Review Session #<3

Happy Valentine’s Day!!!

Objectives

- Function Review
- Recursion Strategies
- List review
- Classes
- Objects

Function Review

Defining Functions

- Syntax:

```
let <name> = (<parameters>): <return type> => {
        //function body
};
```

- Example

```
let valentine = (name : string) : string =>{
        return “Happy Valentine’s Day, “
        +name+”!!!”;
};
```

Calling Functions

- Syntax:

```
<name> (<arguments>);
```

- Example:

```
let card : string = valentine(“Mason”);
print(valentine(“Izzi”));
valentine(“Brooks”);
```

Strategies for Solving Recursion

- Draw it out
- Work backwards
- Analyze the problem
  - What is the base case?
  - What are the other cases?
  - What is really happening each time we call the function?

Analyzing Recursion

```
let bottles = (b: number): string => {
    if (b <= 0) {
        return "No more bottles left on the wall :(";
    } else {
        let oneLess: number = b - 1;
        return b + " bottles of water on the wall. " + b + " bottles of water.  Take one
        down pass it around. " + oneLess + " bottle of water on the wall! "
        +   bottles(b - 1);
    }
};
```

Base Case

Recursive Case

Recursive Call

List Toolkit

Function Name	Use	Example use

`cons(<value>, <list>)`	Combining one value with a list to form a new list	`let groceries: List<string>;`
		`groceries = cons( "zebra cakes", cons("gold fish", cons("juice", null)));`
		`print(groceries);`

`first(<list>)`	Retrieving the first value from a list	`let item1: string= first(groceries);`
		`print(item1);`

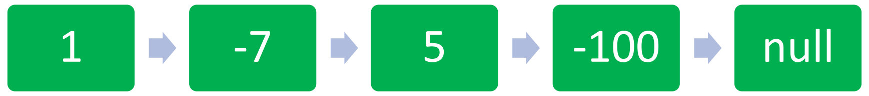
`rest(<list>)`	Getting the list that follows the first item	`let stillNeed : List<string>;`
		`stillNeed= rest(groceries);`
		`print(stillNeed);`

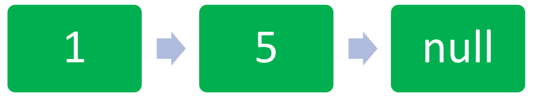
`listify(<comma separated values>)`	Creating lists with several values as arguments	`let groceries: List<string>;`
		`groceries= listify(“zebra cakes”, “gold fish”, “juice”);`
		`Print(groceries);`

List Practice: What does foo do?

```
let numbers: List<number> = listify(1, -7, 5, -100);
let foo = (list: List<number>): List<number> => {
    if (list === null) {
        return null;
    } else {
        let current : number = first(list);

        if (current >= 0) {
            return cons(current, foo(rest(list)));
        } else {
            return foo(rest(list));
        }
    }
};
print(numbers);
print(foo(numbers));
```

Strategy: Draw it out





What is a Class?

- Classes are blue prints for objects
- A class is a set of properties
- In a class definition we have
  - Key word: **class**
  - **Name** for the class
    - Usually starts with a capital letter
  - **Properties**
    - Variables given default values

- Syntax:

```
class Name {
        property1: type1= defaultValue;
        property2: type2= defaultValue;
}
```

- Example:

```
class BankAccount {
        user: string= “username”;
        savings: number= 0;
}
```

What is an Object?

- An object is a specific implementation of a class
- We can have many objects of the same class type
- Objects of the same class have the same properties but can have different values for those properties

Creating New Objects

- To create a new object we use the following syntax:

```
    let name : ClassName = new ClassName();
```

- Example:

```
    let gates : BankAccount = new BankAccount();
```

Accessing Object Properties

- To change or access a property of an object, we use the following syntax:

```
objectName.property = value;
print(objectName.property);
let temp : type= objectName.property;

let gates: BankAccount = new BankAccount();
gates.user= “Bill Gates”;
gates.savings= 867530900000;
let userOfAccount : string = gates.user;
```

Objects Summary

Creating New Objects

- Syntax:

```
let name : ClassName = new ClassName();
```

- Example:

```
let broke : BankAccount = new BankAccount();
```

Accessing Properties

- Syntax

```
objectName.property=value;

print(objectName.property);
```

- Example

```
broke.user=“sganci”;

print(broke.user + “has “+ broke.savings+”
in savings”);
```

Classes vs. Objects

Class

- General blue prints
- Ex:

```
class Food {
         name : string = “food name”;
         cal : number = 0;
         healthy : Boolean = false;
}
```

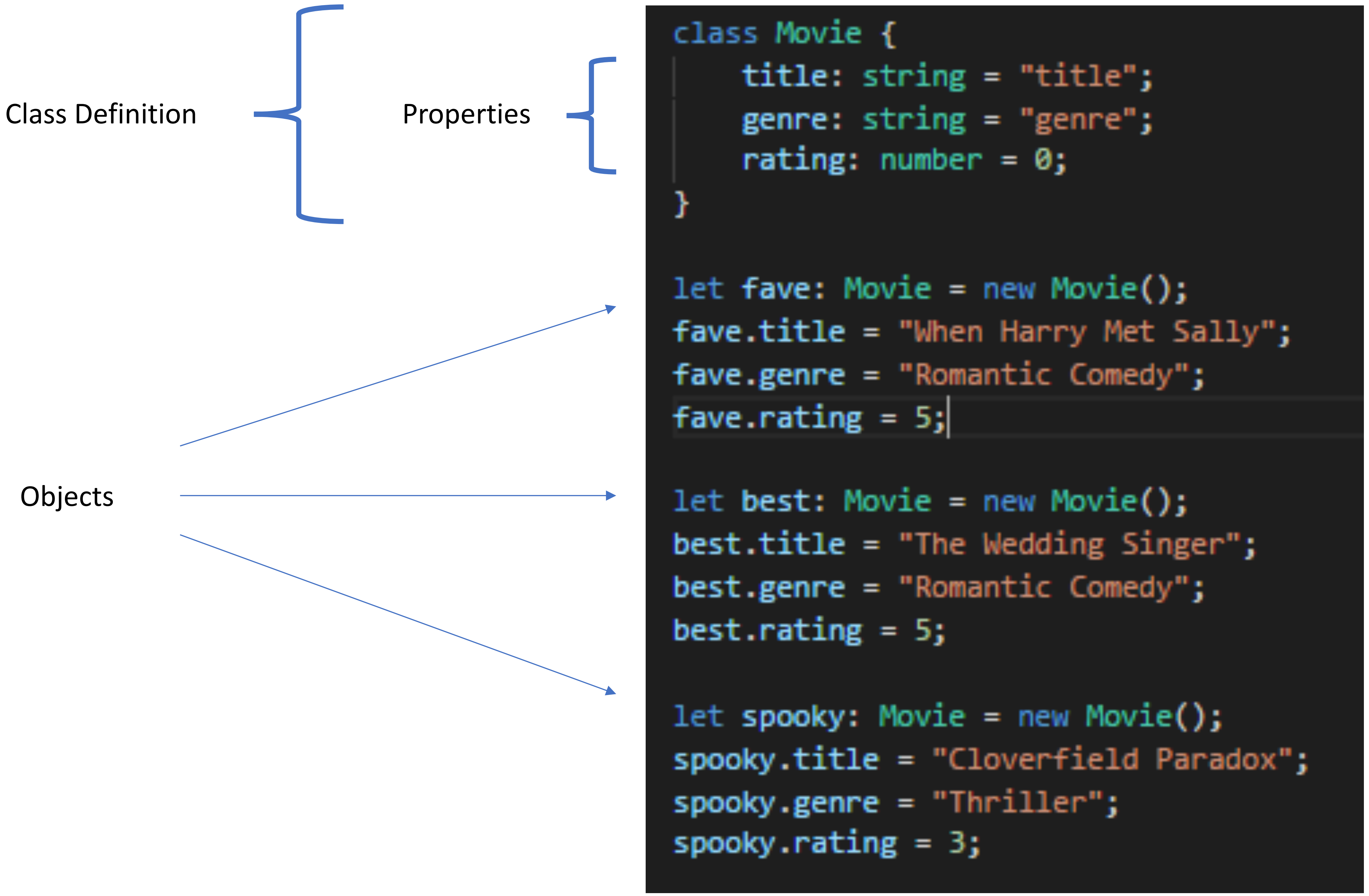
Object

- Specific instances
- Ex:

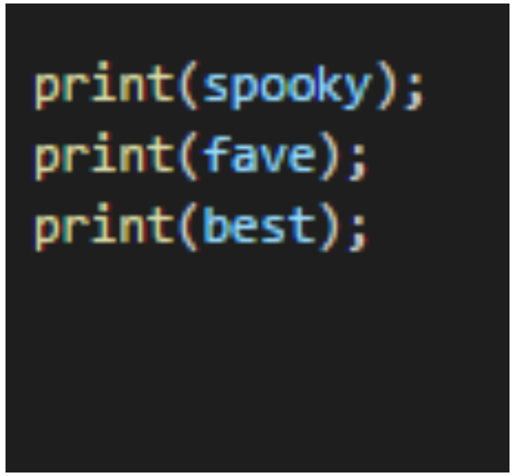
```
let yum: Food = new Food();
yum.name= “pizza”;
yum.cal= 500;
yum.healthy= false;

let meh: Food = new Food();
meh.name= “salad”;
meh.cal=150;
meh.healthy= true;
```

Class Practice

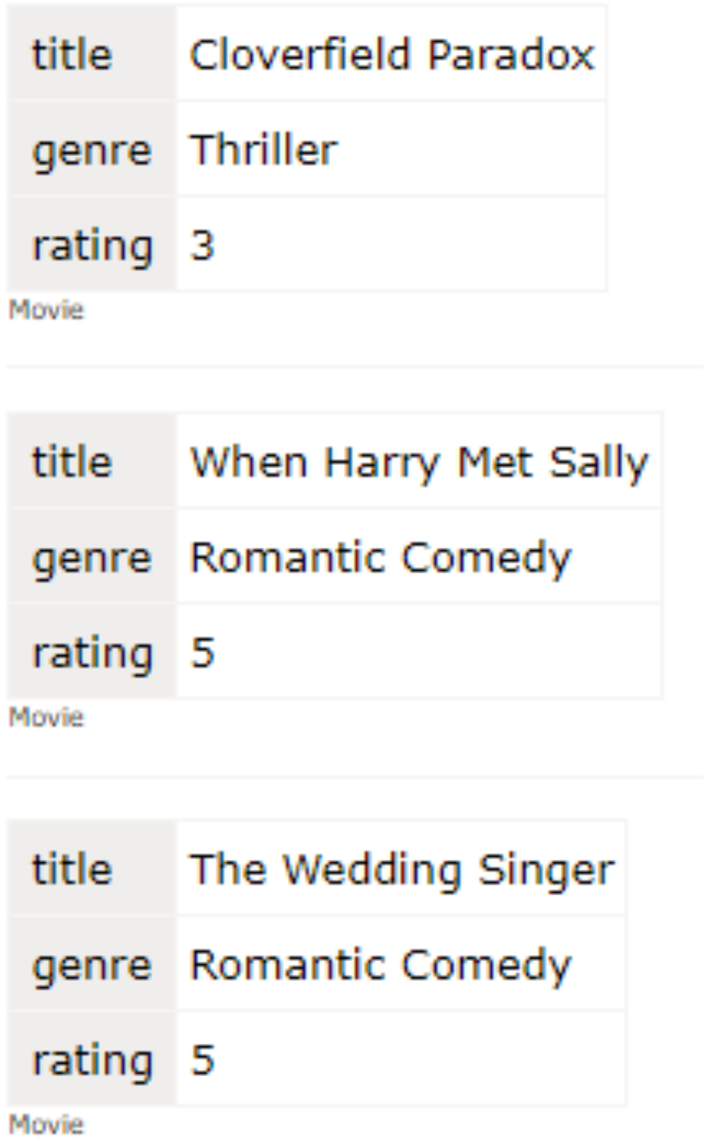
- Create a class called Movie
- The class should have the following properties:
  - A title of type string with a default value of “movie title”
  - A genre of type string with a default value of “movie genre”
  - A rating of type number with a default value of 0
- After you have written the Movie class, try to create some Movie objects using your favorite movies as inspiration ☺



```
class Movie {
    title: string = "title";
    genre: string = "genre";
    rating: number = 0;
}

let fave: Movie = new Movie();
fave.title = "When Harry Met Sally";
fave.genre = "Romantic Comedy";
fave.rating = 5;

let best: Movie = new Movie();
best.title = "The Wedding Singer";
best.genre = "Romantic Comedy";
best.rating = 5;

let spooky: Movie = new Movie();
spooky.title = "Cloverfield Paradox";
spooky.genre = "Thriller";
spooky.rating = 3;
```



```
print(spooky);
print(fave);
print(best);
```



title	Cloverfield Paradox
genre	Thriller
rating	3

Movie

title	When Harry Met Sally
genre	Romantic Comedy
rating	5

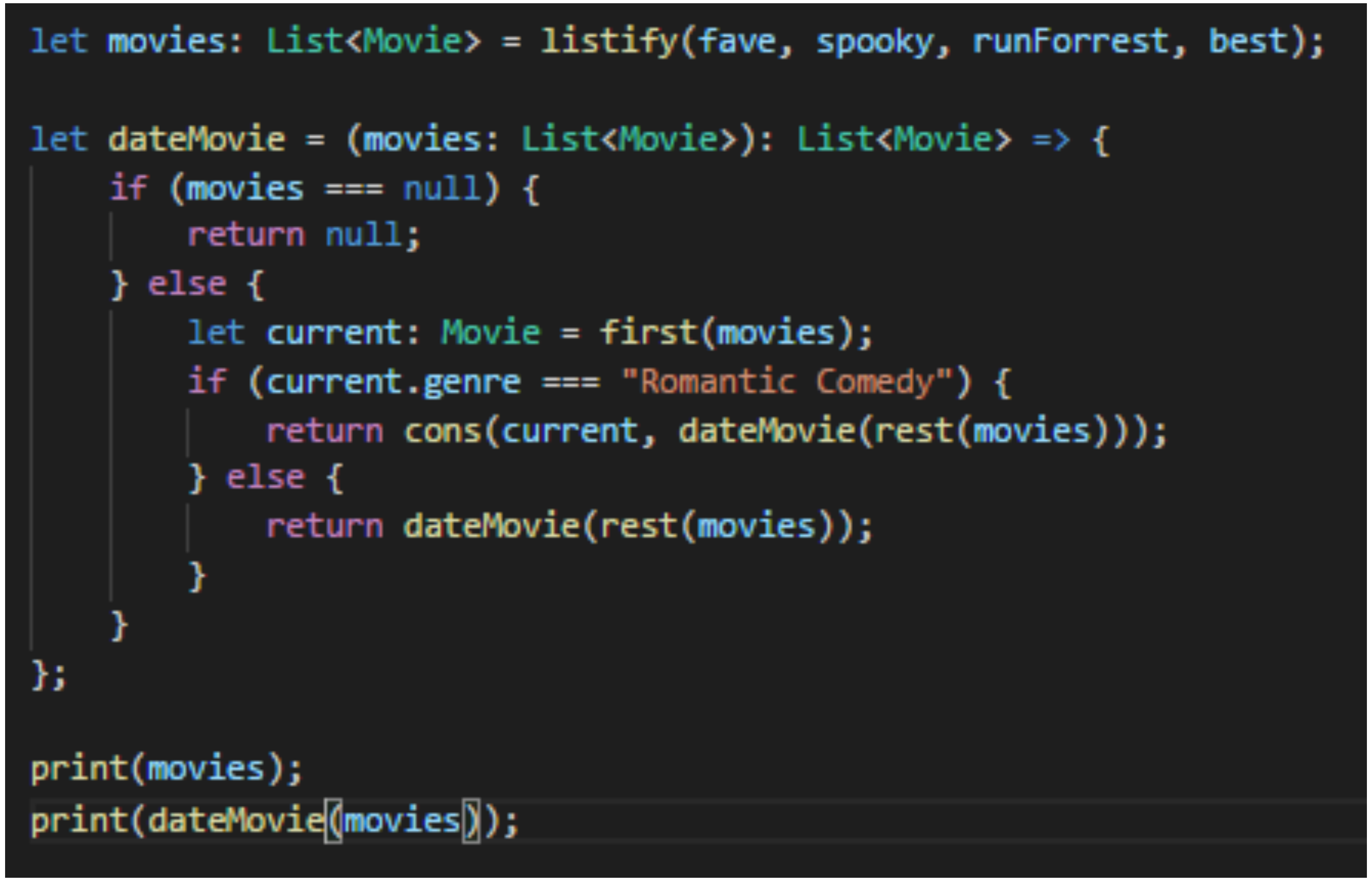
Movie

title	The Wedding Singer
genre	Romantic Comedy
rating	5

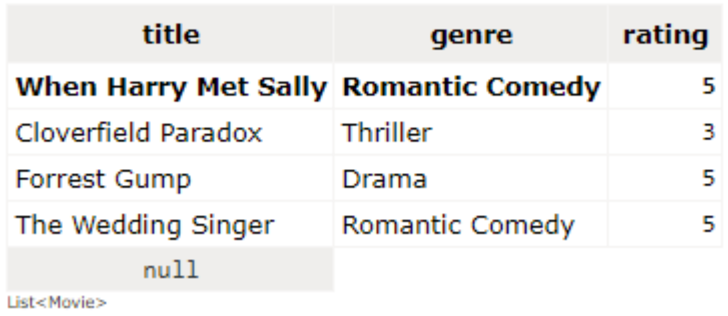
Movie

Date Night Dilemma

- You and your crush are hanging out (score). You have a huge list of movies to choose from. Let’s write a function to help you narrow down your choices
- Function requirements:
  - Name: dateMovie
  - Input: should take in a list of movies
  - Output: a list of movies that have a genre of “Romantic Comedy”

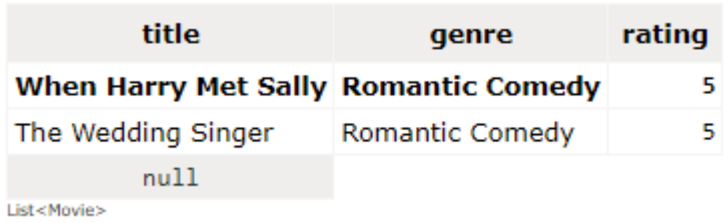


```
let movies: List<Movie> = listify(fave, spooky, runForrest, best);

let dateMovie = (movies: List<Movie>): List<Movie> => {
    if (movies === null) {
        return null;
    } else {
        let current: Movie = first(movies);
        if (current.genre === "Romantic Comedy") {
            return cons(current, dateMovie(rest(movies)));
        } else {
            return dateMovie(rest(movies));
        }
    }
};

print(movies);
print(dateMovie(movies));
```



title	genre	rating
When Harry Met Sally	**Romantic Comedy**	5
Cloverfield Paradox	Thriller	3
Forrest Gump	Drama	5
The Wedding Singer	Romantic Comedy	5
null		

List<Movie>



title	genre	rating
When Harry Met Sally	**Romantic Comedy**	5
The Wedding Singer	Romantic Comedy	5
null		

List<Movie>