S06 - Programación Funcional Built-in

September 14, 2017

1 Programación funcional

Santiago Torres y Andrés Fernández

1.1 ¿Por qué?

- Existen lenguajes de programación que utilizan este paradigma.
- Procesamiento de información optimizada.

1.2 Actividad 6 2017-1

Pueden encontrar el enunciado completo de esta actividad aquí.

- `movies.txt`: Archivo de texto que contiene la información de las películas que puedes ver, de la forma: IDPelícula, Nombre Pelicula, Rating, Fecha de estreno, género_1, género_2, ..., género_n (es una cantidad variable de géneros).
- `cast.txt`: Archivo de texto que alberga la información de los artistas que trabajaron en las películas. Está distribuido de la forma: Nombre Película, Nombre Artista, Nombre Personaje.

```
In [1]: from collections import namedtuple

        Cast = namedtuple("Cast", ["movie", "name", "character"])
```

- Cada película debe tener su propio `id` el cual debe estar implementado con **generadores**. El `id` es distinto al que trae la base de datos, para que puedas distinguirla con tu programa.

```
In [2]: def get_id():
            i = 0
            while True:
                yield i
                i += 1

In [3]: class Movie:

            id_ = get_id()
```

```
            def __init__(self, title, rating, release, *genres):
                self.id = next(Movie.id_)
                self.title = title
                self.rating = float(rating)
                self.release = release
                self.genres = list(genres)
```

```
In [4]: parser = lambda line: line.rstrip('\n').split(',')

        MOVIES = list()
        with open("movies.txt") as file:
            MOVIES = [Movie(*(parser(line)[1:])) for line in file]

        CASTS = list()
        with open("cast.txt") as file:
            CASTS = [Cast(*parser(line)) for line in file]
```

- Crear el método `popular` que dado un número, retorne todas las películas que tienen un rating superior a dicho valor.

```
In [5]: def popular(movies, n):
                """ Retorna una lista de peliculas con rating sobre n"""
                return filter(lambda x: x.rating >= n, movies)
```

```
In [6]: for top in popular(MOVIES, 20):
            print(top.title)
```

```
Fantastic Beasts and Where to Find Them
Doctor Strange
Kong: Skull Island
Arrival
Guardians of the Galaxy
Rings
Split
John Wick
Mad Max: Fury Road
Jurassic World
Logan
Fifty Shades Darker
Beauty and the Beast
Captain America: Civil War
Interstellar
Assassin's Creed
```

- Crear el método `with_genres` que dado un número `n`, retorne todas las películas que tienen `n` o más géneros.

```
In [7]: def with_genres(movies, n):
                """Retorna una lista de peliculas con mas de n generos """
                return filter(lambda x: len(x.genres) >= n, movies)
```

- Crear el método `tops_of_genre` que dado un género, retorne las 10 mejores películas ordenadas.

```
In [8]: def tops_of_genre(movies, genre):
            """ Retorna las 10 mejores peliculas de un género"""
            one_genre = sorted(filter(lambda x: genre in x.genres,
                                      movies),
                               key=lambda x: x.rating)
            return one_genre[:10] if len(one_genre) > 10 else one_genre
```

```
In [9]: for i, top in enumerate(tops_of_genre(MOVIES, "Drama")):
            print(i + 1, top.title)
```

```
1 Run All Night
2 World War Z
3 The Godfather
4 Dogtooth
5 Ex Machina
6 Fifty Shades of Grey
7 Schindler's List
8 The Summer I Turned 15
9 Cinderella
10 Sully
```

- Crear el método `actor_rating` que dado el nombre de un actor, retorna el promedio del rating de las películas en las que ha participado.

```
In [10]: from functools import reduce
```

```
In [11]: def actor_rating(movies, casts, actor):
             """Dado el nombre de un actor, retorna el promedio del rating de las
                 peliculas en las que ha participado """
             actors_movies = [c.movie for c in filter(
                 lambda cast: cast.name == actor, casts)]
             if len(actors_movies) > 0:
                 return reduce(
                     lambda x, y: x + y,
                     map(
                         lambda m: m.rating,
                         filter(
                             lambda movie: movie.title in actors_movies,
                             movies))) / len(actors_movies)
             else:
                 return 0
```

```
In [12]: print("Hugh Jackman's rating: {}".format(actor_rating(MOVIES, CASTS, "Hugh Jackman")))
```

```
Hugh Jackman's rating: 58.03259566666666
```

- Crear el método `compare_actors` que dado el nombre de dos actores, imprima cual de los dos esta mejor valorado según su promedio.

```
In [13]: def compare_actors(movies, casts, actor1, actor2):
             """Compara el rating de dos actores e imprime quién supera a quién"""
             if actor_rating(movies, casts, actor1) > actor_rating(movies, casts, actor2):
                 print('{} supera a {}'.format(actor1, actor2))
             else:
                 print('{} supera a {}'.format(actor2, actor1))
```

- Crear el método `movies_of` que dado el nombre de un actor. retorne una lista con tuplas que contenga los pares (pelicula, personaje) en los que actuo.

```
In [14]: def movies_of(casts, actor):
             """ Retorna una lista de tuplas pelicula, personaje en las que ha
             participado un actor en especifico """
             return [(c.movie, c.character) for c in casts if c.name == actor]
```

```
In [15]: for movie, character in movies_of(CASTS, "Hugh Jackman"):
             print(movie)
```

```
Chappie
X-Men: Days of Future Past
Logan
```

- Crear el método `from_year` que dado un año, retorne todas las peliculas que se estrenaron en ese año.

```
In [16]: def from_year(movies, year):
             return filter(lambda m: m.release.split('-')[0] == str(year), movies)
```

```
In [17]: for movie in from_year(MOVIES, 2017):
             print(movie.title)
```

```
John Wick: Chapter 2
Kong: Skull Island
Pirates of the Caribbean: Dead Men Tell No Tales
Rings
Split
Get Out
xXx: Return of Xander Cage
Life
Logan
Fifty Shades Darker
The Boss Baby
Dead Over Heels: An Aurora Teagarden Mystery
Beauty and the Beast
Ghost in the Shell
```

1.3 Generador con `send`

```
In [18]: def promedio_movil():
             total_acumulado = yield
             cantidad_numeros = 1
             while True:
                 nuevo = yield total_acumulado / cantidad_numeros
                 cantidad_numeros += 1
                 total_acumulado += nuevo

In [19]: generador = promedio_movil()
         next(generador) # ¿Por qué?

In [20]: for i in range(10):
             print(generador.send(i))
```

```
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
```

1.4 Otras funciones

- enumerate
- filter
- iter
- len
- map
- max
- min
- next
- reduce
- repr
- reversed
- sorted
- str
- sum
- zip