Task 1

The loyalty program discount calculation is based on the function:

```
function calculateDiscount(years, paymentMethod, returnedOrder) {
 let discount = 0;
 if (years >= 1)
 {
  if (returnedOrder) {
   discount = 0;
   return discount;
  }
  if (paymentMethod === 'MasterCard') {
   discount = 0.1;
  }
  if (getCustomerBirthdayDate() === getCurrentDate()) {
   discount = 0.5;
  }
 }
 return discount;
}
```

//years — number of years since the user made their first purchase.
//paymentMethod — stores the value with payment type.
//returnedOrder -stores information on whether the product was previously returned by the same buyer.
//getCustomerBirthdayDate — internal function. It returns customer's Birthday Date.
//getCurrentDate — internal function. It returns the current date.

List of requirements from the customer:

- Discounts are available only for buyers who made their first purchase at least 1 year ago.
- All members of the loyalty program have a 50 % discount on their birthday.
- If the product is re-ordered by the same buyer after the return, no discount will be applied.
- All members of the loyalty program who pay with a MasterCard receive a 10% discount.
- Discounts are not cumulative.

Task1. Solution

Define main equivalence classes:

1. years < 1 - buyers who made their first purchase less than 1 year ago
2. years >= 1 - buyers who made their first purchase at least 1 year ago

where

years = current_date - customer_first_purchase_date

Then add 3 more conditions that we have according to the requirements and make a **decision table**: (true = T, false = F)

Condition		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	>=1 year	T	T	T	T	T	T	T	T	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
2	not_returned	T	T	T	T	**F**	**F**	**F**	**F**	T	T	T	T	F	F	F	F
3	MasterCard	T	T	F	**F**	T	T	F	F	T	T	F	F	T	T	F	F
4	birthday	T	F	T	**F**	T	F	T	F	T	F	T	F	T	F	T	F
discount %		**50**	**10**	**50**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

where:

1. First purchase at least 1 year ago - “>=1 year”
2. The product isn’t re-ordered by the same buyer after the return - “not_returned”
3. The payment method is MasterCard - “MasterCard”
4. The purchase on a customer's birthday (birthday date = current date) - “birthday”

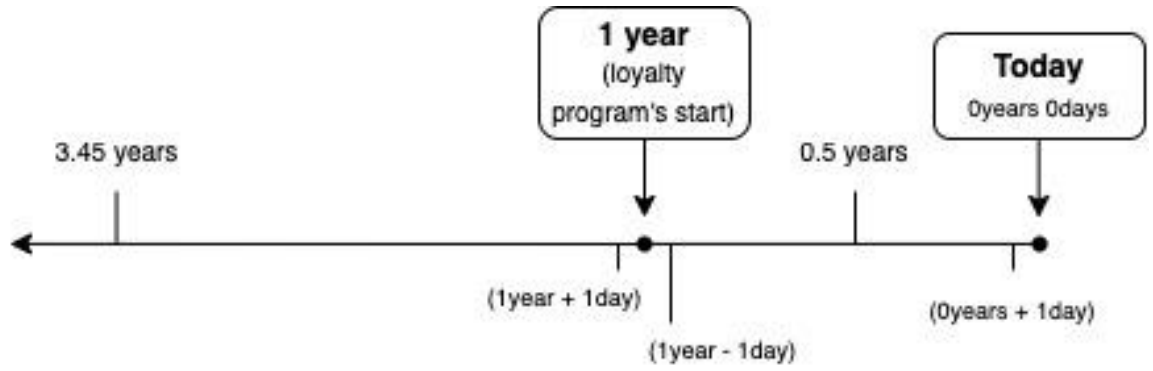
According to the requirements, find out the system’s behavior for each case - the presence of a discount and its size or the absence of a discount.

Now we can identify 5 classes with equivalent behavior (expected results of our tests). Let’s define them:

# #	**Equivalence class**	**Comment**	**Expected result, discount**
1	<1years	First purchase less than 1 year ago. Other conditions don’t matter, the loyalty program only for members	0%
2	>=1 year, but the product was returned	No discount when the product is re-ordered by the same buyer after the return	0%
3	>=1 year, the product was not returned, pay with not a MasterCard, no birthday	No discounts available	0%
4	>=1 year, the product was not returned, MasterCard, not birthday	10% for members who pay with a MasterCard	10%
5	>=1 year, the product was not returned, birthday	50% discount on a customer's birthday (priority but not cumulative in cases “Birthday + Mastercard)	50%

Two of those conditions have only two possible states each - true or false (“not_returned” and “MasterCard” conditions)

But for quality testing coverage we should apply the **boundary value technique** for the other two conditions (“>=1 year” and “birthday” conditions). When combined with the **equivalence partitioning technique**, we get points for testing.



For the ”>=1 year” condition (I decided to use 1 day as a minimum step but it may be 1 hour or even 1 sec if the time of the first purchase is also important):

- 0years 0days boundary value
- (0years + 1day) upper class bound
- 0.5years within the class
- (1year - 1day) lower class bound
- 1year boundary value
- (1year + 1day) upper class bound
- 3.45years within the class

I consider that we don’t need all those points for the “<1years”(not members) group, so I suggest discarding one of them (0.5years) for optimization (it has the same sense with (0years + 1day) point because both are within the same class)

For the “birthday” condition only the boundary value point matters (the exact birthday of the client) not the period of time. Therefore, it will be enough to check the boundary value and the points before and after the bound:

- (BD - 1day) lower class bound
- Birthday boundary value
- (BD + 1day) upper class bound

So, we have such parameters:

Year: 0year, (0years+1d), (1years-1d), 1year, (1year+1d), 3.45years
Return: returned, not_returned

Payment: MasterCard, Otherbirthday: BD-1d, BD, BD+1d

and need 6*2*2*3= 72 combinations (tests) to check all of them

Let's use the **pairwise technique** to optimize amount of tests and then check the coverage quality.

List of tests after applying the PICT tool:

class	year	returned	mastercard	birthday
1	0years	returned	MasterCard	BD+1d
1	0years	not_returned	Other	BD-1d
1	0years	returned	Other	BD
1	(0years+1d)	not_returned	Other	BD+1d
1	(0years+1d)	returned	MasterCard	BD-1d
1	(0years+1d)	not_returned	MasterCard	BD
1	(1year-1d)	returned	MasterCard	BD-1d
1	(1year-1d)	not_returned	Other	BD+1d
1	(1year-1d)	returned	Other	BD
2	1year	returned	Other	BD+1d
4	1year	not_returned	MasterCard	BD-1d
2	1year	returned	Other	BD
2	(1year+1d)	returned	MasterCard	BD+1d
5	(1year+1d)	not_returned	Other	BD
2	(1year+1d)	returned	Other	BD-1d
4	3.45years	not_returned	MasterCard	BD+1d
2	3.45years	returned	Other	BD
2	3.45years	returned	Other	BD-1d

and **traceability matrix** to check the coverage of each of our classes:

	Equivalence class	Discount	tests
1	<1years	0	9 (-5)
2	>=1 year, but the product was returned	0	6 (-3)
3	>=1 year, the product was not returned, pay with not a MasterCard, not birthday	0	0 (+2)
4	>=1 year, the product was not returned, MasterCard, not birthday	10	2
5	>=1 year, the product was not returned, birthday	50	1 (+2)

We see an excess number of tests for classes 1 and 2, where the system behaves in the same way and no discounts are awarded. Using the pairwise technique agan, we can reduce the number of tests for 1 and 2 classes (those tests allow testing all possible bets of parameters).

And we have a problem in the classes 3 and 5 (no coverage and ansufficient coverage). Here I added tests manually to check important combination (especially for class 5 with the most complex logic)

class	year	returned	mastercard	birthday
5	1year	not_returned	MasterCard	BD
5	3.723year	not_returned	MasterCard	BD
3	1year	not_returned	Other	BD+1d
3	(1year+1d)	not_returned	Other	BD-1d

As a result, we have 4+3+2+2+3 = 14 tests for this task (TestCases).

Using test design techniques, we significantly reduce their number but at the same time provide full coverage of all important parameters and potential problem areas:

class	year	returned	mastercard	birthday	ex.res
1	0years	returned	MasterCard	BD+1d	0%
1	(0years+1d)	not_returned	Other	BD	0%
1	(0years+1d)	not_returned	MasterCard	BD	0%
1	(1year-1d)	returned	MasterCard	BD-1d	0%
2	1year	returned	Other	BD	0%
2	(1year+1d)	returned	MasterCard	BD+1d	0%
2	3.45years	returned	Other	BD-1d	0%
3	1year	not_returned	Other	BD+1d	0%
3	(1year+1d)	not_returned	Other	BD-1d	0%
4	1year	not_returned	MasterCard	BD-1d	10%
4	3.45years	not_returned	MasterCard	BD+1d	10%
5	(1year+1d)	not_returned	Other	BD	50%
5	1year	not_returned	MasterCard	BD	50%
5	3.723year	not_returned	MasterCard	BD	50%