시작하고 s는 있거나 없거나
    3. :// 체크
    4. zA-Z0-9 등 \w에 매칭되는 문자와 -은 1번 이상
    5. 점(.)
    6. 4,5번의 조합이 1번 이상
    7. / 이후 \w에 매칭되는 문자와 -./?&%= 조합의 0번 이상이 있거나 없거나
- 신용카드번호
    - 마스터카드(5로시작); <alert type="info">^5\d{3}-?\d{4}-?\d{4}-?\d{4}$</alert>
    - 비자카드(4로시작); <alert type="info">^4\d{3}-?\d{4}-?\d{4}-?\d{4}$</alert>
    - 국내전용(9로시작); <alert type="info">^9\d{3}-?\d{4}-?\d{4}-?\d{4}$</alert>
    - 아메리칸익스프레스(3으로 시작, 두번째숫자는 4또는 7); <alert type="info">^3[47]\d{2}-?\d{4}-?\d{4}-?\d{4}$</alert>
        1. ^문장의 시작부터 $끝까지 체크
        2. \d{4} 0-9까지의 숫자가 네자ㄹ
        3. -? -가 있거나 없거나
- HTML 주석; <alert type="info"><!-{2,}.*?-{2,}></alert>
    1. <!로 시작
    2. -가 2개 이상
    3. .*? 아무 문자가 0번 이상
    4. -가 2개 이상
    5. >로 닫음

- ~~이름 masking; <alert type="info">(JAVA) str.replaceFirst("(\\W)\\W+\$", "\$1*****"))</alert>~~
- 숫자를 제외한 모든 문자 제거; <alert type="info">(JAVA) strConverted = str.replaceAll("\\D",

"");</alert>

- 숫자만으로 되어있는 사업자 번호 → 사업자번호 형식; <alert type="info">(JAVA) strConverted = str.replaceFirst("(^\\d{3})(\\d{2})(\\d{5})\$", "\$1-\$2-\$3");</alert>
- 숫자만으로만 되어있는지 확인; <alert type="info">(JAVA) bDigitsOnly = str.matches("^\\d+\$");</alert>
- 첫글자만 보이게 마스킹; <alert type="info">(JAVA) strMasked = str.replaceFirst("(^\\S)\\S+\$", "\$1*****");</alert>
- Password Validataions (JAVA & TypeScript)

```
    public static Boolean hasSpecialCharacter(String strPassword)
    {
        Boolean bHasSpecialCharacter = Pattern.matches("^(?=.*[\\x21-
\\x2f\\x3a-\\x40\\x5b-\\x60\\x7b-\\x7e]).{1,}", strPassword);

        if(true == bHasSpecialCharacter)
        {
            System.out.println(strPassword + " has Special
Character.");
        }
        else
        {
            System.out.println(strPassword + " has no Special
Character.");
        }

        return bHasSpecialCharacter;
    }

    public static Boolean isValidPassword(String strPassword)
    {
        Boolean bValid = Pattern.matches("^(?=.*[a-zA-
Z])(?=.*[0-9])(?=.*[\\x21-\\x2f|\\x3a-\\x40|\\x5b-\\x60|\\x7b-
\\x7e]).{10,16}", strPassword);

        System.out.println(strPassword + ":" + bValid);
        return bValid;
    }

    public static Boolean hasAlphabet(String strPassword)
    {
        Boolean bHasAlphabet = Pattern.matches("^(?=.*[a-zA-Z]).{1,}",
strPassword);

        if(true == bHasAlphabet)
        {
            System.out.println(strPassword + " has Alphabet.");
        }
        else
        {
            System.out.println(strPassword + " has no Alphabet.");
```

```
        }

        return bHasAlphabet;
    }

    public static Boolean hasDigit(String strPassword)
    {
        Boolean bHasDigit = Pattern.matches("^(?=.*[\\d]).{1,}",
strPassword);

        if(true == bHasDigit)
        {
            System.out.println(strPassword + " has Digits.");
        }
        else
        {
            System.out.println(strPassword + " has no Digits.");
        }

        return bHasDigit;
    }

    public static void testPassword(String strPassword)
    {
        System.out.println("Your password is : " + strPassword);

        hasAlphabet(strPassword);
        hasDigit(strPassword);
        hasSpecialCharacter(strPassword);
    }
```

```
function hasAlphabet(strPassword: string) {
  const reg = new RegExp("^(?=.*[a-zA-Z]).{1,}$")
  const bHasAlphabet = reg.test(strPassword)

  if(false === bHasAlphabet) {
    console.log(`${strPassword} has no alphabet.`)
  } else {
    console.log(`${strPassword} has alphabet.`)
  }
  return bHasAlphabet
}

function hasDigit(strPassword: string) {
  const reg = new RegExp("^(?=.*[0-9]).{1,}$")
  const bHasDigit = reg.test(strPassword)

  if(false === bHasDigit) {
    console.log(`${strPassword} has no digit.`)
  } else {
```

```
    console.log(`${strPassword} has digit.`)
  }

  return bHasDigit
}

function hasSpecialCharacter(strPassword: string) {
  const reg = new RegExp("^(?=.*[\\x21-\\x2f\\x3a-\\x40\\x5b-
\\x60\\x7b-\\x7e]).{1,}$")
  const bHasSpecialCharacter = reg.test(strPassword)

  if(false === bHasSpecialCharacter){
    console.log(`${strPassword} has no special character.`)
  } else {
    console.log(`${strPassword} has special character.`)
  }

  return bHasSpecialCharacter
}

function isValidPassword(strPassword: string) {
  const reg = new RegExp("^(?=.*[a-zA-Z])(?=.*[0-9])(?=.*[\\x21-
\\x2f|\\x3a-\\x40|\\x5b-\\x60|\\x7b-\\x7e]).{10,16}$")
  const bValid = reg.test(strPassword)

  if(false === bValid) {
    console.log(`${strPassword} is invalid.`)
  } else {
    console.log(`${strPassword} is valid.`)
  }

  return bValid
}

function testPassword(strPassword: string) {
  console.log(`Your Password is : ${strPassword}`)

  hasAlphabet(strPassword)
  hasDigit(strPassword)
  hasSpecialCharacter(strPassword)

  isValidPassword(strPassword)

}
```

- camelCase to snake_case and MACRO_CASE (JAVA)

```
    public static String getSnakeCaseFromCamelCase(String strCamelCase)
    {
        return strCamelCase.replaceAll("([A-Z])", "_$1").toLowerCase();
    }
```

```
    public static String getMacroCaseFromCamelCase(String strCamelCase)
    {
        return strCamelCase.replaceAll("([A-Z])", "_$1").toUpperCase();
    }
```

- Card Number Validation (TypeScript)

```
function validateIBKCEOCard(strCardNumber: string) {
  const regDigitsOnly = /(\D+)/gi
  const strDigits = strCardNumber.replace(regDigitsOnly, "")

  if (strDigits.length != 16) {
    console.log(strCardNumber + ' is invalid card number')
    return
  }
  console.log('validate IBK CEO Card: ' + strCardNumber + ' => ' +
strDigits + '(' + strDigits.length + ')')

  const regValidate: RegExp = /^943003|^552103\d+$/g
  const bValid = regValidate.test(strDigits)

  console.log('is valid? ' + bValid)
}
```

Tools

프로그램과 라이브러리

- Perl
- PCRE(Perl Compatible Regular Expression)
- 루비에서 사용하는 오니구루마
- 파이썬
- RE2

On-line

- RegEx Pal
- RegExr
- Rubular

Offline

- QED
- ed
- sed; s/pattern1/replacetext/ s(substitute)
- vi(vim)

- grep
- awk
- TextMate macOS
- Notepad++ Windows
- Oxygen XML Editor
- reggy

References

- [자바 코딩] REGULAR EXPRESSION 정규표현식 - 숫자만 허용하기
- [Java] 자바 정규 표현식 (Pattern, Matcher) 사용법 & 예제
- 정규 표현식
- 실제 사용하는 정규식 패턴
- 자바스크립트 정규식 정리(숫자만, 한글만, 메일, 전화번호, 비밀번호)
- [JAVA] 정규식을 이용한 마스킹(정규표현식 마스킹 처리)
- 정규 표현식 기본
- 정규표현식(Reg)
- [JAVA] 이름 마스킹처리
- Class Pattern
- [Javascript] 정규식을 이용한 특수문자, 한글 등 특정 문자 체크(제거)

1) 이 외에 더 있음, 정규표현식 엔진에 따라 지원 여부 다양

2) 유닉스에서는 \n(U+0004), 윈도우에서는 \n과\r(U+000D)

3) 서브패턴 이름 지정은 http://www.pcre.org/pcre.txt 참고

4) http://perldoc.perl.org/perlre.html#Modifiers 참고

From:
https://theta5912.net/ - **reth**

Permanent link:
https://theta5912.net/doku.php?id=public:computer:regexp&rev=1672502216

Last update: **2023/01/01 00:56**