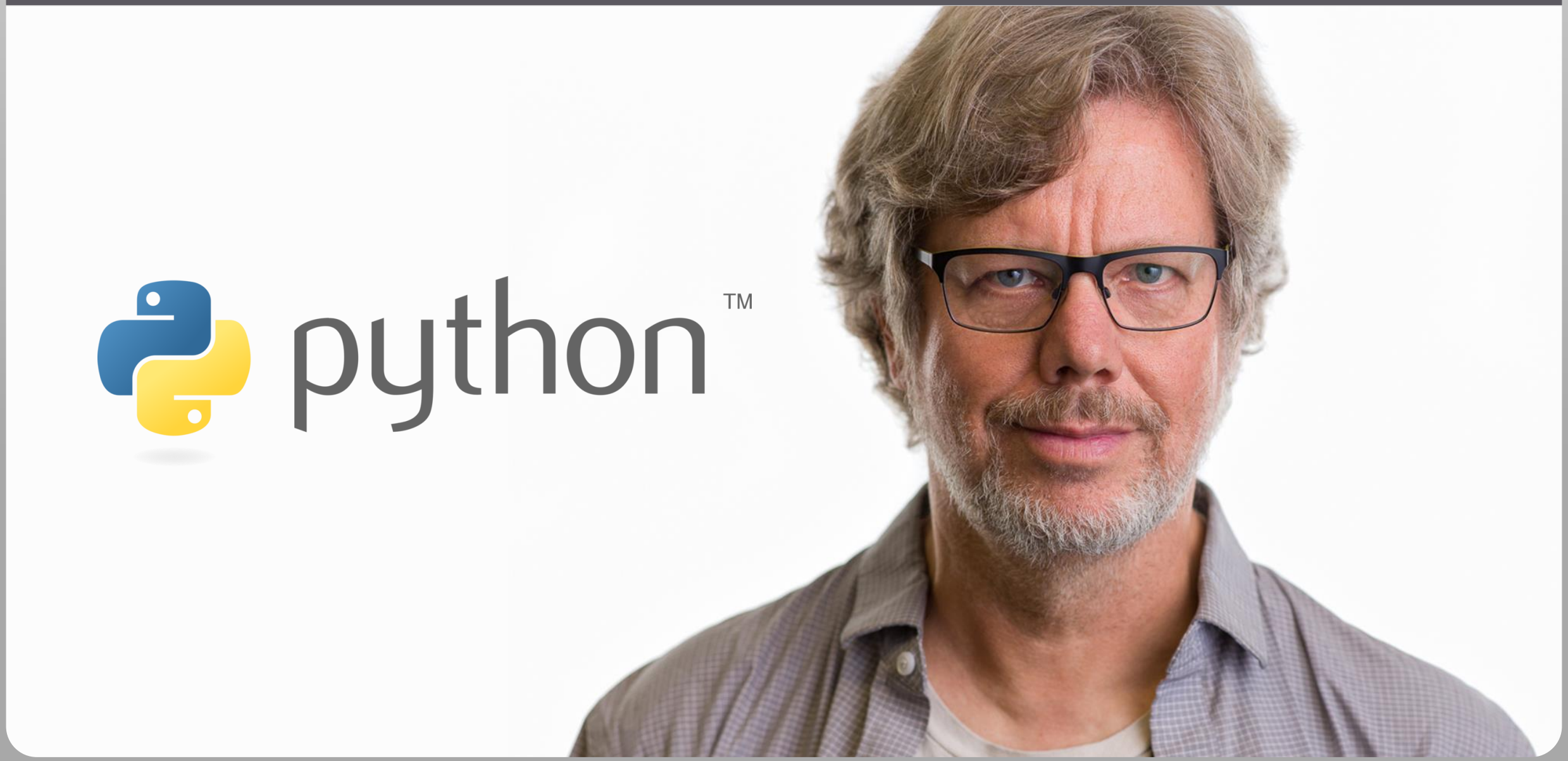
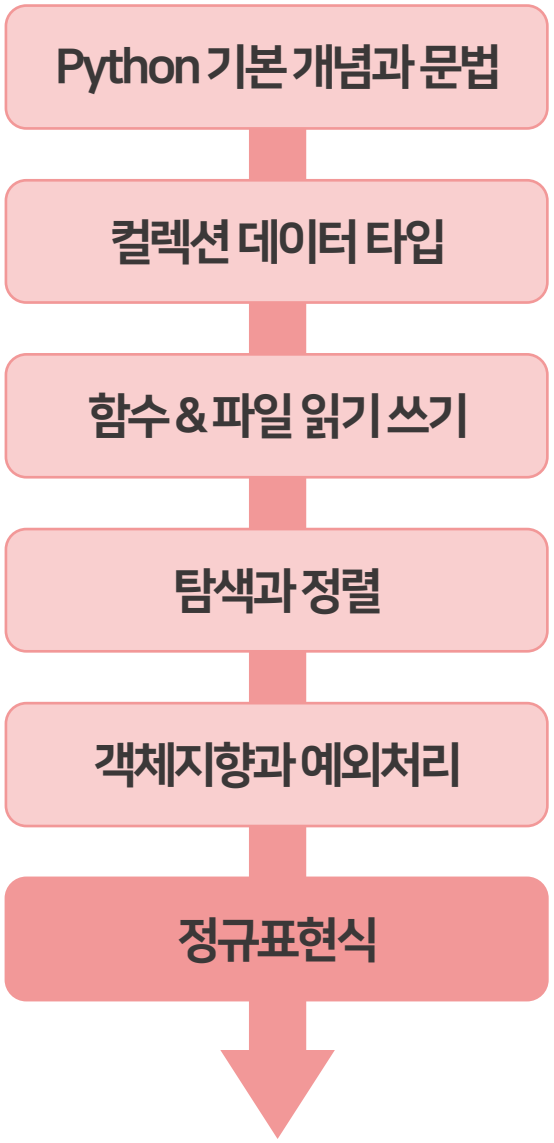
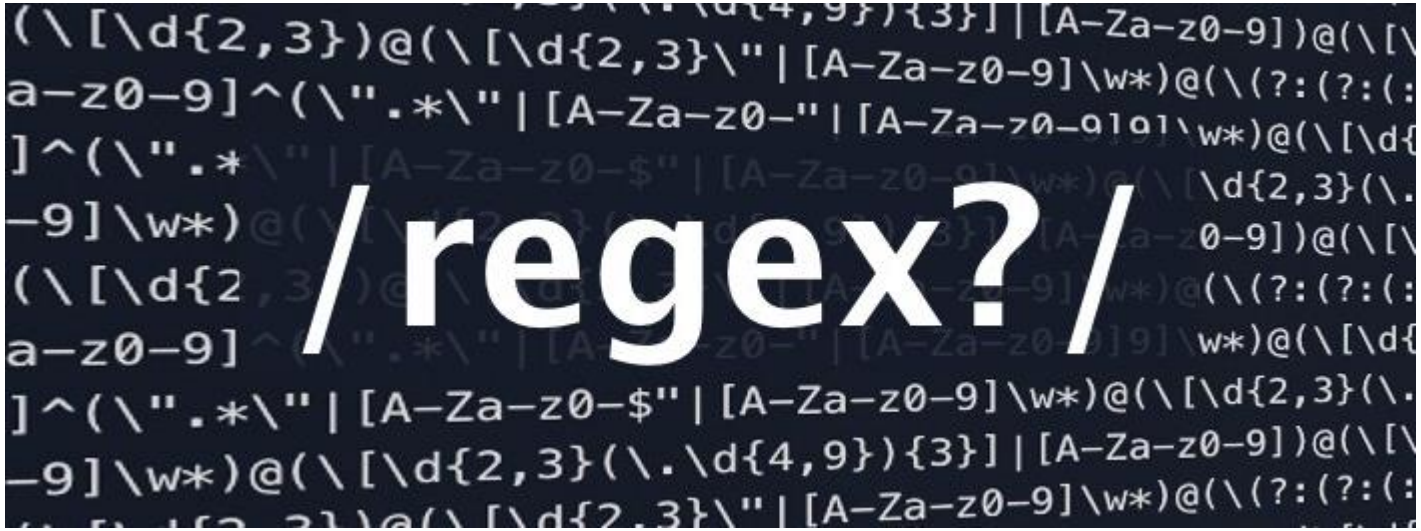

python™



1. 정규표현식
2. 정규표현식 - 메타문자 []
3. 정규표현식 - 메타문자 .
4. 정규표현식 - 메타문자 *, +
5. 정규표현식 - 메타문자 ?
6. match 함수
7. search 함수
8. finddll, finditer 함수
9. 정규표현식 - 메타문자 |, ^, $
10. 정규표현식 - 메타문자 ()
11. 자주 쓰이는 패턴

정규표현식(Regular Expressions)

복잡한 문자열을 처리하는 기법인 정규표현식은 Python만의 고유문법이 아니라
문자열을 처리하는 모든 곳에서 통용됩니다.



정규표현식(Regular Expressions) - 메타문자

```
.  ^  $  *  +  ?  {  }  [  ]  \  |  (  )
```

정규표현식에서 특별한 용도로 사용되는 메타문자에는 위와 같은 것들이 있습니다.

정규표현식(Regular Expressions) - 문자 클래스

'[ ]'는 문자 클래스로, '[ ]' 사이에 명시된 문자들과의 매치 여부를 확인할 수 있습니다.

정규식	평가 대상 문자열	매치 여부	설명
[abc] 또는 [a-c]	a	Yes	정규식과 일치하는 "a"가 존재하므로 매치
	before	Yes	정규식과 일치하는 "b"가 존재하므로 매치
	dude	No	정규식과 일치하는 문자가 하나도 없음

정규식	설명
[a-zA-Z]	알파벳만 매치
[0-9]	숫자만 매치
[^0-9]	숫자가 아닌 것들과 매치
[a-zA-Z0-9]	알파벳 + 숫자만 매치
[^a-zA-Z0-9]	알파벳+숫자가 아닌 것들과 매치

정규식	설명
\s	whitespace 문자와 매치 =[\t\n\r\f\v]
\S	whitespace 문자가 아닌 것들과 매치
\d	숫자만 매치
\D	숫자가 아닌 것들과 매치
\w	숫자를 포함한 모든 문자 매치
\W	알파벳+숫자가 아닌 것들과 매치

```
import re

def match_check(regex, string):
    if re.search(re.compile(regex), string):
        print(f'{regex}, {string} is Match!')
    else:
        print(f'{regex}, {string} is ' + '\033[101m' + 'Not' +'\033[0m' + ' Match!')
```

정규표현식(Regular Expressions) - 문자 클래스

```
match_check("[abc]", "alphabet")
match_check("[abc]", "liberty")
match_check("[abc]", "digit")

match_check("[0-9]", "123") # match_check("\d", "123")
match_check("[^0-9]", "123") # match_check("\D", "123")

match_check("[a-z]", "Alphabet")
match_check("[a-zA-Z]", "Alphabet")

match_check("[a-zA-Z0-9]", "GilDong123") # match_check("\w", "GilDong123")
match_check("[^a-zA-Z0-9]", "GilDong123") # match_check("\W", "GilDong123")

match_check("\s", "Hello World")

match_check("[가-힣]", "홍길동")
```

정규표현식(Regular Expressions) - .

메타문자 '.'은 줄바꿈 문자인 \n을 제외한 모든 문자와 매치됨을 의미합니다.

정규식	평가 대상 문자열	매치 여부	설명
a.b	aab	Yes	'a'와 'b' 사이에 'a'는 모든 문자 .에 매치
	a0b	Yes	'a'와 'b' 사이에 '0'은 모든 문자 .에 매치
	abc	No	'a'와 'b' 사이에 어떤 문자라도 있어야 한다.

* 메타문자가 아닌 일반 문자로써 .을 사용하기 위해서는 [.]과 같이 사용해야 합니다.

정규표현식(Regular Expressions) - .

```
match_check("a.b", "a0b")
match_check("a.b", "a.b")
match_check("a.b", "aaab")
match_check("a.b", "a\nb")
match_check("a.b", "a\tb")
match_check("a.b", "a   b")
match_check("a.b", "a b")
match_check("a.b", "acb")
match_check("a.b", "a\n\tb")

match_check("a[.]b", "a.b")
match_check("a[.]b", "a0b")
match_check("a[.]b", "a\nb")
```

정규표현식(Regular Expressions) - 반복

'*'은 반복을 의미하는 메타문자로, 0번 이상 반복될 수 있는다는 것을 의미합니다.

정규식	평가 대상 문자열	매치 여부	설명
ca*t	ct	Yes	'a'가 0번 반복되므로 매치
	cat	Yes	'a'가 0번 이상 반복되므로 매치 (1번)
	caaat	Yes	'a'가 0번 이상 반복되므로 매치 (3번)

'+'은 반복을 의미하는 또 다른 메타문자로, 1번 이상 반복될 수 있는다는 것을 의미합니다.

정규식	평가 대상 문자열	매치 여부	설명
ca+t	ct	No	'a'가 0번 반복되어 매치될 수 없다.
	cat	Yes	'a'가 1번 이상 반복되므로 매치 (1번)
	caaat	Yes	'a'가 1번 이상 반복되므로 매치 (3번)

정규표현식(Regular Expressions) - 반복

```
match_check("go*d", "gd")
match_check("go*d", "god")
match_check("go*d", "goooooood")

match_check("go+d", "gd")
match_check("go+d", "god")
match_check("go+d", "goooooood")

match_check("g[A-Z]+d", "gOOOOOd")
```

정규표현식(Regular Expressions) - 반복

'{ }'를 이용하면 반복할 횟수를 더 구체적으로 할 수 있습니다.

정규식	평가 대상 문자열	매치 여부	설명
ca{3}t	cat	No	3번 반복이 되지 않아 매치될 수 없다.
	caaat	Yes	3번 반복되어 매치

정규식	평가 대상 문자열	매치 여부	설명
ca{2,5}t	cat	No	2번 이상 5번 이하 반복되지 않아 매치될 수 없다.
	caat	Yes	2번 이상 5번 이하 반복되어 매치
	caaaaat	Yes	2번 이상 5번 이하 반복되어 매치

정규식	평가 대상 문자열	매치 여부	설명
ca{2,}t	caaaaaaaaat	Yes	2번 이상 반복되어 매치
ca{,5}t	caaaaat	Yes	5번 이하 반복되어 매치

정규표현식(Regular Expressions) - ?

'?'는 정규식 '{0,1}'과 동일한 역할을 한다. 즉, 해당 문자가 없거나 하나만 있는 경우에 매치된다.

정규식	평가 대상 문자열	매치 여부	설명
ca?t	ct	Yes	해당 문자가 없는 경우, 매치
	cat	Yes	해당 문자가 하나만 있는 경우, 매치
	caaat	No	'a'가 반복되어 매치될 수 없다.

정규표현식(Regular Expressions) - 반복

```
match_check("hel{2}o", "hello")

match_check("hel{1,3}o", "heo")
match_check("hel{1,3}o", "helo")
match_check("hel{1,3}o", "hello")
match_check("hel{1,3}o", "helllo")
match_check("hel{1,3}o", "hellllo")

match_check("hell?o", "helo")
match_check("hell?o", "hello")
match_check("hell?o", "helllo")
```

```
match_check("hel{,3}o", "heo")
match_check("hel{,3}o", "helo")
match_check("hel{,3}o", "hello")
match_check("hel{,3}o", "helllo")
match_check("hel{,3}o", "hellllo")

match_check("hel{3,}o", "hello")
match_check("hel{3,}o", "helllo")
match_check("hel{3,}o", "hellllo")

match_check("go{0,}d", "gd") # match_check("go*d", "gd")
match_check("go{1,}d", "gd") # match_check("go+d", "gd")
```

정규표현식(Regular Expressions)

함수	설명
re.complie("정규식")	정규식을 컴파일 하고, 컴파일된 객체를 반환
c_re_obj.match("문자열")	문자열의 처음부터 정규식과 매칭되는지 조사
c_re_obj.search("문자열")	문자열 전체에서 정규식과 매칭되는지 조사
c_re_obj.findall("문자열")	문자열 전체에서 정규식과 매칭되는 모든 문자열 반환
c_re_obj.finditer("문자열")	문자열 전체에서 정규식과 매칭되는 모든 문자열 반환

정규표현식(Regular Expressions) - match 함수

```
match_obj = re.match('[a-z]+', 'pyth0n')
```

```
import re

c_re_obj = re.compile('[a-z]+') # 알파벳 소문자가 1개 이상 있는 문자열

match_obj = c_re_obj.match('pyth0n') # 문자열의 처음부터 매치되는지 확인 [매치 객체 반환]

print(match_obj)
print(match_obj.group()) # 매치된 문자열 반환
print(match_obj.start()) # 매치된 문자열의 시작 위치 반환
print(match_obj.end()) # 매치된 문자열의 끝 위치 반환
print(match_obj.span()) # 매치된 문자열의 (시작, 끝) 위치를 튜플로 반환
```

정규표현식(Regular Expressions) - search 함수

```
match_obj = re.search('[a-z]+', '파이th0n')
```

```
import re

c_re_obj = re.compile('[a-z]+') # 알파벳 소문자가 1개 이상 있는 문자열

match_obj = c_re_obj.search('파이th0n') # 문자열 전체를 검색해 매치되는지 확인 [매치 객체 반환]

print(match_obj)
print(match_obj.group()) # 매치된 문자열 반환
print(match_obj.start()) # 매치된 문자열의 시작 위치 반환
print(match_obj.end()) # 매치된 문자열의 끝 위치 반환
print(match_obj.span()) # 매치된 문자열의 (시작, 끝) 위치를 튜플로 반환
```

정규표현식(Regular Expressions) - findall, finditer 함수

```
match_obj = re. findall('[a-z]+', '파이th0n')
```

```
import re

c_re_obj = re.compile('[a-z]+') # 알파벳 소문자가 1개 이상 있는 문자열
match_list = c_re_obj.findall('파이th0n') # 매치된 문자열을 리스트로 반환

print(match_list)
```

```
match_obj = re. finditer('[a-z]+', '파이th0n')
```

```
import re

c_re_obj = re.compile('[a-z]+') # 알파벳 소문자가 1개 이상 있는 문자열
match_iter = c_re_obj.finditer('파이th0n') # 매치된 결과를 반복 가능한 객체로 반환

print(match_iter)

for match_obj in match_iter:
        print(match_obj)
```

정규표현식(Regular Expressions) - |, ^, $

정규식	평가 대상 문자열	매치 여부	설명
Apple|Banana	ApplePie	Yes	'Apple' 매치
	BananaCake	Yes	'Banana' 매치

정규식	평가 대상 문자열	매치 여부	설명
^Love	Love of my life	Yes	첫 문자열 'Love' 매치
	My Love	No	첫 문자열이 'Love'가 아니라 매치될 수 없다.

정규식	평가 대상 문자열	매치 여부	설명
Love$	Love of my life	No	마지막 문자열이 'Love'가 아니라 매치될 수 없다.
	My Love	Yes	마지막 문자열 'Love' 매치

정규표현식(Regular Expressions) - ( )

정규식	평가대상문자열	매치여부	설명
(ABC)+	ABCDEF	Yes	ABC가한번이상반복되어매치
	ABCABC	Yes	ABC가한번이상반복되어매치

```
import re

match_obj = re.search('(\w+)\s+([0,1]{3}[-]\d{4}[-]\d{4})', '홍길동 010-1234-5678')

print(match_obj.group(0))
print(match_obj.group(1))
print(match_obj.group(2))
```

정규표현식(Regular Expressions) - 자주 사용되는 패턴

```
pattern_dict = {
    'email': '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
    'mobile': '^(01[0|1|6|7|8|9])-?([0-9]{3,4})-?([0-9]{4})$',
    'ip': '^([0-9]{1,3}\.){3}[0-9]{1,3}$',
    'url': '^https?://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
    'date': '^[0-9]{4}-[0-9]{2}-[0-9]{2}$',
    'time': '^[0-9]{2}:[0-9]{2}:[0-9]{2}$',
    'password': '^[a-zA-Z0-9!@#$%^&*]{8,}$',
}
```