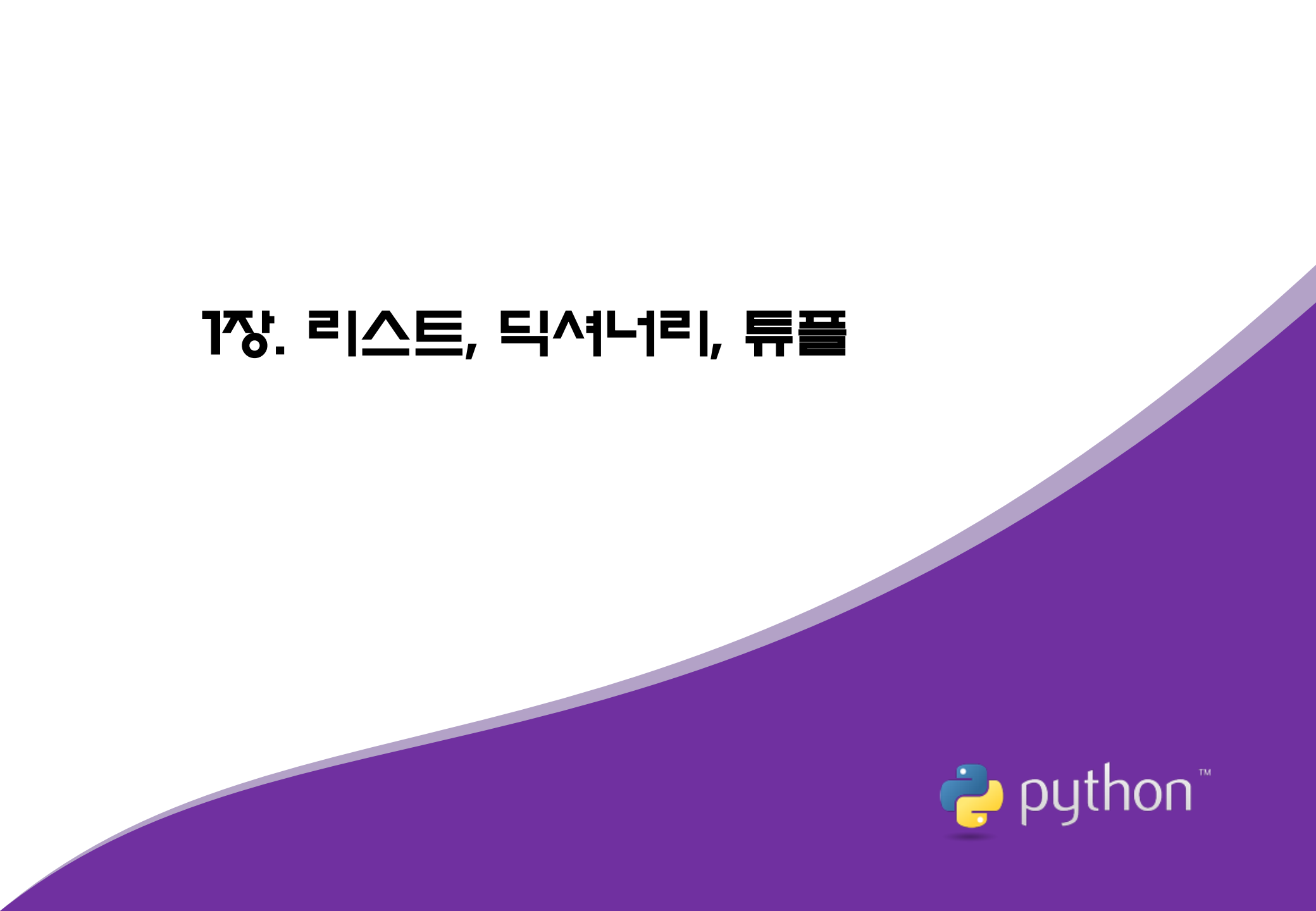
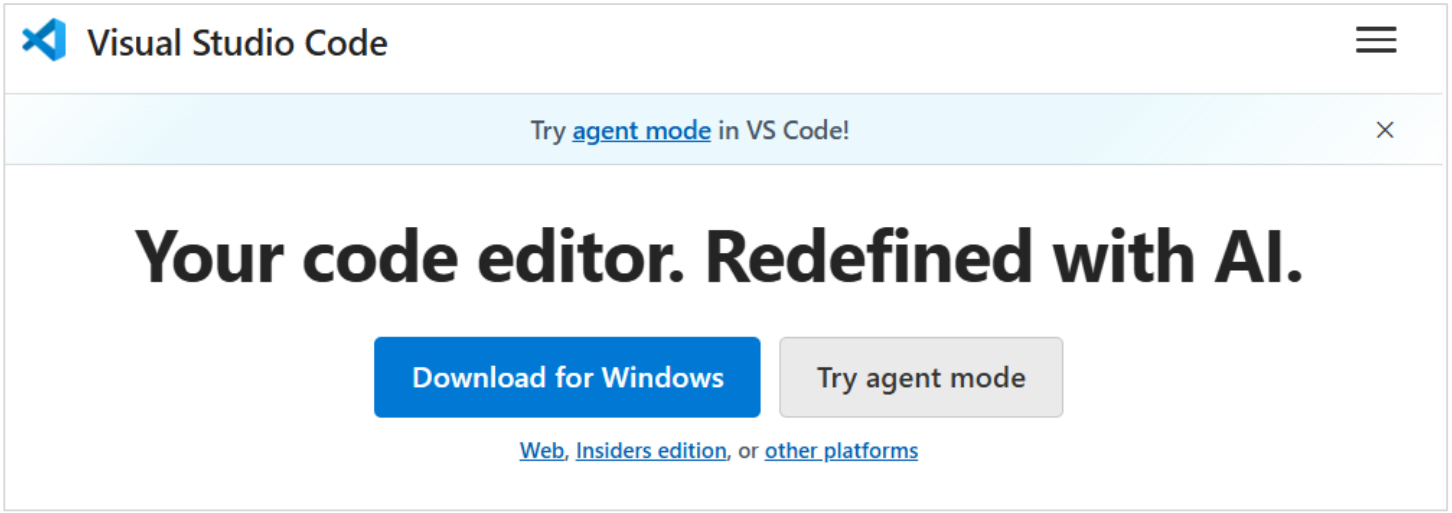
1장. 리스트, 딕셔너리, 튜플



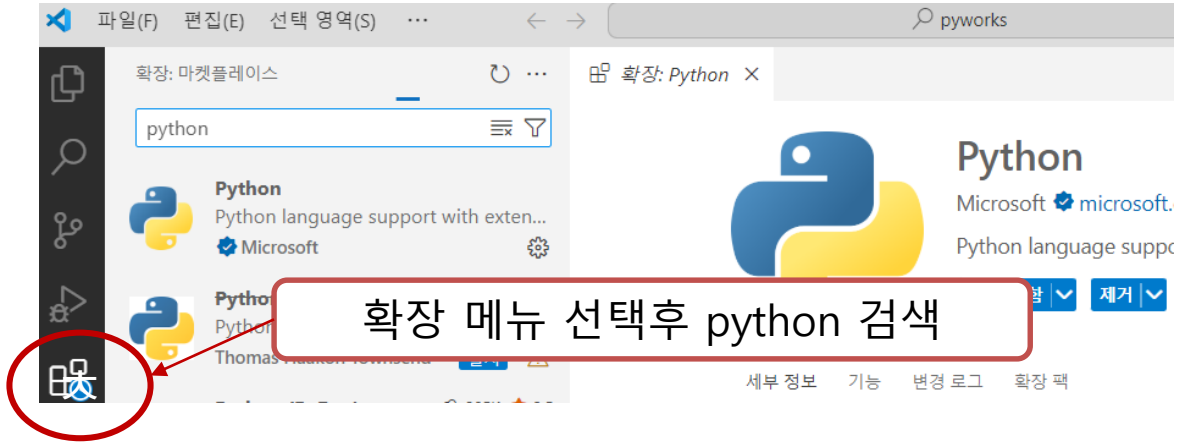
비주얼 스튜디오 코드 설치

◆ 비주얼 스튜디오 코드(VS code) 다운로드

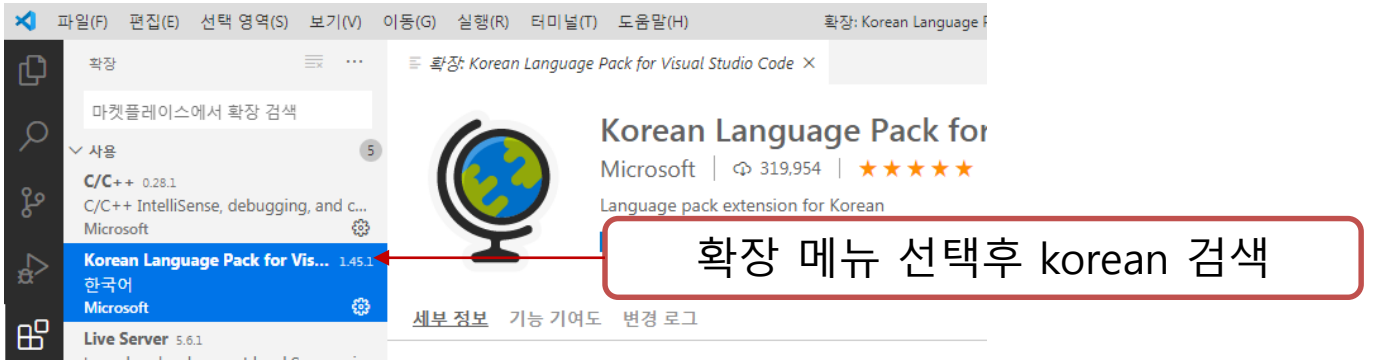


비주얼 스튜디오 코드 환경 설정

◆ **Python 언어 지원 확장 팩 설치하기**

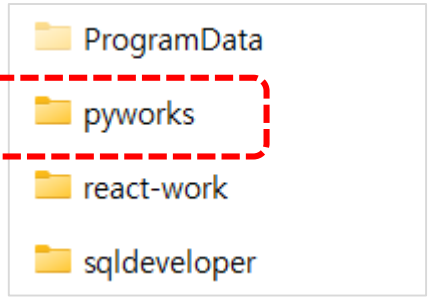


◆ 한국어 팩 설치

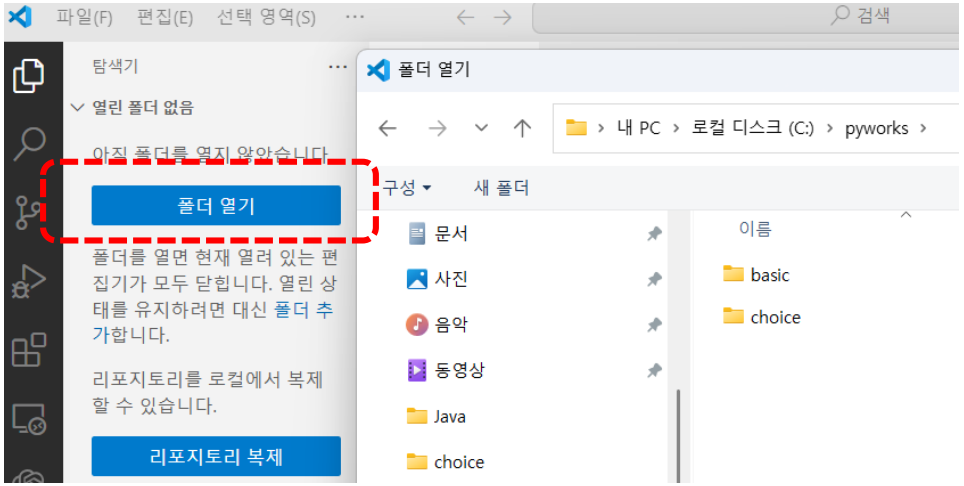


VS code – 작업 폴더 설정하기

◆ VS code – 작업 폴더 설정



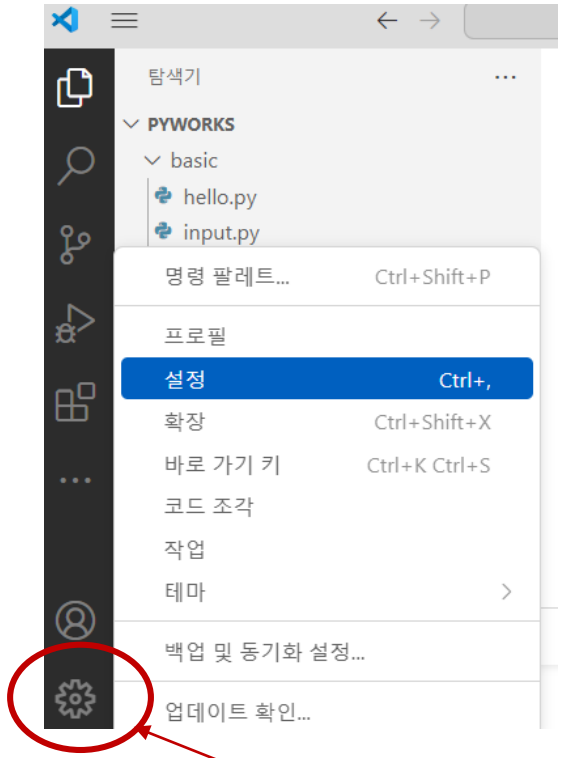
① 작업영역 폴더 만들기



② 작업영역 폴더 설정

VS code – 설정 화면

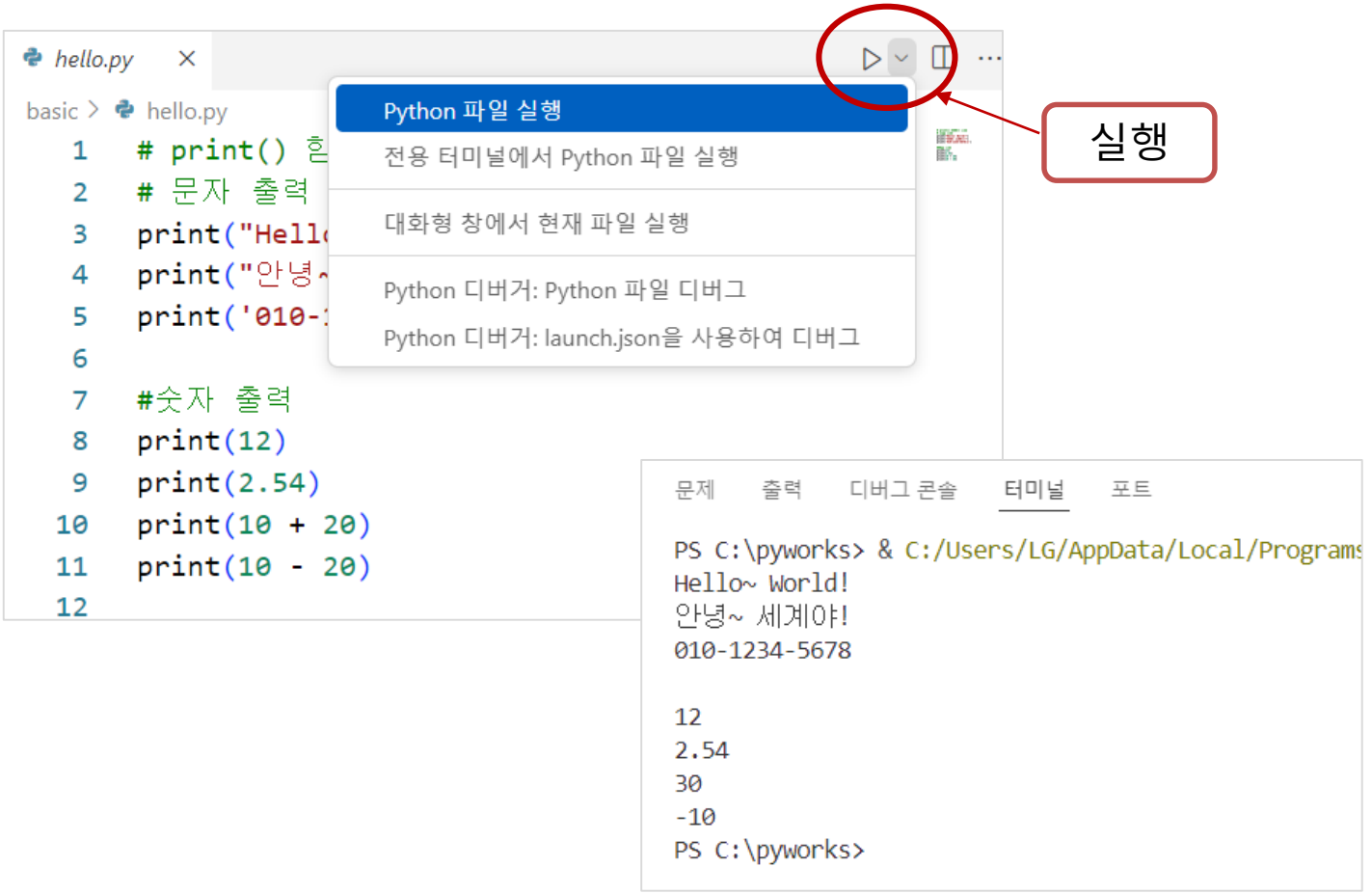
◆ VS code – 관리 도구



관리 > 설정

파이썬 파일 실행

◆ 파일 실행 및 터미널 출력



리스트(list)란?

- **리스트(list)란?**

  - 여러 개의 연속적인 값을 저장하고자 할 때 사용하는 자료형이다.
  - 변수는 1개의 값만을 저장하고 변경할 수 있다.

- **리스트의 생성**

  리스트 이름 = [요소1, 요소2, 요소3...]

  season = [“봄”, “여름”, “가을”, “겨울”]

  number = [1. 2, 3, 4, 5]

리스트(list)의 생성

- 문자형 리스트

```
# 리스트(배열) 생성
carts = ["라면", "커피", "계란", "토마토"]

# 리스트 객체 출력
print(carts)
print(type(carts)) #자료형

# 요소 접근(인덱싱)
print(carts[0])
print(carts[3])
print(carts[-1])
```

```
# 요소 수정
carts[1] = "우유"

# 요소 삭제
del carts[2]

# 리스트 출력
print(carts)
```

```
['라면', '커피', '계란', '토마토']
<class 'list'>
라면
토마토
토마토
['라면', '우유', '토마토']
```

리스트(list)의 활용

- 정수형 리스트

```
# 리스트 생성 - 요소 중복 가능
numbers = [10, 40, 30, 10, 30]

# 리스트 출력
print(numbers)

# 리스트의 크기
print("리스트의 크기: ", len(numbers))

# 요소 수정
numbers[2] = 10

# 요소 삭제
del numbers[0]

# 리스트 출력
print(numbers)
```

```
[10, 40, 30, 10, 30]
리스트의 크기:  5
[40, 10, 10, 30]
```

리스트(list) 반복 – in 사용

- **for 변수 in 리스트:**

```
# for 반복문
# in 리스트 - 리스트 존재 유무 확인
print(30 in numbers)
print(50 in numbers)
print(50 not in numbers)
print()

# 전체 요소 출력
for num in numbers:
    print(num, end=' ')
print()

# 40보다 작은 값 출력
for num in numbers:
    if num < 40:
        print(num, end=' ')
print()
```

```
True
False
True

40 10 10 30
10 10 30
```

리스트(list) 반복 – in 사용

- **if 변수 in [list]**

  리스트 내부에 값이 있으면 True, 없으면 False

```
# 음식 분류하기 - 한식, 일식, 중식
foods = ["비빔밥", "짜장면", "초밥", "김치찌게"]

for food in foods:
    if food in ["짜장면",  "짬뽕"]:
        print(f'{food}는(은) 중식입니다.')
    elif food in ["초밥", "우동"]:
        print(f'{food}는(은) 일식입니다.')
    else:
        print(f'{food}는(은) 한식입니다.')
```

```
비빔밥는(은) 한식입니다.
짜장면는(은) 중식입니다.
초밥는(은) 일식입니다.
김치찌게는(은) 한식입니다.
```

리스트(list)의 연산

◆ 리스트의 연산 – 개수, 합계, 평균 구하기

```
score = [70, 80, 50, 60, 90, 40]
total = 0
count = len(score)

for i in score:
    total += i

avg = total / count

print("개수:", count)
print("합계:", total)
print("평균:", avg)

# 내장함수 sum()과 비교
print("합계:", sum(score))
```

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

함수	기능	사용 예
append()	요소 추가	a = [1, 2, 3] a.append(4) a = [1, 2, 3, 4]
insert()	특정 위치에 추가	a = [2, 4, 5] a.insert(1,3) #1번 위치에 3 삽입 a = [2, 3, 4, 5]

pop()	요소 삭제	a = [1, 2, 3, 4, 5]
		a.pop() # 마지막 위치의 요소 제거
		a = [1, 2, 3, 4]
		a.pop(1) #1 위치의 2 제거
		a = [1, 3, 4]

remove()	특정 요소 삭제	s = ['모닝', 'BMW', 'BENZ', '스포티지']
		s.remove('BMW') #요소 직접 삭제
		s = ['모닝', 'BENZ', '스포티지']

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

함수	기능	사용 예
sort()	정렬	a = [1, 4, 2, 3] a.sort() [1, 2, 3, 4]
reverse()	뒤집기	lower = ['b', 'c', 'a'] lower.reverse() ['a', 'b', 'c']
extend(리스트)	리스트의 끝에 리스트 추가	li = ['a', 'b'], li.extend(['c','d']) ['a', 'b', 'c', 'd'],
copy()	리스트 복사	n = [1, 2, 3] m = n.copy()

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

```
a = [1, 2, 3]
print(a)

# 리스트의 크기
print(len(a))

# 맨 뒤에 추가
a.append(4)

# 1번 위치에 추가
a.insert(1, 5)

print(a)
```

```
# 맨 뒤에서 삭제
a.pop()

# 1번 위치에서 삭제
a.pop(1)

print(a)
print("--------------------")
```

```
[1, 2, 3]
3
[1, 5, 2, 3, 4]
[1, 2, 3]
--------------------
```

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

```
car = ["Sonata", "BMW", "EV3", "IONIC6"]
print(car)

# 리스트의 크기
print(len(car))

# 추가
car.append("모닝")

# 삭제
car.pop()

# 특정 요소 삭제
car.remove("BMW")
print(car)
```

```
['Sonata', 'BMW', 'EV3', 'IONIC6']
4
['Sonata', 'EV3', 'IONIC6']
```

리스트(list)의 주요 함수

- 리스트의 주요 메서드(함수)

```
# 리스트의 정렬과 뒤집기
n = [1, 4, 3, 2]
n.sort()
print(n) #[1, 2, 3, 4]

lower = ['b', 'c', 'a']
lower.reverse()
print(lower) #['a', 'c', 'b']

n2 = [1, 3, 5, 4, 2]
n2.sort()
n2.reverse()
print(n2) #[5, 4, 3, 2, 1]
```

```
# 리스트 추가
li = ['a', 'b']
li.extend(['c', 'd'])
print(li) #['a', 'b', 'c', 'd']

# 리스트 복사
n = [1, 2, 3] #원본
print(n)  #[1, 2, 3]
m = n.copy()  #복사본
print(m)  #[1, 2, 3]

# 요소 수정
n[1] = 5
print(n)  #[1, 5, 3]
print(m)  #[1, 2, 3]
```

리스트(list) 복사

◆ 리스트의 복사

```
a1 = [1, 2, 3, 4, 5]
a2 = []
a3 = []

print("a1 =", a1)

# a1을 a2에 복사
for i in a1:
    a2.append(i)

print("a2 =", a2)
```

리스트(list) 복사

◆ 리스트의 복사

```
# a1의 요소중 홀수만 저장
total = 0
for i in a1:
    if i % 2 == 1:
        a3.append(i)
        total += i

print("a3 =", a3)
print("홀수의 합계:", total)
```

```
a1 = [1, 2, 3, 4, 5]
a2 = [1, 2, 3, 4, 5]
a3 = [1, 3, 5]
홀수의 합계: 9
```

리스트(list) 내포

◆ 리스트 내포 사용하기

[표현식 for 항목(요소) in 리스트]

```
arr1 = [1, 2, 3, 4, 5]
arr2 = []
arr3 = []
arr4 = []

# arr1의 요소를 3의 배수로 저장
for i in arr1:
    arr2.append(i * 3)
print("arr2 =", arr2)

# 리스트 내포 - 3의 배수로 저장
arr3 = [i * 3 for i in arr1]
print("arr3 =", arr3)

# arr1에서 홀수만 저장
arr4 = [i for i in arr1 if i % 2 == 1]
print("arr4 =", arr4)
```

```
arr2 = [3, 6, 9, 12, 15]
arr3 = [3, 6, 9, 12, 15]
arr4 = [1, 3, 5]
```

리스트 슬라이싱

◆ 리스트의 슬라이싱(범위 검색)

```
carts = ["라면", "커피", "계란", "토마토"]

print(carts[0:4])
print(carts[:])

print(carts[0:3])
print(carts[0:-1])

print(carts[0:2])
print(carts[0:-2])
```

```
['라면', '커피', '계란', '토마토']
['라면', '커피', '계란', '토마토']
['라면', '커피', '계란']
['라면', '커피', '계란']
['라면', '커피']
['라면', '커피']
```

문자열 인덱싱, 슬라이싱

◆ 문자열은 특별한 1차원 리스트이다.

문자열(시작번호:끝번호)

※ 끝번호는 (끝번호 -1)과 같다

```
# 문자열은 1차원 리스트이다.
say = "Have a nice day"

print(say[0])
print(say[-1])
print(say[0:4])
print(say[0])
print(say[7:])
```

문자열 인덱싱, 슬라이싱

◆ 문자열은 특별한 1차원 리스트이다.

문자열(시작번호:끝번호)

※ 끝번호는 (끝번호 -1)과 같다

```
say = "Have a nice day"

print(say[0])
print(say[-1])
print(say[0:4])
print(say[0])
print(say[7:])
```

```
s = "20240621Rainy"
year = s[:4]
print(year)

day = s[4:8]
print(day)

weather = s[8:]
print(weather)
```

```
H
y
Have
H
nice day
2024
0621
Rainy
```

챗봇(chatbot)

◆ 챗봇 프로그램

단어가 포함되어 있으면 문장을 완성해주는 챗봇 프로그램 만들기

```
사용자(exit 입력시 종료): 안녕
챗봇: 안녕하세요! 반가와요!
사용자(exit 입력시 종료): 이름
챗봇: 저는 Python 챗봇입니다.
사용자(exit 입력시 종료): 날씨
챗봇: 날씨앱이나 검색 기능을 이용하세요.
사용자(exit 입력시 종료): 시간
챗봇: 죄송해요. 잘 이해하지 못했어요.
사용자(exit 입력시 종료): exit
챗봇: 대화를 종료합니다. 안녕히 가세요!
```

챗봇(chatbot)

◆ 챗봇 프로그램

```
#단어가 포함되어 있으면 출력하는 프로그램
animal = "dog"

# in 명령어 - 있다/없다를 확인하는 명령어임
print('d' in animal) #True
print('c' in animal) #False
print('g' not in animal) #False

animals = "dog cat horse"
print('cat' in animals) #True
print('cow' in animals) #False
```

챗봇(chatbot)

◆ 챗봇 프로그램

```
while True:
    user_input = input("사용자(exit 입력시 종료): ")

    if user_input == "exit":
        print("챗봇: 대화를 종료합니다. 안녕히 가세요!")
        break
    elif "안녕" in user_input:
        print("챗봇: 안녕하세요! 반가와요!")

    elif "이름" in user_input:
        print("챗봇: 저는 Python 챗봇입니다.")

    elif "날씨" in user_input:
        print("챗봇: 날씨앱이나 검색 기능을 이용하세요.")

    else:
        print("챗봇: 죄송해요. 잘 이해하지 못했어요.")
```

문자열 함수

- **문자열 함수(메서드) 정리**

메서드	설명
split()	s = 'banana, grape, kiwi' s = fruit.split(',') [구분기호로 나누고 리스트로 만듬] s ['banana', ' grape', ' kiwi']
replace()	s = 'Hello, World' s = s.replace('World', 'Korea') [문자를 변경함] 'Hello, Korea'
find()	s = "Hello" s.find('H') 0 s.find('k') -1 [문자열이 존재하는 위치 반환. 없으면 -1반환]
strip()	s = " Hi, lee" s.strip() Hi, lee

문자열 함수

- 문자열 함수(메서드)

```
fruit = "banana,grape,kiwi"
print(fruit)
print(type(fruit)) # 자료형 - str

# split(구분기호) - 문자열을 리스트로 변환해 줌
fruit = fruit.split(',')
print(fruit)   # ['banana', 'grape', 'kiwi']
print(type(fruit))  # 자료형 - list

# 인덱싱과 슬라이싱
print(fruit[0])
print(fruit[2])
print(fruit[-1])

print(fruit[0:2]) #끝인덱스-1, banana, grape
print(fruit[0:3]) #banana, grape, kiwi
print(fruit[:])   #banana, grape, kiwi
```

문자열 함수

- 문자열 함수(메서드)

```
# replace("변경전문자", "변경후문자")
s = "Hello, World"
s = s.replace("World", "Korea")
print(s) # Hello, Korea

# find(문자) - 문자의 인덱스(위치) 반환
print(s.find('H')) # 0
print(s.find('World')) # 7
print(s.find('k')) # -1

# 공백 문자 제거 - strip()
str = "  Hi~ han."
print(str.strip())   #양쪽 공백 제거
print(str.lstrip())  #왼쪽 공백 제거

str2 = "Hi~ han.  "
print(str2.rstrip()) #오른쪽 공백 제거
```

2차원 리스트(list)

- **2차원 리스트의 선언 및 생성**
  - 리스트 내부에 리스트를 가진 자료 구조이다.
  - 행과 열의 표(테이블) 형태를 이루고 있다.

리스트 이름 = [요소1, 요소2, **[요소1, 요소2, 요소3]**]

	열1	열2
행1	a[0][0]	a[0][1]
행2	a[1][0]	a[1][1]
행3	a[2][0]	a[2][1]

2차원 리스트(list)

- 2차원 리스트 생성 및 출력

```
d = [
    [10, 20],
    [30, 40],
    [50, 60]
]
print(d)
print(type(d))

# 인덱싱
print(d[0])  # 0번 인덱스 [10, 20]
print(d[1])  # 1번 인덱스 [30, 40]
print(d[0][0]) # 10
print(d[0][1]) # 20
print(d[1][0]) # 30
print(d[1][1]) # 40
```

2차원 리스트(list)

- 2차원 리스트 생성 및 출력

```
# 전체 출력 - 인덱싱 방식
for i in range(len(d)):
    for j in range(len(d[i])):
        print(d[i][j], end=' ')
    print() #행 바꿈

# 전체 출력 - 행과 요소 순회
for row in d:
    for val in row:
        print(val, end=' ')
    print()

# 행 단위로 출력
for row in d:
    print(row)

# 특정 열(1열: 인덱스 0)
for row in d:
    print(row[0])
```

```
d의 크기(행): 3
d의 크기(열): 2
d의 크기(열): 2
10 20
30 40
50 60
10 20
30 40
50 60
[10, 20]
[30, 40]
[50, 60]
[50, 60]
10
30
50
```

2차원 리스트(list)

- 2차원 리스트의 추가 및 수정

```
# 요소 추가
d.append([70, 80])
print(d)

# 요소 수정 - 40을 100으로 변경
d[1][1] = 100
# [[10, 20], [30, 100], [50, 60], [70, 80]]

# 요소 삭제 - [50, 60] 삭제
del d[2]
print(d) # [[10, 20], [30, 100], [70, 80]]

# 특정 열 삭제
for row in d:
    del row[0]
print(d)

# d 리스트 삭제
d.clear()
print(d) # [] - 빈 리스트
```

2차원 리스트(list)

- 2차원 리스트의 연산

```
d2 = [
    [10, 20],
    [30, 40],
    [50, 60, 70],
]

total = 0
count = 0

#print(len(d2[2])) #3
# 계산 1
'''
for i in range(len(d2)):
    for j in range(len(d2[i])):
        count += 1
        total += d2[i][j]
'''
```

2차원 리스트(list)

- 2차원 리스트의 연산

```
# 계산 2
for row in d2:
    for val in row:
        count += 1
        total += val

print("합계:", total)
print("개수:", count)

# 평균 = 총점 / 개수
avg = total / count
print("평균:", avg)
```

```
합계: 280
개수: 7
평균: 40.0
```

딕셔너리(Dictionary)

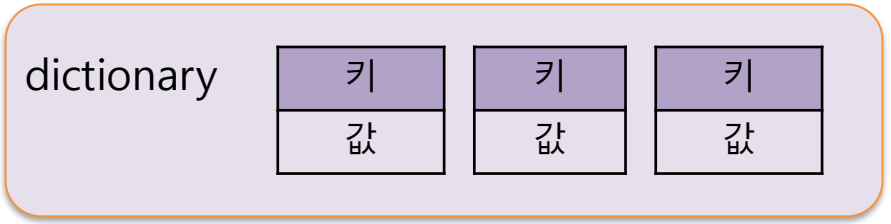
◆ 딕셔너리

리스트 처럼 여러 개의 값을 저장할 수 있고, 키(key)와 값(value)으로 대응시켜 저장하는 자료구조이다.

중괄호{ }를 사용한다.

딕셔너리 이름 = **{ 키:값, 키:값....}**

{ 'name': '한국민', 'age': 28 }



딕셔너리(Dictionary)

◆ 딕셔너리 주요 메서드

함수	사용 예
d[key] = value	d = {'Tomas':13, 'Jane':9} d['Mike'] = 10 **# 요소 추가** {'Tomas':13, 'Jane':9, 'Mike':10 }
del d[key]	del d['Jane'] **#요소 삭제** {'Tomas':13, 'Mike':10 }
d.pop(key)	d.pop('Mike') 10 {'Tomas':13}
clear()	d.clear() # d={ } 빈 딕셔너리
d.keys()	d.keys() **# 모든 키 가져오기** d_keys(['Tomas', 'Mike'])
d.values()	d.Values() **# 모든 값 가져오기** d_values([13, 10])

딕셔너리(Dictionary)

◆ 딕셔너리 생성 및 관리

```
# 딕셔너리 생성
d = {1: 'a', 2: 'b', 3: 'c'}
print(d) #{1: 'a', 2: 'b', 3: 'c'}
print(type(d)) #<class 'dict'>

print(d.keys()) #dict_keys([1, 2, 3])
print(d.values()) #dict_values(['a', 'b', 'c'])

print(d[1]) #a
print(d[3]) #c

# 수정
d[2] = 'd'
print(d) #{1: 'a', 2: 'd', 3: 'c'}
```

딕셔너리(Dictionary)

◆ 딕셔너리 생성 및 관리

```
person = {} #빈 딕셔너리
print(person) # 딕셔너리 객체 출력

# 요소 추가
person['name'] = "오상식"
person['age'] = 35
person['phone'] = "010-1234-5678"

# 객체 출력
print(person)
print(type(person)) #자료형

# 특정 요소 출력
print(person['name'])
```

딕셔너리(Dictionary)

◆ 딕셔너리 생성 및 관리

```
# 특정 요소 수정
person['name'] = "최지능"

# 요소 삭제
del person['age']

# 전체 출력
for key in person:
    print(key, ':', person[key])
```

```
{}
{'name': '오상식', 'age': 35, 'phone': '010-1234-5678'}
<class 'dict'>
오상식
name : 최지능
phone : 010-1234-5678
```

딕셔너리(Dictionary)

◆ **dictionary 메서드 사용**

```
student = {'정우': 13, '유진': 9}
print(student)  #{'정우': 13, '유진': 9}

print(student.keys())
print(student.values())

# 요소 추가
student['민영'] = 11

# 요소 수정 - 키로 검색
student['유진'] = 8

# 요소 삭제 - 키로 삭제
student.pop('정우')

print(student) # {'유진': 8, '민영': 11}

for st in student:
    print(st, ':', student[st])
```

딕셔너리(Dictionary)

- 용어 사전 만들기

```
♠ 컴퓨터 용어 사전 ♠

검색할 용어를 입력하세요(종료: q or Q): 이진수
컴퓨터가 사용하는 0과 1로 이루어진 수
검색할 용어를 입력하세요(종료: q or Q): 버그
프로그램이 적절하게 동작하는데 실패하거나 오류가 발생하는 코드 조각
검색할 용어를 입력하세요(종료: q or Q): 함수
정의된 단어가 없습니다.
검색할 용어를 입력하세요(종료: q or Q): q
프로그램 종료!
```

1. Dictionary 자료구조에 컴퓨터 용어와 정의를 저장한다.
2. 용어를 계속 반복해서 검색 할 수 있다.
3. 검색한 용어가 없으면 정의된 단어가 없음을 알려준다.
4. 검색을 종료하려면 'q' 또는 'Q'를 입력한다.

딕셔너리(Dictionary)

- 용어 사전 만들기

```
print("♠ 컴퓨터 용어 사전 ♠")
print()

# 딕셔너리 생성
dic = {
    "이진수" : "컴퓨터가 사용하는 0과 1로 이루어진 수",
    "알고리즘": "어떤 문제를 해결하기 위해 정해진 일련의 절차",
    "버그": "프로그램이 적절하게 동작하는데 실패하거나 \
오류가 발생하는 코드 조각"
}
```

딕셔너리(Dictionary)

- 용어 사전 만들기

```
while True:
    word = input("검색할 용어를 입력하세요(종료: q or Q): ")

    if word == 'q' or word == 'Q':
        print("프로그램 종료!")
        break
    else:
        if word in dic:
            definition = dic[word] #키로 값을 검색
            print(definition)
        else:
            print("정의된 단어가 없습니다.")
```

학생의 성적 관리

- **학생의 성적 통계**

  학생 4명의 국어, 영어, 수학 과목의 합계 및 평균 계산하기

```
student_list = [
    {"name":"이대한", "kor":80, "eng":80, "math":75},
    {"name":"박민국", "kor":70, "eng":65, "math":60},
    {"name":"오상식", "kor":75, "eng":70, "math":50},
    {"name":"최지능", "kor":90, "eng":95, "math":90}
]

# 첫번째 요소 검색
print(student_list[0])

print("======= 성적표 =======")
print(" 이름  국어 영어 수학")
for student in student_list:
    print(f'{student["name"]} {student["kor"]} {student["eng"]} {student["math"]}')
```

학생의 성적 관리

- 학생의 성적 통계

```
# 개인별 총점과 평균
print("== 개인별 총점과 평균 ==")
print(" 이름  총점  평균")
for student in student_list:
    total = student["kor"] + student["eng"] + student["math"]
    avg = total / 3
    print(f'{student["name"]} {total} {avg:.2f}')

# 과목별 총점과 평균
sum_subj = [0, 0, 0]
avg_subj = [0.0, 0.0, 0.0]

# 과목별 총점 계산
for student in student_list:
    sum_subj[0] += student["kor"]
    sum_subj[1] += student["eng"]
    sum_subj[2] += student["math"]
```

```
======= 성적표 =======
 이름  국어 영어 수학
이대한 80 80 75
박민국 70 65 60
오상식 75 70 50
최지능 90 95 90
== 개인별 총점과 평균 ==
 이름  총점  평균
이대한 235 78.33
박민국 195 65.00
오상식 195 65.00
최지능 275 91.67
최지능 275 91.67
```

학생의 성적 관리

- 학생의 성적 통계

```
print("== 과목별 총점 ==")
print(f'국어 총점 : {sum_subj[0]}')
print(f'영어 총점 : {sum_subj[1]}')
print(f'수학 총점 : {sum_subj[2]}')

# 과목별 평균 계산
for student in student_list:
    avg_subj[0] = sum_subj[0] / len(student_list)
    avg_subj[1] = sum_subj[1] / len(student_list)
    avg_subj[2] = sum_subj[2] / len(student_list)

print("== 과목별 평균 ==")
print(f'국어 평균 : {avg_subj[0]:.1f}')
print(f'영어 평균 : {avg_subj[1]:.1f}')
print(f'수학 평균 : {avg_subj[2]:.1f}')
```

```
== 과목별 총점 ==
국어 총점 : 315
영어 총점 : 310
영어 총점 : 310
수학 총점 : 275
== 과목별 평균 ==
국어 평균 : 78.8
영어 평균 : 77.5
수학 평균 : 68.8
```

실습 문제 – 딕셔너리

다음의 실행 결과가 나오도록 빈 칸을 작성하시오.(파일: member.py)

```
member = {"이름": "신유빈", "나이": 20, "특기": "탁구"}
result = 

print(member)
print(result)
```

☞ 실행 결과

```
{'이름': '신유빈', '특기': '탁구'}
20
```

튜플(tuple)

- **튜플(tuple)**
  - 튜플의 요소를 변경(추가, 수정, 삭제)할 수 없다.
  - 요소 추가는 초기화나 튜플간 합치기를 하면 가능함
  - 리스트처럼 동일한 방식으로 인덱싱과 슬라이싱 가능함
  - 소괄호( ) 를 사용한다.

튜플 이름 = **(요소1, 요소2....)**

```
t1 = ()
t2 = (1, )
t3 = (1, 2, 3)
t4 = ('a', 'b', 'c')
```

튜플(tuple)

- **튜플 자료형**

```
# 튜플 자료구조는 소괄호() 사용
t = (1, 2, 3)
print(t) #(1, 2, 3)
print(type(t)) #<class 'tuple'>

# 인덱싱(조회)
print(t[0]) #1
print(t[1])
print(t[2])

# 슬라이싱
print(t[1:3])  #(2, 3)
print(t[:])    #(1, 2, 3)
```

튜플(tuple)

- 튜플 자료형

```
# 수정 불가
t[1] = 4

# 삭제 불가
# del t[1]

# 요소를 1개 저장하기 - 콤머를 붙임
# t1 = (1) - 튜플이 아닌 정수임
t1 = (10,)
print(t1)
print(type(t1))

# tuple 합치기
t2 = t + t1
print(t2)  #(1, 2, 3, 10)
```

튜플의 요소는 수정 및 삭제 할 수 없다.

```
Traceback (most recent call last):
  File "d:\korea_IT\pyworks2\dict, tuple, set\tuple_ex.py",
    t[1] = 4
    ~^^^
TypeError: 'tuple' object does not support item assignment
```

튜플(tuple)

- **튜플 자료형 활용**

```
# 은행 거래 내역에 튜플 사용
transaction_history = [] #빈 리스트 생성
transaction_history.append(("입금", 10000))
transaction_history.append(("출금", 5000))
print(transaction_history) #[('입금', 10000), ('출금', 5000)]

# 요소 조회(검색)
print(transaction_history[0]) #('입금', 10000)
print(transaction_history[1]) #('출금', 5000)

# 이차원 검색
print(transaction_history[0][1]) #10000
print(transaction_history[1][1]) #5000

# 전체 요소 검색
for transaction in transaction_history:
    print(f"|{transaction[0]}|: {transaction[1]}원")
```

```
[('입금', 10000), ('출금', 5000)]
('입금', 10000)
('출금', 5000)
('출금', 5000)
('출금', 5000)
10000
5000
|입금|: 10000원
|출금|: 5000원
```

정규식 – Regular Expression

❖ ***정규표현식이란?***

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어이다. 문자열의 검색과 치환을 지원한다.

정규식 – Regular Expression

- 자주 사용하는 정규 표현식

표현식	설 명
^	정규식 시작
$	정규식 끝
^[0-9]*$	숫자
^[a-zA-Z]*$	영문 대, 소문자
^[가-힣]*$	한글
^010[-](d{3}|\d{4})[-]\d{4}$	휴대폰
^\d{6}[-][1-4]{6}$	주민등록번호

정규식 – Regular Expression

- 정규표현식에 사용되는 메타문자

메타문자	설 명(사용 예)
[ ]	대괄호는 [ ]사이의 문자들과 일치함, [x]
-	문자의 범위를 지정하는 하이픈(-), [1-4]
^	부정을 나타내는 캐럿, [^0-9]
*****	0번 이상 반복, 1번 이상 반복(+)
{m}	m은 반복횟수, {3,4} – 3개 또는 4개
( )	소괄호는 서브 클래스. 그룹을 만들 때 사용
\d	숫자 – [0-9]
\w	알파벳 + 숫자
\s	공백

정규표현식 지원 – re 모듈

- 정규 표현식 활용

1. re.compile(**'[a-z]+'**) : 정규 표현식을 컴파일 한다.
2. match("**korea**") : 문자열의 시작 부분에서 정규 표현식과 일치하는 부분을 찾음
   대소문자 구분함.

<match 객체의 주요 메서드>

메서드	기 능
group()	매치된 문자열을 돌려준다.
start()	매치된 문자열의 시작위치를 돌려준다.
end()	매치된 문자열의 끝위치를 돌려준다
span()	매치된 문자열의 (시작, 끝)에 해당하는 튜플 반환.

정규식을 사용한 문자열 검색

- 정규 표현식 활용

```
import re

pat = re.compile("[a-z]") #정규 표현식
mat = pat.match("korea")   #조사할 문자열
print(mat)
print(mat.group())
print(mat.start())
print(mat.end())
print(mat.span())

if mat:
    print('문자열 있음: ', mat.group())
else:
    print('문자열 없음')
```

정규식을 사용한 문자열 검색

- 정규 표현식 활용
  - ✓ 메타 문자 * 과 +의 차이

```
# *은 0개이상, +는 1개 이상
pat = re.compile("a*b")
mat = pat.match("b")  #aaab
# print(mat)
if mat:
    print('문자열 있음: ', mat.group())
else:
    print('문자열 없음')
```

정규식을 사용한 문자열 검색

- 유효성 검사
  - ✓ fullmatch() 함수 – 문자열 전체가 정규 표현식과 일치하는지를 찾음
  - ✓ match()는 시작 부분만 일치(첫문자)하는지 찾음

```
# 전화번호 검증
# phone_pat = re.compile('010-\d{3,4}-\d{4}')
phone_pat = re.compile("010-[0-9]{3,4}-[0-9]{4}")
mat = phone_pat.fullmatch("010-12-5678")
print(bool(mat)) #False

# 한글과 전화번호 패턴 검사
name_pat = "제갈수연";
pat = re.compile("[가-힣]{2,5}")
mat = pat.fullmatch(name_pat)
print(bool(mat)) #True
```

정규식을 사용한 문자열 검색

- 유효성 검사 예제

```
# 전화번호 패턴 유효성 검사
def validate_phone_number(phone):
    """전화번호 유효성 검사 (010-XXXX-XXXX 형식)"""
    phone_pat = re.compile("010-\d{3,4}-\d{4}")
    return bool(phone_pat.fullmatch(phone))

phone_list = [
    "010-1234-5678",  # 유효
    "010-123-4567",   # 유효
    "010-12-5678",    # 무효
    "012-1234-5678",  # 무효
    "01012345678",    # 무효
    "010-1234-567"    # 무효
]

print("=== 전화번호 검증 결과 ===")
for phone in phone_list:
    print(f"{phone}: {validate_phone_number(phone)}")
```

정규식을 사용한 문자열 검색

- 유효성 검사 예제

```
# 한글이름 패턴 유효성 검사
def validate_name(user_name):
    pattern = re.compile("^[가-힣]{2,5}$")
    return bool(pattern.fullmatch(user_name))

while True:
    user_name = input("한글 이름 입력(2~5자): ")

    if validate_name(user_name):
        print(f"이름: {user_name}")
        break
    else:
        print("올바른 한글 이름이 아닙니다. 다시 입력하세요")
```

그루핑(Grouping)

- 그루핑(Grouping)

  문자열 중에서 특정 부분의 문자열만 추출하고 싶을 때 사용한다.

  소괄호()를 사용해서 그룹을 구분한다.

group(인덱스)	설 명
group(0)	매치된 전체 문자열
group(1)	첫 번째 그룹에 해당하는 문자열
group(2)	두 번째 그룹에 해당하는 문자열
group(n)	n 번째 그룹에 해당하는 문자열

그루핑(Grouping)

- 이름과 전화번호를 구분하여 문자열 추출

```
# 그룹 - 소괄호()
phone = "jang 010-1234-5678"
pat = re.compile("(\w+)\s{1,2}(010-\d{3,4}-\d{4})")
mat = pat.match(phone)
print(mat.group())
print(mat.group(1)) #jang
print(mat.group(2)) #010-1234-5678
```

그루핑(Grouping)

- **sub()를 사용한 문자 마스킹 처리**

sub(\g <그룹 인덱스>)

```
# 폰번호 뒷 4자리 마스킹 처리
pattern = re.compile("(\w+)\s{1,2}(010-\d{3,4})-\d{4}")

print(pattern.sub("\g<1>", phone)) #jang
print(pattern.sub("\g<2>-****", phone)) #010-1234-****
```

그루핑(Grouping)

- **sub()를 사용한 문자 마스킹 처리**

sub(\g <그룹 인덱스>)

```
# 주민등록번호 마스킹 처리
data = """
kim 920815-1234567
lee 031011-4123456
"""

pat = re.compile("(\d{6})[-]\d{7}")
print(pat.sub("\g<1>-*******", data))

pat2 = re.compile("(\d{6}[-]\d{1})\d{6}")
print(pat2.sub("\g<1>******", data))
```

```
kim 920815-*******
lee 031011-*******


kim 920815-1******
lee 031011-4******
```