교수의 파이썬

02_1 리스트 축약1

창원대학교 정보통신공학과 교수 박동규

널널한 교수의 파이썬

02_1 리스트 축약1

창원대학교 정보통신공학과 교수 박동규

널널한 교수의 고급 파이썬

02_1 리스트 축약1

창원대학교 정보통신공학과 교수 박동규

널널한 교수의 고급 파이썬

02_1 리스트 축약1

창원대학교 정보통신공학과 교수 박동규

리스트 축약

리스트 축약

- 리스트의 축약 표현(list comprehension) 이란 ?

리스트 축약

- 리스트의 축약 표현(list comprehension) 이란 ?
  - 반복가능 객체를 이용하여 쉽게 리스트를 생성할 수 있는 기능

리스트 축약

- 리스트의 축약 표현(list comprehension) 이란 ?
  - 반복가능 객체를 이용하여 쉽게 리스트를 생성할 수 있는 기능
  - map, filter 기능 구현 가능

리스트 축약

- 리스트의 축약 표현(list comprehension) 이란 ?
  - 반복가능 객체를 이용하여 쉽게 리스트를 생성할 수 있는 기능
  - map, filter 기능 구현 가능
  - 람다식의 본체가 될 식을 그대로 사용하기 때문에 따로 람다함수를 정의할 필요가 없다.

리스트 축약

- 리스트의 축약 표현(list comprehension) 이란 ?
  - 반복가능 객체를 이용하여 쉽게 리스트를 생성할 수 있는 기능
  - map, filter 기능 구현 가능
  - 람다식의 본체가 될 식을 그대로 사용하기 때문에 따로 람다함수를 정의할 필요가 없다.
  - 리스트 뿐만아니라 집합과 같은 반복가능한 모든 객체에 대해 적용 할 수 있다.

리스트 축약의 문법

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

반복가능 객체의 요소에
대해 적용되는 수식
⇨ 맵 기능

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

“반복가능 객체의 순환하는 기능”

반복가능 객체의 요소에
대해 적용되는 수식
⇨ 맵 기능

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

"반복가능 객체의 순환하는 기능"

반복가능 객체의 요소에
대해 적용되는 수식
⇨ 맵 기능

필터의 기능,
생략 가능

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

"반복가능 객체의 순환하는 기능"

반복가능 객체의 요소에
대해 적용되는 수식
⇨ 맵 기능

필터의 기능,
생략 가능

리스트 축약의 문법

[{표현식} for {변수} in {반복자/연속열} if {조건 표현식}]

"반복가능 객체의 순환하는 기능"

반복가능 객체의 요소에
대해 적용되는 수식
⇨ 맵 기능

필터의 기능,
생략 가능

[{표현식} for {변수} in {반복자/연속열}]

연습

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x for x in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [x ** 2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

맵과 람다 함수를 이용한 리스트의 제곱 구하기

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

```
a = [x**2 for x in range(1,8)]
print(a)
```

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

```
a = [x**2 for x in range(1,8)]
print(a)
```

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

```
a = [x**2 for x in range(1,8)]
print(a)
```

위의 세 코드의 실행결과는 모두 동일하다.

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

```
a = [x**2 for x in range(1,8)]
print(a)
```

맵과 람다 함수를 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = list(map(lambda x: x**2, a)) # 리스트의 각 요소에 대하여 람다 함수 적용
print(a)
```

리스트 축약 표현식을 이용한 리스트의 제곱 구하기

```
a = [1, 2, 3, 4, 5, 6, 7] # 연속된 값을 가지는 리스트
a = [x**2 for x in a] # 리스트의 각 요소에 대하여 x**2를 적용
print(a)
```

리스트 축약 표현식과 range를 이용한 리스트의 제곱 구하기

```
a = [x**2 for x in range(1,8)]
print(a)
```

실행 결과

```
[1, 4, 9, 16, 25, 36, 49]
```

Lab

문자열 각각을 대문자로 바꾸는 기능

```
st = 'Hello World'
s_list = [x.upper() for x in st] # 문자열 각각에 대해 upper() 메소드 적용
print(s_list)
```

문자열 각각을 대문자로 바꾸는 기능

```
st = 'Hello World'
s_list = [x.upper() for x in st] # 문자열 각각에 대해 upper() 메소드 적용
print(s_list)
```

문자열 각각을 대문자로 바꾸는 기능

```
st = 'Hello World'
s_list = [x.upper() for x in st] # 문자열 각각에 대해 upper() 메소드 적용
print(s_list)
```

반복자로는
문자열을 이용하는 것도
가능하다

문자열 각각을 대문자로 바꾸는 기능

```
st = 'Hello World'
s_list = [x.upper() for x in st] # 문자열 각각에 대해 upper() 메소드 적용
print(s_list)
```

반복자로는
문자열을 이용하는 것도
가능하다

실행 결과

문자열 각각을 대문자로 바꾸는 기능

```
st = 'Hello World'
s_list = [x.upper() for x in st] # 문자열 각각에 대해 upper() 메소드 적용
print(s_list)
```

반복자로는
문자열을 이용하는 것도
가능하다

실행 결과

```
['H', 'E', 'L', 'L', 'O', ' ', 'W', 'O', 'R', 'L', 'D']
```

Lab

map( ) 함수와 리스트 축약 표현

map( ) 함수와 리스트 축약 표현

- 리스트 축약 표현의 장점
  - map( ) 함수는 map 객체를 생성하지만, 축약 표현을 사용하면 리스트 객체를 바로 생성할 수 있다
  - 간결한 표현으로 강력한 기능을 사용할 수 있다
  - map( ) 함수를 사용하는 것보다 속도가 더 빠르다
  - 필터링까지 할 수 있어 filter( ) 함수의 역할도 수행한다.

감사합니다.