Exercise Sheet 06 Solutions – Sorting and Object Oriented Programming

Sebastian Höffner Aline Vilks

Deadline: Mon, 22 May 2017 08:00 +0200

Exercise 1: Sorting Bond movies

File: `movies.py`

```
class Movie:
    """A movie contains a title, a release year, an actor, budget and revenue.
    """

    def __init__(self, title, year, actor, revenue, budget):
        self.title = title
        self.year = year
        self.actor = actor
        self.revenue = revenue
        self.budget = budget

    def __str__(self):
        return self.title + ' (' + str(self.year) + ', ' + self.actor + ')'

    def income(self):
        """Returns the income of the movie.

        Returns:
            The income is the difference between the revenue and the budget.
        """
        return self.revenue - self.budget


def movie_sort(movies):
    """Sorts a list of movies by their release date using bubble sort.

    Args:
        movies: a list of movies.
```

```
    Returns:
        A sorted list of movies.
    """
    sorted_movies = movies.copy()
    swapped = True
    while swapped:
        swapped = False
        for i in range(1, len(sorted_movies[1:])):
            if sorted_movies[i - 1].year > sorted_movies[i].year:
                sorted_movies[i], sorted_movies[i - 1] = \
                    sorted_movies[i - 1], sorted_movies[i]
                swapped = True
    return sorted_movies


def movie_sort_bonus(movies, key=lambda movie: movie.year):
    """Sorts a list of movies by their release date using bubble sort.

    Args:
        movies: a list of movies.
        key: the key to sort by.

    Returns:
        A sorted list of movies.
    """
    sorted_movies = movies.copy()
    swapped = True
    while swapped:
        swapped = False
        for i in range(1, len(sorted_movies[1:])):
            if key(sorted_movies[i - 1]) > key(sorted_movies[i]):
                sorted_movies[i], sorted_movies[i - 1] = \
                    sorted_movies[i - 1], sorted_movies[i]
                swapped = True
    return sorted_movies


def read_movies(filename):
    """Reads a csv file and returns a list of movies.

    Args:
        filename: The file to read.

    Returns:
        A list of movies.
```

```
    """
    movies = []
    with open(filename, 'r') as movie_file:
        for line in movie_file.read().splitlines()[1:]:
            title, year, actor, budget, revenue = line.split(',')
            movies.append(Movie(title, int(year), actor, float(budget) * 1e6,
                                float(revenue) * 1e6))
    return movies


def print_movies(movies):
    """Prints a list of movies.

    Args:
        A list of movies.
    """
    for movie in movies:
        print(movie)


def bonus(movies):
    """This function tries to solve the bonus exercise on a list of movies.

    You can safely ignore it if you didn't try to solve that!

    Args:
        movies: The movie list to sort.
    """
    try:
        def actor_selector(movie):
            """Flips first and last name of an actor."""
            return ' '.join(movie.actor.split(' ')[::-1])

        sorted_by_actor = movie_sort_bonus(movies, key=actor_selector)
        print('\nSorted list (by actor, first 10):')
        print_movies(sorted_by_actor[:10])

        sorted_by_income = movie_sort_bonus(movies,
                                            key=lambda movie: movie.income())
        print('\nSorted list (by income, first 10):')
        print_movies(sorted_by_income[:10])
    except TypeError:
        pass
```

```python
def main():
    """Reads a list of movies from bond.csv and sorts it."""
    movies = read_movies('bond.csv')

    print('Original list (first 10):')
    print_movies(movies[:10])

    sorted_movies = movie_sort(movies)
    print('\nSorted list (by year, first 10):')
    print_movies(sorted_movies[:10])

    bonus(movies)


if __name__ == '__main__':
    main()
```

Output:

```
Original list (first 10):
A View to a Kill (1985, Roger Moore)
Casino Royale (1967, David Niven)
Casino Royale (2006, Daniel Craig)
Diamonds Are Forever (1971, Sean Connery)
Die Another Day (2002, Pierce Brosnan)
Dr. No (1962, Sean Connery)
For Your Eyes Only (1981, Roger Moore)
From Russia with Love (1963, Sean Connery)
GoldenEye (1995, Pierce Brosnan)
Goldfinger (1964, Sean Connery)

Sorted list (by year, first 10):
Dr. No (1962, Sean Connery)
From Russia with Love (1963, Sean Connery)
Goldfinger (1964, Sean Connery)
Thunderball (1965, Sean Connery)
Casino Royale (1967, David Niven)
On Her Majesty's Secret Service (1969, George Lazenby)
Diamonds Are Forever (1971, Sean Connery)
Live and Let Die (1973, Roger Moore)
The Man with the Golden Gun (1974, Roger Moore)
The Spy Who Loved Me (1977, Roger Moore)

Sorted list (by actor, first 10):
Die Another Day (2002, Pierce Brosnan)
GoldenEye (1995, Pierce Brosnan)
```

```
The World Is Not Enough (1999, Pierce Brosnan)
Tomorrow Never Dies (1997, Pierce Brosnan)
Diamonds Are Forever (1971, Sean Connery)
Dr. No (1962, Sean Connery)
From Russia with Love (1963, Sean Connery)
Goldfinger (1964, Sean Connery)
Never Say Never Again (1983, Sean Connery)
Thunderball (1965, Sean Connery)

Sorted list (by income, first 10):
Casino Royale (1967, David Niven)
On Her Majesty's Secret Service (1969, George Lazenby)
Dr. No (1962, Sean Connery)
From Russia with Love (1963, Sean Connery)
The Man with the Golden Gun (1974, Roger Moore)
Diamonds Are Forever (1971, Sean Connery)
Live and Let Die (1973, Roger Moore)
Licence to Kill (1989, Timothy Dalton)
Goldfinger (1964, Sean Connery)
A View to a Kill (1985, Roger Moore)
```

Exercise 2: Castles crashed... again!

File: knights.py

```python
class Knight:

    def __init__(self, level=31, strength=20, magic=20, defense=30, agility=7):
        """Instantiates a knight with its default attributes.

        Args:
            level: Defaults to 31.
            strength: Defaults to 20.
            magic: Defaults to 20.
            defense: Defaults to 30.
            agility: Defaults to 7.
        """
        self.level = level
        self.strength = strength
        self.magic = magic
        self.defense = defense
        self.agility = agility
        self.health = self.maximum_health()
```

```
    def maximum_health(self):
        """Returns the maximum health of this knight."""
        return 69 + 3 * self.level + 28 * self.defense

    def is_alive(self):
        """Returns true if this knight is alive."""
        return self.health > 0

    def normal_attack(self, other):
        """Calculates the attack damage with a normal attack and lets other
        take damage accordingly.

        Args:
            other: The attack target.
        """
        damage = (3 + self.strength + 0.1 * self.level) // 1
        other.take_damage(damage)

    def strong_attack(self, other):
        """Calculates the attack damage with a strong attack and lets other
        take damage accordingly.

        Args:
            other: The attack target.
        """
        damage = (5 + 1.15 * self.strength + 0.1 * self.level) // 1
        other.take_damage(damage)

    def throw_attack(self, other):
        """Calculates the attack damage with a throw attack and lets other
        take damage accordingly.

        Args:
            other: The attack target.
        """
        damage = (10 + 1.2 * self.strength + 0.1 * self.level) // 1
        other.take_damage(damage)

    def arrow_attack(self, other):
        """Calculates the attack damage with an arrow attack and lets other
        take damage accordingly.

        Args:
            other: The attack target.
        """
```

```
        damage = 2 + self.agility
        other.take_damage(damage)

    def take_damage(self, attack_damage):
        """Reduces the health by the defense modified attack_damage.

        Args:
            attack_damage: The attack damage.
        """
        self.health -= (attack_damage * (1.2 - 0.01 * self.defense) + 0.5) // 1


if __name__ == '__main__':
    red = Knight(level=32, strength=32)
    blue = Knight()

    # This creates lists of functions!
    red_combo = [Knight.strong_attack] * 2 + [Knight.normal_attack]
    blue_combo = [Knight.normal_attack] * 3 + [Knight.throw_attack]

    # The dashing blue knight attacks twice with his arrow before the battle
    # starts...
    blue.arrow_attack(red)
    blue.arrow_attack(red)

    # Each round consists of red attacking (odd half_rounds) and blue attacking
    # (even half_rounds). An attack goes through the respective combo and calls
    # each attacks therein with the attacker as self and the attacked as other.
    half_round = 1
    while red.is_alive() and blue.is_alive():
        for attack in red_combo if half_round & 1 else blue_combo:
            # The * unpacks. Calling attack(*[red, blue]) is equivalent to
            # attack(red, blue).
            # As this was hard to read, especially with the [::1] or [::-1],
            # we changed this to a more readable version (which is probably
            # better anyways:
            if half_round & 1:
                attack(red, blue)
            else:
                attack(blue, red)
            # Original:
            # attack(*([red, blue][::1 if half_round & 1 else -1]))
        half_round += 1

    print('Round:', half_round // 2)
```

```
    print('Red alive?', red.is_alive(), 'HP:', red.health)
    print('Blue alive?', blue.is_alive(), 'HP:', blue.health)
```

Output:

```
Round: 9
Red alive? True HP: 173.0
Blue alive? False HP: -24.0
```