2018년도 가을학기 SCSC 알고리즘 실습 2주차

문현수, munhyunsu@cs-cnu.org

Thursday September 13, 2018

1 개요

BBC에서 제작한 다큐멘터리 'The Secret Rules of Modern Living: Algorithms'에는 "13 chocolates and 1 chilli" 게임이 소개된다. 마지막으로 고추를 꺼내면 패배하는 이 게임은 아래와 같은 룰로 진행된다.

1. 항아리에 13개의 초콜릿과 1개의 고추를 넣은 후 게임 시작!
2. 선수 2명이 선공/후공을 정한다. 본 설명서에서는 P1이 선공, P2가 후공이라 간주한다.
3. P1이 항아리에 남은 초콜릿 중 총 1–3개 꺼낸다(예. 2개).
4. P2가 항아리에 남은 초콜릿 중 총 1–3개 꺼낸다(예. 1개).
5. 3–4번을 반복하며 초콜릿이 없어 고추를 꺼내야하는 사람이 패배한다.

이 게임은 단순하지만 필승 규칙이 있다. 다큐멘터리에서는 그 규칙을 소개하며 알고리즘도 이와 같은 규칙, 연산임을 알려준다. 이번 실습에서는 13 초콜릿 1 고추 게임을 구현한다. 이를 통하여 우리는 아래와 같은 학습 목표를 이룬다.

- Data types 이해
- Flow control 이해 및 활용
- Function 이해 및 활용
- Class 이해 및 활용

2 13 Chocolates and 1 Chili

우리는 이 게임을 구현하기 위하여 다양한 Python 3 활용법을 알아야한다. 이 실습 설명서는 실습을 차례차례 따라갈 수 있도록 구성되었다.

2.1 게임 준비

게임 시작을 하기 위해 항아리에 초콜릿 13개와 1개의 고추를 넣어야한다. 프로그램으로 이것을 표현하는 방법을 고민해보자. 다양한 방법이 있겠지만 가장 쉬운 방법은 '정수'로 초콜릿과 고추를 저장하는 것이다. 고추는 무조건 1개이므로 하나의 '변수'만으로 초콜릿과 고추를 표현할 수 있다.

이 항아리는 단순히 '정수'를 저장하고 있는 변수가 아니다(다른말로 초콜릿과 고추를 담고만 있는 것이 아니다.). 선수들은 항아리에서 초콜릿을 '꺼내야'한다. 객체 내부에 다른 변수를 저장하고(멤버 변수), 객체가 할 수 있는 동작을 표현(멤버 함수)하는 가장 편한 방법은 '클래스'다. 우리는 이 항아리를 클래스로 구현하여 활용할 것이다.

항아리의 테스트 코드는 아래와 같다.

Listing 1: ChocolateJar Testcode

```
1  import unittest
2
3  from chocolate_jar import ChocolateJar
4
5
6  class ChocolateJarTest(unittest.TestCase):
7      def test_create(self):
8          jar = ChocolateJar(13)
9          self.assertEqual(14, jar.chocolates)
10
11     def test_take(self):
12         jar = ChocolateJar(13)
13         jar.take_chocolate(3)
14         self.assertEqual(11, jar.chocolates)
15         jar.take_chocolate(2)
16         self.assertEqual(9, jar.chocolates)
17         jar.take_chocolate(1)
18         self.assertEqual(8, jar.chocolates)
19
20
21 if __name__ == '__main__':
22     unittest.main(verbosity=2)
```

테스트 코드는 설계도이며 곧 구현될 함수 혹은 클래스가 동작해야할 방향이다. 따라서 우리는 테스트 코드가 의도하는 바를 이해하고 구현할 줄 알아야한다. 테스트 코드에 따르면 `ChocolateJar`를 생성하였을 때 인자로 입력한 값에 1을 더한 값이 chocolate라는 멤버 변수에 저장된다(`test_create()`). 또한 `take_chocolate()`라는 멤버 함수를 호출하면 인자로 입력된 만큼 chocolate 멤버 변수에서 빠진다.

설계를 만족하는 ChocolateJar 클래스는 아래와 같다.

Listing 2: ChocolateJar Class

```
1  class ChocolateJar(object):
2      def __init__(self, chocolates=13):
3          self.chocolates = chocolates+1
4
5      def take_chocolate(self, hands):
6          self.chocolates = self.chocolates - hands
```

`ChocolateJar` 클래스에서 `__init__`은 일종의 생성자로 클래스가 객체로 생성될 때마다 수행되는 멤버 함수다. 이 생성자의 인자로 `self`와 chocolates가 전달될 수 있다. Python 3에서 모든 클래스 멤버 함수는 `self`를 첫번째 인자로 가진다. JAVA에서의 `this`와 같은 기능을 하는 이 키워드는 객체 자기 자신을 가르키며 Python 3의 철학처럼 '명시적으로' 전달됨을 표시한다. 또한 2번째 인자인 chocolates는 기본 값으로 13을 가지게 하여 만약 객체 생성시에 인자가 전달되지 않아도 값을 가지게 하였다. 위의 예제에서는 13 초콜릿 1 고추 게임이기 때문에 기본값을 13으로 하였다.

2.2 사용자 입력

게임 준비를 마쳤으니 선공과 후공을 정하고 사용자에게 몇개의 초콜릿을 꺼낼지 입력받아야한다. 본래 이 게임은 2명의 사람이 즐겨야하나 이번 실습에서는 선공은 사용자가 후공은 컴퓨터가 하도록 하자(무조건 이기는 법!).

아래 코드는 13 초콜릿 1 고추 게임 중 사용자 입력을 받는 부분의 코드다. `input()`으로 사용자에게서 입력을 받는다. `.strip()`로 문자열 앞뒤에 입력되었을 수 있는 whitespace를 제거한다. `int()`로 문자열을 정수형으로 '변환'한다.

Listing 3: Chocolate game code - user input

```
25          take = input('몇␣개의␣초콜릿을␣빼까요 ?:␣')
26          take = int(take.strip())
27          print('플레이어는␣{0}␣개의␣초콜릿을␣꺼냈습니다.'.
                format(take))
28          jar.take_chocolate(take)
```

2.3 컴퓨터 무작위 선택

본래 이 게임은 2명의 사용자가 즐겨야하지만 지금은 편의상 1명의 사용자, 1개의 컴퓨터가 대결한다. 따라서 후공을 맡고있는 컴퓨터가 몇개를 꺼낼 것인지 무작위로 결정하여 항아리에서 초콜릿을 꺼내야한다. Python 3에서는 무작위로 숫자를 뽑거나 선택할 때 `random`이라는 라이브러리를 사용한다. 소스코드 상단 2번 라인에 `import random`을 입력하여 random 라이브러리를 불러오자.

게임 규칙은 1–3개의 초콜릿 혹은 고추를 꺼내는 것이므로 무작위로 1–3을 선택해도 된다. 하지만, 상식적으로 패배하지 않기 위해 초콜릿 1개, 고추 1개가 남았을 경우 초콜릿 1개만 꺼내도록 해야한다. 이것을 위하여 무작위 선택의 범위를 결정해주는 알고리즘이 필요하다. 아래는 컴퓨터의 무작위 선택 코드다. 게임의 즐거움을 위하여 컴퓨터가 선택할 때 `sleep()`으로 대기하도록 하였다. 이를 사용하기 위하여 소스코드 상단 1번 라인에 `import time`을 입력하여 time 라이브러리를 활용할 수 있도록 하자.

Listing 4: Chocolate game code - computer random take

```
34          max_take = min(jar.chocolates - 1, 3)
35          take = random.randint(1, max_take)
36          print(' 컴퓨터 ␣ 고민중 ... ', end='␣')
37          time.sleep(take)
38          print(' 컴퓨터는 ␣{0} 개의 ␣ 초콜릿을 ␣ 꺼냈습니다 .'.format
                (take))
39          jar.take_chocolate(take)
```

2.4 승리자가 나올 때까지 반복

사용자와 컴퓨터가 번갈아가며 초콜릿을 빼냈다. 우리는 이 작업을 승리자가 나올 때까지, 즉 항아리에 아무것도 없을 때까지 반복해야 한다. '조건'을 기준으로 반복할 때에는 `while`이 효율적이다. 따라서 순서를 계속 반복하도록 하며 사용자에게 게임 진행도를 알려주기 위하여 아래와 같이 코드할 수 있다.

Listing 5: Chocolate game code - some line of main()

```
15  def main():
16      jar = ChocolateJar(13)
17      print(' 게임을 ␣ 시작합니다 .')
18      print(' 항아리에 ␣{0} 개의 ␣ 초콜릿와 ␣1 개의 ␣ 고추가 ␣
              있습니다 .'.format(13))
19      print('1~3 개의 ␣ 초콜릿 ␣ 혹은 ␣ 고추를 ␣ 꺼낼 ␣수␣ 있으며 ␣
            고추를 ␣ 꺼내면 ␣ 패배합니다 .')
20      print(' 시작 !')
21      show_jar(jar)
22      turn = 1
23      while jar.chocolates > 0:
24          print('----+␣{0}번␣턴!␣----+'.format(turn))
```

2.5 승리/패배 조건 확인

프로그래밍에서 '조건'을 확인한 후 '분기'할 때는 '분기문'을 사용한다. 13 초콜릿 1 고추 게임에서도 승리/패배 조건 확인을 위하여 '분기문'을 사용해야한다. 이것을 모두 고려하면 게임은 완성된다. 아래 코드는 지금까지의 코드를 모두 합한 `main()`함수다. `show_jar()`함수는 아직 구현하지 않았으니 기다리자.

Listing 6: Chocolate game code - main() entire lines

```
15 def main():
16     jar = ChocolateJar(13)
17     print('게임을 시작합니다.')
18     print('항아리에 {0} 개의 초콜릿와 1 개의 고추가
           있습니다.'.format(13))
19     print('1~3 개의 초콜릿 혹은 고추를 꺼낼 수 있으며
           고추를 꺼내면 패배합니다.')
20     print('시작!')
21     show_jar(jar)
22     turn = 1
23     while jar.chocolates > 0:
24         print('----+{0}번턴!----+'.format(turn))
25         take = input('몇 개의 초콜릿을 뺄까요 ?:')
26         take = int(take.strip())
27         print('플레이어는 {0} 개의 초콜릿을 꺼냈습니다.'.
               format(take))
28         jar.take_chocolate(take)
29         show_jar(jar)
30         if jar.chocolates == 1:
31             print('플레이어 승리!')
32             break
33
34         max_take = min(jar.chocolates - 1, 3)
35         take = random.randint(1, max_take)
36         print('컴퓨터 고민중 ... ', end=' ')
37         time.sleep(take)
38         print('컴퓨터는 {0} 개의 초콜릿을 꺼냈습니다.'.format
               (take))
39         jar.take_chocolate(take)
40         show_jar(jar)
41         if jar.chocolates == 1:
42             print('컴퓨터 승리!')
43             break
44         turn = turn+1
```

```
45
46
47 if __name__ == '__main__':
48     main()
```

2.6 항아리 출력

항아리에 몇개의 초콜릿과 고추가 남았는지 숫자로 알려주는 것은 시각적으로 재미가 없다. 게임은 시각적으로도 재미있어야 그 가치가 높아진다. 따라서 항아리에 남아있는 초콜릿과 고추의 수를 Bar로 알려주도록 하자. 아래 '함수'는 함수임에도 불구하고 사용자 출력을 담당하는 함수이기 때문에 테스트 코드를 작성하지 않았다. 원한다면 어떻게 테스트 코드를 작성할 수 있을지 고민해보고 테스트해보자.

Listing 7: Chocolate game code - show_jar()

```
1  import time
2  import random
3
4  from chocolate_jar import ChocolateJar
5
6
7  def show_jar(jar):
8      chocolates = jar.chocolates
9      print('■', end='')
10     for index in range(0, chocolates-1):
11         print('□', end='')
12     print('')
```

2.7 더 높은 난이도를 원한다면...

게임은 지속적으로 업데이트되며 더 재미있어진다. 이 게임도 다양한 버전으로 강화되어 더 재미있어질 수 있다. 원한다면 다양한 아이디어를 구현해보도록하자. 혹시 아이디어가 필요하다면 아래를 구현하는 것으로 시작해도 좋다.

- 사람 2명이 대결하도록 변경
- 초콜릿의 개수 변경(게임시작 할 때 입력)
- 컴퓨터가 선공을 잡은 후 컴퓨터가 '무조건' 이기도록 변경

3 Blackjack game

Twenty-one이라고도 불리는 블랙잭(Blackjack)은 카지노에서 가장 널리 행해지는 카드 게임이다. 딜러와 플레이어가 대결하는 게임으로 플레잉카드(포커카드라고도 부른다.)마다 1부터 11까지의 점수를 놓고 21점을 맞추는 것을 목적으로 한다. 따라서 Suit(Spades, Diamonds, Hearts, Clubs)는 의미가 없으며, Rank(Ace – King)만 점수로서의 의미를 가진다. 플레잉카드를 구매하면 들어있는 Joker를 제외하고 52장의 카드로 게임한다.

3.1 플레이어

블랙잭에서 플레이어는 딜러를 제외하고 최소 1명이 필요하다. 쉽게 예상할 수 있듯 딜러는 컴퓨터가 맡도록 제작할 것이고 사용자는 게임을 진행하는 사람이 될 것이다. 게임을 더욱 발전시키기 위하여 네트워크 통신을 이용한 다중 사용자를 지원할 수 있을 것이다. 하지만 이번 실습에서 우리는 컴퓨터(딜러)와 노는 것만 제작한다.

3.2 플레잉카드

플레잉카드는 4가지의 Suit(Spades, Diamonds, Hearts, Clubs), 13가지의 Rank(Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King)으로 구성된다. 따라서 총 52장의 카드가 있다. 52장의 카드 한 벌을 덱(Deck)이라고 부른다. 플레이어가 딜러에게 받은 카드들을 핸드(Hand)라고 부르며 블랙잭에서는 핸드의 합으로 대결한다.

3.3 게임 방법

게임이 시작되면 사용자는 딜러에게 카드를 요청할 '수' 있다. 사용자는 자신이 가진 핸드의 합이 21이 넘지 않도록 추가 카드를 요청할 '수' 있다. 21이 넘어버려서 22 이상이 되면 무조건 패배가 되니 주의해야한다. 만약 다수의 플레이어가 있을 경우 21를 넘지 않은 플레이어중 21에 가장 가까운 플레이어가 승리한다.

4 Blackjack Program

블랙잭 게임을 만들기위한 기본 지식은 모두 학습했으니 본격적으로 블랙잭 프로그램을 만들 시간이다. 우리가 만들어야하는 것은 아래와 같다.

- 카드 클래스: 카드 한장, 한장을 나타냄. suit와 rank를 멤버 변수로 가짐.
- 딜러 클래스: 카드를 52장 만들어 덱을 소유. 사용자에게 카드를 한장씩 제공함.

4.1 Card Class

요구사항에 대한 Card class 테스트 코드는 아래와 같다.

Listing 8: Card class test code

```
1  import unittest
2
3  from card import Card
4
5
6  class CardTest(unittest.TestCase):
7      def test_create(self):
8          suit = 'Hearts'
9          rank = 'Ace'
10         card1 = Card(suit, rank)
11         self.assertEqual((suit, rank), card1.get_value())
12
13     def test___eq__(self):
14         card1 = Card('Spades', 'Queen')
15         card2 = Card('Spades', 'Queen')
16         self.assertEqual(card1, card2)
17         card3 = Card('Hearts', 'Queen')
18         self.assertNotEqual(card1, card3)
19
20
21 if __name__ == '__main__':
22     unittest.main(verbosity=2)
```

카드는 suit와 rank를 멤버 변수로 가진다. 우리가 아직 학습하지 않은 몇가지 내부 함수(Python 3가 내부적으로 활용하는 함수)가 필요하므로 Skeleton code를 기반으로 코드한다.

Listing 9: Card class skeleton code

```
1  class Card(object):
2      def __eq__(self, other):
3          if (self.suit == other.suit) and (self.rank == other.rank):
4              return True
5          return False
6
7      def __repr__(self):
8          return '({0}, {1})'.format(self.suit, self.rank)
9
10     def __init__(self, suit, rank):
```

```
11          pass
12
13      def get_value( self ):
14          pass
```

4.2 Dealer Class

요구사항에 대한 Dealer class 테스트코드는 아래와 같다.

Listing 10: Dealer class test code

```
1   import unittest
2
3   from card import Card
4   from dealer import Dealer
5
6
7   class DealerTest(unittest.TestCase):
8       def setUp(self):
9           suits = ['Spades', 'Diamonds', 'Hearts', 'Clubs']
10          ranks = ['Ace',
11                   '2', '3', '4', '5', '6', '7', '8', '9', '10',
12                   'Jack', 'Queen', 'King']
13          self.deck = list()
14          for suit in suits:
15              for rank in ranks:
16                  self.deck.append(Card(suit, rank))
17
18      def tearDown(self):
19          del self.deck
20
21      def test__create_deck( self ):
22          dealer = Dealer()
23          deck = dealer._create_deck()
24          self.assertEqual(self.deck, deck)
25
26      def test_get_card( self ):
27          dealer = Dealer()
28          for index in range(0, len(self.deck)):
29              hand = dealer.get_card()
30              self.assertIn(hand, self.deck)
31
```

```
32      def test_get_values_k10(self):
33          hands = [Card('Hearts', 'King'),
34                   Card('Spades', '10')]
35          dealer = Dealer()
36          self.assertEqual(20, dealer.get_values(hands))
37
38      def test_get_values_q08(self):
39          hands = [Card('Diamonds', 'Queen'),
40                   Card('Clubs', '8')]
41          dealer = Dealer()
42          self.assertEqual(18, dealer.get_values(hands))
43
44      def test_get_values_aj10(self):
45          hands = [Card('Spades', 'Jack'),
46                   Card('Hearts', 'Ace'),
47                   Card('Diamonds', '10')]
48          dealer = Dealer()
49          self.assertEqual(21, dealer.get_values(hands))
50
51      def test_get_values_aa09(self):
52          hands = [Card('Hearts', 'Ace'),
53                   Card('Diamonds', 'Ace'),
54                   Card('Clubs', '9')]
55          dealer = Dealer()
56          self.assertEqual(21, dealer.get_values(hands))
57
58      def test_get_values_akqj(self):
59          hands = [Card('Spades', 'Ace'),
60                   Card('Hearts', 'King'),
61                   Card('Diamonds', 'Queen'),
62                   Card('Clubs', 'Jack')]
63          dealer = Dealer()
64          self.assertEqual(31, dealer.get_values(hands))
65
66
67  if __name__ == '__main__':
68      unittest.main(verbosity=2)
```

딜러는 Card를 52개 만들어 덱을 구축하며 섞어둔다. 클래스 전체를 구현하는 것은 우리의 학습 목적과 맞지 않으므로 Skeleton code를 기반으로 코드한다.

Listing 11: Dealer class skeleton code

```
1  import random
2
3  from card import Card
4
5  SUIT = ['Spades', 'Diamonds', 'Hearts', 'Clubs']
6  RANK = ['Ace',
7          '2', '3', '4', '5', '6', '7', '8', '9', '10',
8          'Jack', 'Queen', 'King']
9  VALUE = {'Ace': 11,
10           '2': 2, '3': 3, '4': 4, '5': 5, '6': 6,
11           '7': 7, '8': 8, '9': 9, '10': 10,
12           'Jack': 10, 'Queen': 10, 'King': 10}
13
14
15 class Dealer(object):
16     def __init__(self):
17         self._create_deck()
18         random.shuffle(self.deck)
19
20     def _create_deck(self):
21         self.deck = list()
22         for suit in SUIT:
23             for rank in RANK:
24                 self.deck.append(Card(suit, rank))
25         return self.deck
26
27     def get_card(self):
28         pass
29
30     def get_values(self, cards):
31         pass
```

4.3 main()

Card와 Dealer class를 구현하여 테스트를 통과하였다면 블랙잭 게임을 만들 수 있다. 다양한 방법으로 활용가능하므로 자신만의 블랙잭 게임을 만들어보도록 하자. 아래 예제는 블랙잭 게임의 main()함수 예제다.

Listing 12: Blackjack program main

```
1  from dealer import Dealer
2
```

```
3
4  def main():
5      dealer = Dealer()
6      user_input = None
7      hands = list()
8
9      while user_input != 'n':
10         print('----+ 당신의 패 ----+')
11         for hand in hands:
12             print('{0} {1}'.format(hand.suit, hand.rank))
13         user_input = input(' 카드를 더 받을까요 ?(y/n):')
14         user_input = user_input.strip()
15         user_input = user_input.lower()
16         if user_input == 'y':
17             hands.append(dealer.get_card())
18
19     print('----+ 당신의 점수 ----+')
20     print(' 총점 :{0}'.format(dealer.get_values(hands)))
21
22
23 if __name__ == '__main__':
24     main()
```

5 과제

실습에서 우리는 13 초콜릿 1 고추 게임을 만들어보았다. 그리고 Data types, Flow control, Function, Class를 모두 이용하는 좋은 예제로 블랙잭 게임을 살펴보았다. 이번 실습의 과제는 구현하지 않은 블랙잭 게임을 완성하는 것이다.

5.1 과제 목표

- Card class 구현
- Dealer class 구현
- Blackjack game 구현

5.2 제출 관련

- 마감 날짜: 2018. 09. 19. 23:59:59
- 딜레이: 1일당 20% 감점

- 제출 방법: 과목 사이버캠퍼스
- 제출 형식: 과제 리포트 PDF(HWP, DOC 받지 않음!), 소스코드(구현한 .py 만 추가할 것)를 압축한 .zip 파일
- 리포트에 포함해야하는 내용: 목표, 목표를 위해 알아야하는 것, 해결 방법, 결과, (선택)느낀점 or전달할 말
- 제출 파일 제목: AL_201550320_문현수_02.zip(파일명 준수!)

5.3 조교 연락처

- 문현수
- munhyunsu@cs-cnu.org
- 공학5호관 633호 데이터네트워크연구실
- 이메일, 연구실 방문