공학적 문제해결 기법

12주차 1차시

만들어져 있는 프로시저 사용하기1

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

I. 개요

모듈 사용하기1

II. 학습 개요

1) 학습목표

객체의 복사본을 생성하는 COPY 모듈을 이용해 복사본의 결과를 같은지
확인해보고, 난수를 만드는 방법, 객체들의 리스트를 무작위로 섞는 방법 등을 알아본다.

2) 학습목차(세부 목차)

- COPY 모듈을 사용해 복사본 만들기
- KEYWORD와 RANDOM 모듈

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 함수와 클래스를 더 쉽게 사용하기 위해 모듈을 사용한다.
- turtle 모듈은 화면에 그림을 그리기 위해 캔버스를 생성
- 모듈을 프로그램에 임포트하면 거기에 담긴 모든 것들을 사용할 수 있다.
- copy 모듈은 객체의 복사본을 생성하는 함수들을 가지고 있다.
- 가끔은 객체의 복사본을 생성하고 새로운 객체를 생성하기 위해 그 복사본을 사용하는 것이 유용
- 여러 단계를 가지는 객체를 생성할 때 유용

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 예를 들어 name, number_of_legs, color 매개변수를 받는 _init_ 함수가 있는 Animal 클래스가 있다고 하자.

```
>>> class Animal:
        def __init__(self, name, number_of_legs, color):
            self.name = name
            self.number_of_legs = number_of_legs
            self.color = color
```

- 다리가 6개 있는 핑크색 히포그리프 값을 가진 harry인 Animal 클래스의 새로운 객체를 생성해 보자

```
>>> harry = Animal('hippogriff', 6, 'pink')
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 만약에 6개의 다리를 가진 핑크색 히포그리프들의 무리를 만들고 싶다면?
- 앞의 코드를 여러 번 반복해서 생성할 수도 있겠지만, copy 모듈에 있는 copy를 사용할 수 있다.

```
>>> import copy
>>> harry = Animal('hippogriff', 6, 'pink')
>>> harriet = copy.copy(harry)
>>> print(harry.species)
hippogriff
>>> print(harriet.species)
hippogriff
```

- harry와 harriet 이 두 개의 객체들은 완전히 다른 객체지만, 같은 종일뿐

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 객체가 매우 많이 복잡해진다면 복사할 수 있다는 것이 훨씬 더 유용함
- Animal 객체들의 리스트를 생성하고 copy 할 수도 있다.

```
>>> harry = Animal('hippogriff', 6, 'pink')
>>> carrie = Animal('chimera', 4, 'green polka dots')
>>> billy = Animal('bogill', 0, 'paisley')
>>> my_animals = [ harry, carrie, billy ]
>>> more_animals = copy.copy(my_animals)
>>> print(more_animals[0].species)
hippogriff
>>> print(more_animals[1].species)
chimera
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 원본 리스트인 my_animals에 있는 Animal 객체 중의 하나의 종을 변경한다면?
- hippogriff를 ghoul로
- more_animals에 있는 것도 변경될까요?

```
>>> my_animals[0].species = 'ghoul'
>>> print(my_animals[0].species)
ghoul
>>> print(more_animals[0].species)
```

결과는?

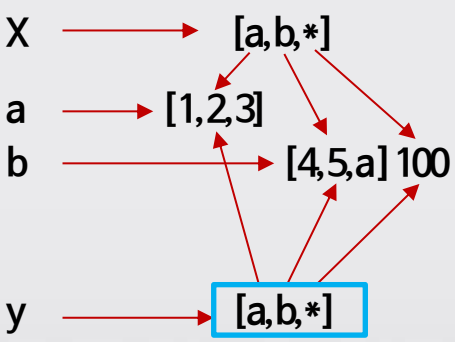
12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- my_animals의 종만 바꾸었는데 왜 두 리스트 모두의 종이 변경될까요?
- 종이 바뀐 이유는 copy가 사실상 얕은 복사(shallow copy)를 만든 것이기 때문

```
>>> import copy
>>> a = [1,2,3]
>>> b = [4,5,a]
>>> x = [a,b,100]
>>> y = copy.copy(x)
```



- copy된 y가 별도의 리스트 객체를 갖지만 실제 내용은 x의 내용과 다르지 않음
- 메인 리스트 객체는 복사했지만,
  리스트 안에 있는 각각의 객체들은 복사한 것이 아님

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 새로운 동물을 마지막 리스트(my_animals)에 추가하면 복사본 (more_animals)은 어떻게 될까요?

```
>>> sally = Animal('sphinx', 4, 'sand')
>>> my_animals.append(sally)
>>> print(len(my_animals))
4
>>> print(len(more_animals))
3
```

- my_animals에 새로운 동물을 추가하면 그 리스트의 복사본인 more_animals 에는 추가되지 않는다.

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- copy 모듈에 있는 다른 함수인 deepcopy는 복사된 객체 안에 있는 모든 객체의 복사본을 실제로 생성함

```
>>> import copy
>>> a = [1,2,3]
>>> b = [4,5,a]
>>> x = [a,b, 100]
>>> y = copy.deepcopy(x)
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 1 - COPY 모듈을 사용해 복사본 만들기

copy 모듈

- 위 그림에서 알 수 있듯이 y는 x의 전체를 복사
- deepcopy를 사용하면 모든 객체들의 복사본을 가진 완벽한 새로운 리스트를 갖게 됨
- 원본 Animal 객체들 중의 하나를 변경하면 새로운 리스트에 있는 객체들에는 영향을 미치지 않음

```
>>> more_animals = copy.deepcopy(my_animals)
>>> my_animals[0].species = 'wyrm'
>>> print(my_animals[0].species)
wyrm
>>> print(more_animals[0].species)
ghoul
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 2 - KEYWORD와 RANDOM 모듈

키워드(keyword)

- if와 else, for처럼 파이썬 언어 자체의 일부인 어떤 단어임
- keyword 모듈은 iskeyword라는 이름의 함수와 kwlist라는 변수를 가지고 있다.
- iskeyword 함수는 어떤 문자열이 파이썬 키워드일 경우에 참을 반환
- kwlist 변수는 모든 파이썬 키워드들의 목록을 반환
- 키워드들은가끔씩변경된다.
- 새로운버전(또는이전버전)은또다른키워드들을가지고있을수있기에이때유용하게사용된다.

```
>>> import keyword
>>> print(keyword.iskeyword('if'))
True
>>> print(keyword.kwlist)
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 2 - KEYWORD와 RANDOM 모듈

RANDOM 모듈로 랜덤 숫자 얻기 – randint

- random 모듈은 컴퓨터에게 '숫자 하나를 뽑아줘.'라고 말하는 것
- 난수(random number)를 생성하는 데 유용한 많은 함수들을 가지고 있다.
- 가장 유용한 함수는 randint와 choice, shuffle
- 난수를 뽑기 위해 randint 사용하기

```
>>> import random
>>> print(random.randint(1, 100))
>>> print(random.randint(100, 1000))
>>> print(random.randint(1000, 5000))
```

- 실행해보면, 결과가 매번 다르게 나올 수 있다.
- 숫자의 범위 안에서 발생할 수 있는 수는 많으니깐

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 2 - KEYWORD와 RANDOM 모듈

RANDOM 모듈로 랜덤 숫자 얻기 - randint

- while 루프를 사용해 간단한 숫자 맞추기 게임을 만들 때 randint를 사용

```
import random
num = random.randint(1, 100)
while True:
    print('Guess a number between 1 and 100')
    guess = input()
    i = int(guess)
    if i == num:
        print('You guessed right')
        break
    elif i < num:
        print('Try higher')
    elif i > num:
        print('Try lower')
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 2 - KEYWORD와 RANDOM 모듈

RANDOM 모듈로 랜덤 숫자 얻기 - choice

- 리스트에서 항목을 무작위로 뽑기 위해 choice 사용하기
- 주어진 범위에서 난수를 선택하는 것이 아니라
- 리스트에서 무작위로 항목을 뽑고 싶다면 choice를 사용

```
〉〉〉 import random
 〉〉〉 desserts = [ 'ice cream', 'pancakes', 'brownies', 'cookies',
'candy' ]
 〉〉〉 print(random.choice(desserts))
```

12주차 1차시 : 만들어져 있는 프로시저 사용하기 1

Ⅲ. 학습 2 - KEYWORD와 RANDOM 모듈

RANDOM 모듈로 랜덤 숫자 얻기 - shuffle

- 리스트를 섞기 위해 shuffle 사용하기
- shuffle 함수는 리스트에 있는 항목을 섞는 역할

```
>>> import random
>>> desserts = [ 'ice cream', 'pancakes', 'brownies', 'cookies',
'candy' ]
>>> random.shuffle(desserts)
>>> print(desserts)
['cookies', 'candy', 'pancakes', 'ice cream', 'brownies']
```

- 만약에 카드 게임을 만들고 있다면 카드들을 나타내는 리스트를 섞을 때 이 함수를 사용할 수 있다.

TONGMYONG UNIVERSITY
동명대학교

『 이 콘텐츠는 2014학년도 학부교육 선도대학 육성사업(ACE)에 의하여 개발한 것임 』