파이썬 프로그래밍

14. 정규표현식

00 Contents

❖ 수업 목표

- 메타 문자의 의미와 사용법을 이해할 수 있다.
- re 모듈의 compile 메서드를 통해 패턴 객체를 생성할 수 있다.
- 패턴 객체를 통해 문자열 검색 메서드를 사용할 수 있다.
- 컴파일 옵션 4가지 종류를 설명할 수 있다.

❖ 세부 목표

00

1. 정규표현식 기초, 메타 문자

❖ 정규표현식(Regular Expression)

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용
- 주로 텍스트 검색, 매칭, 추출, 데이터 변환에 사용
- 주요 특징
  - 패턴 기반 매칭
    - 특정 문자열 패턴을 정의하여 텍스트 데이터에서 원하는 부분을 찾거나 추출
  - 다양한 활용 분야
    - 데이터 검증(이메일, 전화번호 등), 텍스트 분석, 웹 스크래이핑, 로그 파일 처리 등에 활용
  - 언어 독립적
    - 정규표현식 문법은 언어간에 유사하여 여러 언어에서 쉽게 응용 가능

1. 정규표현식 기초, 메타 문자

❖ 메타 문자

- 원래 해당 문자가 가진 뜻이 아닌 특별한 의미를 가진 문자
- 정규 표현식에 다음 메타 문자를 사용하면 특별한 의미를 지님
  - . ^ $ * + ? { } [ ] \ | ( )

1. 정규표현식 기초, 메타 문자

❖ [ ] 문자

- 문자 클래스를 정의하기 위해 사용
  - 문자 클래스는 대괄호([]) 안에 작성
    - 대괄호 내부에서 사용된 메타 문자는 특수한 기능이 없으나 일부는 예외
      - ‘[’ 와 ‘]’ 사이의 문자들과 매칭
        - 문자 클래스를 만드는 메타 문자인 [ ] 사이에는 대부분 문자 포함 가능
          - 예) 정규 표현식 [abc] 의미는 ‘a, b, c 중 한 개의 문자와 매치’ 를 의미
            - » "a"는 정규식과 일치하는 문자인 "a"가 있으므로 매치
            - » "before"는 정규식과 일치하는 문자인 "b"가 있으므로 매치
            - » "dude"는 문자 a, b, c 중 어느 하나도 포함하고 있지 않으므로 매치되지 않음

1. 정규표현식 기초, 메타 문자

❖ [ ] 문자

- [] 안의 두 문자 사이에 하이픈(-)을 사용하면 두 문자 사이의 범위를 의미
  - » 예) [a-c]는 [abc]와 동일 / [0-5]는 [012345]와 동일
  - 하이픈(-) 사용 문자 클래스의 사용 예
    - » [a-zA-Z] : 모든 알파벳
    - » [0-9] : 모든 숫자
- 대괄호 내부에서 .(dot) 나 + 는 문자 자체로 인식됨
- 대괄호 내부의 메타문자를 문자로 인식하려면 \(백슬래시)로 이스케이프 처리해야함
  - » 예) [\^]는 ^ 문자와 매칭
- 대괄호 내부에서 맨 앞에 위치한 ^는 부정(not)을 의미
  - 예를 들어 [^0-9]라는 정규 표현식은 숫자가 아닌 문자만 매치

00

1. 정규표현식 기초, 메타 문자

❖ [ ] 문자

- 자주 사용하는 문자 클래스는 별도의 표기법으로 표현 가능
  - 예) [0-9] 또는 [a-zA-Z]

- \d - 숫자와 매치. [0-9]와 동일한 표현식
- \D - 숫자가 아닌 것과 매치. [^0-9]와 동일한 표현식
- \s - 화이트스페이스(whitespace) 문자와 매치. [\t\n\r\f\v]와 동일한 표현식. 맨 앞의 빈칸은 공백 문자(space)를 의미
- \S - 화이트스페이스 문자가 아닌 것과 매치. [^ \t\n\r\f\v]와 동일한 표현식
- \w - 문자+숫자(alphanumeric)와 매치. [a-zA-Z0-9_]와 동일한 표현식
- \W - 문자+숫자(alphanumeric)가 아닌 문자와 매치. [^a-zA-Z0-9_]와 동일한 표현식
- 대문자로 사용된 것은 소문자의 반대 의미

1. 정규표현식 기초, 메타 문자

❖ .(dot) 문자

- 줄바꿈 문자인 \n을 제외한 모든 문자와 매치된다는 것을 의미
- 정규식 작성 시, re.DOTALL 옵션을 주면 .(dot) 문자와 \n 문자도 매치

- a.b
  - "a + 모든_문자 + b"
  - a와 b라는 문자 사이에 어떤 문자가 들어가도 모두 매치된다는 의미
    - "aab"는 가운데 문자 "a"가 모든 문자를 의미하는 .과 일치하므로 정규식과 매치
    - "a0b"는 가운데 문자 "0"가 모든 문자를 의미하는 .과 일치하므로 정규식과 매치
    - "abc"는 "a"문자와 "b"문자 사이에 어떤 문자라도 하나는 있어야 하는 이 정규식과 일치하지 않으므로 매치되지 않음

1. 정규표현식 기초, 메타 문자

❖ .(dot) 문자

- a[.]b
  - [] 안에 . 문자를 쓰면 여기서 .는 메타 문자가 아니라 '.' 문자 그대로를 의미.
  - "a + . + b"
    - 정규식 a[.]b는 "a.b" 문자열과 매치
    - "a0b" 문자열과는 매치되지 않음

1. 정규표현식 기초, 메타 문자

❖ * 문자

- **ca*t**
  - 반복을 의미하는 * 메타 문자
  - * 바로 앞에 있는 문자 a가 0부터 무한대까지 반복될 수 있다는 의미
    - 메모리 용량에 한계가 있어 실제로는 약 2억 개

- **다음과 같은 문자열이 모두 매치**

정규식	문자열	매치 여부	설명
ca*t	ct	Yes	"a"가 0번 반복되어 매치
ca*t	cat	Yes	"a"가 0번 이상 반복되어 매치 (1번 반복)
ca*t	caaat	Yes	"a"가 0번 이상 반복되어 매치 (3번 반복)

1. 정규표현식 기초, 메타 문자

❖ + 문자

- 반복을 나타내는 메타 문자
- +는 최소 1번 이상 반복될 때 사용
  - 즉, *가 반복 횟수가 0부터라면 +는 반복 횟수가 1부터인 것이다.

- ca+t
  - "c + a가_1번_이상_반복 + t “
  - 정규식에 대한 매치 여부는 다음 표와 같음

정규식	문자열	매치 여부	설명
ca+t	ct	No	"a"가 0번 반복되어 매치되지 않음
ca+t	cat	Yes	"a"가 1번 이상 반복되어 매치 (1번 반복)
ca+t	caaat	Yes	"a"가 1번 이상 반복되어 매치 (3번 반복)

1. 정규표현식 기초, 메타 문자

❖ {} 문자

- 반복 횟수의 범위를 제한하고 싶은 경우 사용
  - 예) 반복 횟수를 3회만 또는 1회부터 3회까지로 제한
- {m, n} 정규식 사용 시, 반복 횟수가 m부터 n까지인 문자와 매치가능
- m 또는 n 생략 가능
  - 예) {3,}처럼 사용하면 반복 횟수가 3 이상인 경우
  - {, 3}처럼 사용하면 반복 횟수가 3 이하인 경우
  - 생략된 m은 0과 동일, 생략된 n은 무한대(약 2억 개 미만)의 의미
  - 즉, {1,}은 +, {0,}은 *와 동일

1. 정규표현식 기초, 메타 문자

❖ {} 사용 정규식 예시

▪ 1. {m}

- **ca{2}t**
  - **"c + a를_반드시_2번_반복 + t “**

- **정규식 매치 표**

정규식	문자열	매치 여부	설명
`ca{2}t`	cat	No	"a"가 1번만 반복되어 매치되지 않음.
`ca{2}t`	caat	Yes	"a"가 2번 반복되어 매치

1. 정규표현식 기초, 메타 문자

❖ {} 사용 정규식 예시

▪ 2. {m, n}

- **ca{2,5}t**
  - **"c + a를_2~5회_반복 + t “**

- **정규식 매치 표**

정규식	문자열	매치 여부	설명
`ca{2,5}t`	cat	No	"a"가 1번만 반복되어 매치되지 않음.
`ca{2,5}t`	caat	Yes	"a"가 2번 반복되어 매치
`ca{2,5}t`	caaaaat	Yes	"a"가 5번 반복되어 매치

1. 정규표현식 기초, 메타 문자

❖ ? 문자

- ? 메타 문자가 의미하는 것은 {0, 1}
  - ab?c
    - "a + b가_있어도_되고_없어도_됨 + c"

  - 정규식 매치 표

정규식	문자열	매치 여부	설명
ab?c	abc	Yes	"b"가 1번 사용되어 매치
ab?c	ac	Yes	"b"가 0번 사용되어 매치

1. 정규표현식 기초, 메타 문자

❖ ^(캐럿, 삿갓모양) 문자

- 문자열의 시작을 의미
  - ^hello : 문자열이 hello로 시작해야 함
- 문자열의 시작에만 사용

❖ $ 문자

- 문자열의 끝을 의미
  - world$: 문자열이 world로 끝나야 함
- 문자열의 끝에만 사용

2. re 모듈

❖ re(regular expression) 모듈

- 파이썬은 정규 표현식 지원을 위해 re 모듈 제공
- 파이썬 표준 라이브러리
- 모듈 사용 방법
  - >>> import re
  - >>> p = re.compile('ab*')

- re.compile을 사용하여 정규 표현식을 컴파일
- re.compile의 리턴 값을 객체 p(컴파일된 패턴 객체)에 할당 후, 작업 수행
  - 정규식을 컴파일할 때 특정 옵션을 주는 것도 가능
  - 패턴 : 정규식을 컴파일한 결과

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

- 컴파일된 패턴 객체를 사용하여 문자열 검색을 수행
- 컴파일된 패턴 객체는 다음 4가지 메서드 제공

Method	목적
match()	문자열의 처음부터 정규식과 매치되는지 조사한다.
search()	문자열 전체를 검색하여 정규식과 매치되는지 조사한다.
findall()	정규식과 매치되는 모든 문자열(substring)을 리스트로 리턴한다.
finditer()	정규식과 매치되는 모든 문자열(substring)을 반복 가능한 객체로 리턴한다.

  - match, search는 정규식과 매치될 때, match 객체를 리턴하고 매치되지 않을 때는 None을 리턴.
  - match 객체란 정규식의 검색 결과로 리턴된 객체

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

- 패턴 예시
  - >>> import re
  - >>> p = re.compile('[a-z]+')

- match 메서드
  - 문자열의 처음부터 정규식과 매치되는지 조사

  - 예1)
    - >>> m = p.match("python")
    - >>> print(m)
    - <re.Match object; span=(0, 6), match='python'>
  - "python" 문자열은 [a-z]+ 정규식에 부합되므로 match 객체 리턴

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

- 패턴 예시
  - >>> import re
  - >>> p = re.compile('[a-z]+')

- match 메서드
  - 예2)
    - >>> m = p.match("3 python")
    - >>> print(m)
    - None
  - "3 python" 문자열은 문자 3이 정규식 [a-z]+에 부합되지 않으므로 None 리턴

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

▪ match 메서드

- match의 결과로 match 객체 또는 None을 리턴
- 파이썬 정규식 프로그램은 보통 다음과 같은 흐름으로 작성

```
p = re.compile(정규표현식)
m = p.match( '조사할 문자열' )
if m:
    print('Match found: ', m.group())
else:
    print('No match')
```

- match의 결과값이 있을 때만 그 다음 작업을 수행하겠다는 것

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

▪ search 메서드

- 컴파일된 패턴 객체 p를 사용하여 search 메서드 수행

- 예1)
  - >>> m = p.search("python")
  - >>> print(m)
  - <re.Match object; span=(0, 6), match='python'>

- "python" 문자열에 search 메서드를 수행하면 match 메서드를 수행했을 때와 동일하게 매치

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

- search 메서드

  - 예2)
    - >>> m = p.search("3 python")
    - >>> print(m)
    - <re.Match object; span=(2, 8), match='python'>

  - "3 python" 문자열의 첫 번째 문자는 "3"이지만, search는 문자열의 처음부터 검색하는 것이 아니라 문자열 전체를 검색하기 때문에 "3" 이후의 "python" 문자열과 매치

- match 메서드와 search 메서드는 문자열의 처음부터 검색할지 여부에 따라 다르게 사용

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

▪ findall 메서드

- 패턴([a-z]+)과 매치되는 모든 값을 찾아 리스트로 리턴

- 예)
  - >>> result = p.findall("life is too short")
  - >>> print(result)
  - ['life', 'is', 'too', 'short']

3. 정규식을 이용한 문자열 검색

❖ 문자열 검색

▪ finditer 메서드

- findall과 기능적으로 동일하나, 그 결과로 반복 가능한 객체(iterator object)를 리턴
- 반복 가능한 객체가 포함하는 각각의 요소는 match 객체임

- 예)
  - >>> result = p.finditer(" life is too short ")
  - >>> print(result)
  - <callable_iterator object at 0x01F5E390>
  - >>> for r in result: print(r)
  - ...
  - <re.Match object; span=(0, 4), match='life'>
  - <re.Match object; span=(5, 7), match='is'>
  - <re.Match object; span=(8, 11), match='too'>
  - <re.Match object; span=(12, 17), match='short'>

4. match 객체 메서드

❖ match 객체 메서드

- 기능 : 매치된 문자열 및 인덱스 확인 가능

- match 객체
  - p.match, p.search 또는 p.finditer 메서드에 의해 리턴 된 객체를 의미

- match 객체 메서드 종류 및 기능

method	목적
group	매치된 문자열을 리턴한다.
start	매치된 문자열의 시작 위치를 리턴한다.
end	매치된 문자열의 끝 위치를 리턴한다.
span	매치된 문자열의 (시작, 끝)에 해당하는 튜플을 리턴한다.

4. match 객체 메서드

❖ match 객체 메서드

▪ match 객체 메서드 예시

- >>> m = p.match("python")
- >>> m.group()
- 'python'
- >>> m.start()
- 0
- >>> m.end()
- 6
- >>> m.span()
- (0, 6)

- match 메서드 수행 결과로 리턴 된 match 객체로 start 메서드 사용 시 결과값은 항상 0 (문자열의 시작부터 조사)

4. match 객체 메서드

- search 객체 메서드 예시

```
>>> m = p.search("3 python")
>>> m.group()
'python'
>>> m.start()
2
>>> m.end()
8
>>> m.span()
(2, 8)
```

4. match 객체 메서드

❖ 모듈 단위로 수행하기

- re 모듈은 축약 형태로 사용 방법 제공

- 예)
  - 축약 전 형태
    - >>> p = re.compile('[a-z]+')
    - >>> m = p.match("python")

  - 축약 후 형태
    - >>> m = re.match('[a-z]+', "python")

  - 축약 형태는 컴파일과 match 메서드를 한 번에 수행 가능
  - 그러나, 생성한 패턴 객체를 여러 번 사용 시, re.compile을 사용하는 것이 편리

5. 컴파일 옵션

❖ 컴파일 옵션

- DOTALL(S) - .(dot)이 줄바꿈 문자를 포함해 모든 문자와 매치될 수 있게 한다.
- IGNORECASE(I) - 대소문자에 관계없이 매치될 수 있게 한다.
- MULTILINE(M) - 여러 줄과 매치될 수 있게 한다. ^, $ 메타 문자 사용과 관계 있는 옵션이다.
- VERBOSE(X) - verbose 모드를 사용할 수 있게 한다. 정규식을 보기 편하게 만들 수 있고 주석 등을 사용할 수 있게 된다.

- 옵션을 사용 시, re.DOTALL처럼 전체 옵션 이름을 써도 되고 re.S처럼 약어 사용도 가능

5. 컴파일 옵션

❖ DOTALL, S

- 도트(.) 메타 문자는 줄바꿈 문자(\n)를 제외한 모든 문자와 매치
- 만약, \n 문자도 포함하여 매치하고 싶다면 re.DOTALL 또는 re.S 옵션을 사용해 정규식 컴파일 수행
- 예)
  - >>> import re
  - >>> p = re.compile('a.b')
  - >>> m = p.match('a\nb')
  - >>> print(m)
  - None

  - 정규식이 a.b인 경우 문자열 a\nb는 매치되지 않음
  - \n은 . 메타 문자와 매치되지 않기 때문

5. 컴파일 옵션

❖ DOTALL, S

- \n 문자와도 매치되게 하려면 다음과 같이 re.DOTALL 옵션을 사용

- 예)
  - >>> p = re.compile('a.b', re.DOTALL)
  - >>> m = p.match('a\nb')
  - >>> print(m)
  - <re.Match object; span=(0, 3), match='a\nb'>

- re.DOTALL 옵션은 여러 줄로 이루어진 문자열에서 줄바꿈 문자에 상관없이 검색할 때 사용

5. 컴파일 옵션

❖ IGNORECASE, I

- re.IGNORECASE 또는 re.I 옵션은 대소문자 구별 없이 매치 수행
- 예)
  - >>> p = re.compile('[a-z]+', re.I)
  - >>> p.match('python')
  - <re.Match object; span=(0, 6), match='python'>
  - >>> p.match('Python')
  - <re.Match object; span=(0, 6), match='Python'>
  - >>> p.match('PYTHON')
  - <re.Match object; span=(0, 6), match='PYTHON'>

  - [a-z]+ 정규식은 소문자를 의미하나, re.I 옵션으로 대소문자 구별 없이 매치

5. 컴파일 옵션

❖ MULTILINE, M

- re.MULTILINE 또는 re.M 형태로 사용
- 여러 줄 모드를 활성화 함
- 문자열에 개행 문자(\n)가 포함된 경우, ^, $ 메타 문자의 동작 방식을 변경함
  - 메타문자 ^는 문자열의 처음, $는 문자열의 마지막을 의미
    - 예를 들어 정규식이 ^python인 경우, 문자열의 처음은 항상 python으로 시작해야 매치되고 만약 정규식이 python$이라면 문자열의 마지막은 항상 python으로 끝나야 매치된다는 의미이다.

5. 컴파일 옵션

❖ MULTILINE, M

- 예) # multiline.py
  - import re
  - p = re.compile("^python\s\w+")

  - data = """python one
  - life is too short
  - python two
  - you need python
  - python three"""
  - print(p.findall(data))

  - 정규식 ^python\s\w+은 python이라는 문자열로 시작하고 그 뒤에 화이트스페이스, 그 뒤에 단어가 와야 한다는 의미

  - 실행 결과
    - » ['python one']
    - » ^ 메타 문자에 의해 python 문자열을 사용한 첫 번째 줄만 매치

5. 컴파일 옵션

❖ MULTILINE, M

- ^ 메타 문자를 문자열 전체의 처음이 아니라 각 라인의 처음으로 인식시키고 싶은 경우, 사용할 수 있는 옵션이 바로 re.MULTILINE 또는 re.M
- 수정 코드) # multiline.py
  - import re
  - p = re.compile("^python\s\w+", re.MULTILINE)

  - data = """python one
    life is too short
    python two
    you need python
    python three"""
  - print(p.findall(data))

  - re.MULTILINE 옵션으로 ^ 메타 문자가 각 줄의 처음이라는 의미 가짐
  - 실행 결과
    - » ['python one', 'python two', 'python three']
    - » 즉, re.MULTILINE 옵션은 ^, $ 메타 문자를 문자열의 각 줄마다 적용

5. 컴파일 옵션

❖ VERBOSE, X

- 정규식은 일반적으로 복잡하므로 주석 또는 줄 단위로 구분하면 가독성이 좋음
- 이 경우, re.VERBOSE 또는 re.X 옵션을 사용 가능
  - 예) re.compile(r'&[#](0[0-7]+|[0-9]+|x[0-9a-fA-F]+);')

    - &: 문자 엔티티의 시작을 나타냄 (HTML 엔티티는 &로 시작)
      - » [#]: # 문자를 문자 그대로 매칭
      - » 숫자 기반의 문자 참조(8진수, 10진수, 16진수)에 사용
      - » & 뒤에 #이 오고 나서, 8진수, 10진수, 16진수 형식 중 하나가 올 수 있도록 패턴이 맞춰짐
    - (0[0-7]+|[0-9]+|x[0-9a-fA-F]+)
      - » 이 부분은 3가지 옵션 중 하나를 매칭합니다:
      - » 0[0-7]+: 8진수 문자 참조 (예: {)
      - » 0: 8진수는 반드시 0으로 시작해야 합니다.
      - » [0-9]+: 10진수 문자 참조 (예: {)
      - » x[0-9a-fA-F]+: 16진수 문자 참조 (예: &x41;)
      - » x: 16진수는 x로 시작합니다.
    - ;
      - » ;는 문자 엔티티의 끝을 나타냄 (HTML 엔티티는 항상 ;로 끝나므로 이를 매칭)

5. 컴파일 옵션

❖ VERBOSE, X

▪ 동일 역할을 하는 다른 예)

```
p= re.compile(r"""
 &[#]                    # Start of a numeric entity reference
 (
    0[0-7]+              # Octal form(8진수 형식)
  | [0-9]+               # Decimal form(10진수 형식)
  | x[0-9a-fA-F]+        # Hexadecimal form(16진수 형식)
 )
 ;                       # 후행 semicolon
""", re.VERBOSE)
```

- 첫 번째와 두 번째 예를 비교
  - 동일한 역할을 수행
  - 정규식이 복잡한 경우, 두 번째처럼 주석 표시 및 여러 줄 표현이 더 가독성이 좋음

- re.VERBOSE 옵션을 사용하면 문자열에 사용된 화이트스페이스는 컴파일할 때 제거됨 (단, [ ] 안에 사용한 화이트스페이스는 제외)

6. 역슬래시 문제

❖ 역슬래시(\) 문제

- 정규 표현식 사용 시 혼란을 주는 요소
  - 예) 어떤 파일 안에 있는 "\section" 문자열을 찾기 위한 정규식 생성 가정
    - \section
  - 이 정규식은 \s 문자가 whitespace로 해석되어 의도한 대로 매치 안됨
  - 이 표현은 다음과 동일한 의미
    - [\t\n\r\f\v]ection
  - 의도 대로 매치하고 싶다면 다음과 같이 변경
    - \\section
  - 정규식에서 사용한 \ 문자가 문자열 자체라는 것을 알려 주기 위해 역슬래시 2개를 사용해 이스케이프 처리를 해야 함
  - 따라서 위 정규식을 컴파일하려면 다음과 같이 작성할 것
    - >>> p = re.compile('\\section')

6. 역슬래시 문제

정규 표현식 사용 시 혼란을 주는 요소

추가 문제 발생 가능

- 앞선 예제처럼 정규식을 만들어서 컴파일하면 실제 파이썬 정규식 엔진에는 파이썬 문자열 리터럴 규칙에 따라 \\이 \로 변경되어 \section이 전달
- 이 문제는 이와 같은 정규식을 파이썬에서 사용할 때만 발생(파이썬의 리터럴 규칙)
  - » 파이썬에서 역슬래시를 문자열로 작성하려면 이스케이프 처리해야 함.

- 결국 정규식 엔진에 \\ 문자를 전달하려면 파이썬은 \\\\처럼 역슬래시를 4개나 사용해야 함
  - » 정규식 엔진 : 정규식을 해석하고 수행하는 모듈

원하는 결과를 도출하는 코드

- >>> p = re.compile('\\\\section')

6. 역슬래시 문제

정규 표현식 사용 시 혼란을 주는 요소

추가 문제 발생 가능

- 추가 문제에 따라 복잡해진 정규식을 해결하려면 raw string 표현법을 사용 가능
- 정규식 문자열 앞에 r 문자를 삽입하면 이 정규식은 raw string 규칙에 의해 역슬래시 2개 대신 1개만 써도 2개를 쓴 것과 동일한 의미를 가짐
- >>> p = re.compile(r'\\section')
- 만약 역슬래시를 사용하지 않는 정규식이면 r 유무에 상관없이 동일한 정규식 수행

7. 연습 문제

❖ 문제 1

- 다음 텍스트에서 전화번호를 –(하이픈)을 제외한 숫자만 추출하시오.
  - text = “My phone number is 123-456-7890”
  - numbers = re.findall(r"\d+", text)
  - print(numbers)

7. 연습 문제

❖ 문제 2

- 다음 텍스트에서 이메일 주소를 추출하시오.
  - text = "Contact us at support@example.com or sales@example.org."
  - pattern = r"[\w.-]+@[\w.-]+\.\w+"
  - emails = re.findall(pattern, text)
  - print(emails)

7. 연습 문제

❖ 문제 3

- 다음 휴대폰 전화번호가 적절하게 사용되었는지 검증하시오.

```
phone = "123-456-7890"

phone_pattern = r"^\d{3}-\d{3}-\d{4}$"
if re.match(phone_pattern, phone):
    print("Valid phone number")
else:
    print("Invalid phone number")
```

00

Notice

❖ 과제

- 1. 메타 문자의 의미를 이해하며 사용하기
- 2. re 모듈의 compile 메서드를 통해 패턴 객체 생성하기
- 3. 패턴 객체를 통해 문자열 검색 메서드 사용하기
- 4. 컴파일 옵션 4가지 사용해 보기

❖ 다음 수업 내용

- 고급함수
  - 이터레이터, 제너레이터