Unit 32.

Status	완료
담당자	
마감일	
완료일	@2022년 11월 28일

Unit 32. 람다 표현식 사용하기

- 람다 표현식으로 익명 함수를 만드는 방법이 있다

32.1 람다 표현식으로 함수 만들기

- 람다 표현식은 다음과 같이 lambda에 매개변수를 지정하고 :(콜론) 뒤에 반환값으로 사용할 식을 지정
- **lambda 매개변수들: 식**

```
>>> lambda x: x + 10
<function <lambda> at 0x02C27270>
```

이 상태에서는 함수를 호출할 수 없다 왜냐하면 람다 표현식은 이름 없는 함수를 만들기 때문

- lambda로 만든 익명 함수를 호출하려면 다음과 같이 람다 표현식을 변수에 할당

```
>>> plus_ten = lambda x: x + 10
>>> plus_ten(1)
11
```

- lambda x: x + 10은 매개변수 x 하나를 받고, x에 10을 더해서 반환한다는 뜻

32.1.1 람다 표현식 자체를 호출하기

- 람다 표현식은 변수에 할당하지 않고 람다 표현식 자체를 바로 호출할 수 있다.
- **(lambda 매개변수들: 식)(인수들)**

```
>>> (lambda x: x + 10)(1)
11
```

32.1.2 람다 표현식 안에서는 변수를 만들 수 없다

- 람다 표현식 안에서는 새 변수를 만들 수 없다는 점

```
>>> (lambda x: y = 10; x + y)(1)
SyntaxError: invalid syntax
```

- 람다 표현식 바깥에 있는 변수는 사용가능

```
>>> y = 10
>>> (lambda x: x + y)(1)
11
```

32.1.3 람다 표현식을 인수로 사용하기

- 함수의 인수 부분에서 간단하게 함수를 만들기 위해서 대표적인 예가 map

```
>>> def plus_ten(x):
...     return x + 10
...
```

```
>>> list(map(plus_ten, [1, 2, 3]))
[11, 12, 13]
```

32.2 람다 표현식과 map, filter, reduce 함수 활용하기

32.2.1 람다 표현식에 조건부 표현식 사용하기

- **lambda 매개변수들: 식1 if 조건식 else 식2**
- 람다 표현식 안에서 조건부 표현식을 쓸때는 : (콜론)을 붙이지 않는다.
- if를 사용한다면 else는 무조건 사용해야 한다 아니면 에러
- elif를 사용할 수 없다 대신 if를 연속으로 사용해야 한다
- **lambda 매개변수들: 식1 if 조건식1 else 식2 if 조건식2 else 식3**

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(map(lambda x: str(x) if x % 3 == 0 else x, a))
[1, 2, '3', 4, 5, '6', 7, 8, '9', 10]
```

32.2.2 map에 객체를 여러 개 넣기

- map은 리스트 등의 반복 가능한 객체를 여러 개 넣을 수도 있습니다
- 리스트 두 개를 처리할 때는 람다 표현식에서 lambda x, y: x * y처럼 매개변수를 두 개로 지정하면 된다. 그리고 map에 람다 표현식을 넣고 그다음에 리스트 두 개를 콤마로 구분해서 넣어준다. 즉, 람다 표현식의 매개변수 개수에 맞게 반복 가능한 객체도 콤마로 구분해서 넣어주면 된다

```
>>> a = [1, 2, 3, 4, 5]
>>> b = [2, 4, 6, 8, 10]
>>> list(map(lambda x, y: x * y, a, b))
[2, 8, 18, 32, 50]
```

32.2.3 filter 사용하기

- filter는 반복 가능한 객체에서 특정 조건에 맞는 요소만 가져오는데, filter에 지정한 함수의 반환값이 True일 때만 해당 요소를 가져온다
- **filter(함수, 반복가능한객체)**

```
>>> def f(x):
...     return x > 5 and x < 10
...
>>> a = [8, 3, 2, 10, 15, 7, 1, 9, 0, 11]
>>> list(filter(f, a))
[8, 7, 9]
```

32.2.4 reduce 사용하기

- reduce는 반복 가능한 객체의 각 요소를 지정된 함수로 처리한 뒤 이전 결과와 누적해서 반환하는 함수
- **from functools import reduce**
- **reduce(함수, 반복가능한객체)**

```
>>> def f(x, y):
...     return x + y
...
>>> a = [1, 2, 3, 4, 5]
>>> from functools import reduce
>>> reduce(f, a)
15
```