

Table of Contents

1. Validating arguments.....	1
1.1. Validation rules.....	2
1.2. Another method for dealing with nulls	3

The previous examples of functions assumed that the input arguments were reasonable values. You can do some simple validation of the parameters to handle the situation where a user supplied an argument that does not work for your function.

1. Validating arguments

What can we do in the function to check some of our arguments? Let's go back to the function `new_salary_1`

Demo 01: `A_emp.newsalary_1` Everyone is going to get a \$300 raise

```
Drop function if exists a_emp.newsalary_1#

create function a_emp.newsalary_1 (
    in_salary    decimal (9,2))
    returns decimal (10,2)
begin
    declare c_increase_amount decimal (9,2) default 300;
    declare v_new_salary      decimal (10,2);

    set v_new_salary := in_salary + c_increase_amount;
    return v_new_salary ;
end;
#
```

This gives us a null return for a null argument and accepts a negative value for salary (scary thought!).

```
Select a_emp.newsalary_1 (4500) as Result #
+-----+
| Result |
+-----+
| 4800.00 |
+-----+

Select a_emp.newsalary_1 (null) as Result #
+-----+
| Result |
+-----+
|  NULL  |
+-----+

Select a_emp.newsalary_1 (-4500) as Result #
+-----+
| Result |
+-----+
| -4200.00 |
+-----+
```

1.1. Validation rules

Suppose we were told to follow the following rules for validation.

If the input is null, then the return should be the minimum increase (in this example \$300.00)

If the input is negative, then the return should be 0

We can handle this logic with If tests

Demo 02:

```
Drop function if exists a_emp.newsalary_1_v2#

create function a_emp.newsalary_1_v2 (
    in_salary decimal (9,2))
    returns decimal (10,2)
begin
    declare c_increase_amount decimal (10,2) default 300;
    declare v_new_salary decimal (10,2);

    if in_salary is null then
        set v_new_salary := c_increase_amount;
    elseif in_salary < 0 then
        set v_new_salary := 0;
    else
        set v_new_salary := in_salary + c_increase_amount;
    end if;

    return v_new_salary ;
end;
#
```

Demo 03: Testing

```
Select
    Salary
, AnticipatedValue
, a_emp.newsalary_1_v2 (salary) as "new salary"
, AnticipatedValue - A_emp.newsalary_1_v2(salary) as "problem"
From (
    Select 10000 as Salary
        , 10300 as AnticipatedValue
    union all
    Select null, 300
    union all
    Select -500, 0
) as tstTbl#
```

Salary	AnticipatedValue	new salary	problem
10000	10300	10300.00	0.00
NULL	300	300.00	0.00
-500	0	0.00	0.00

```
Select
    a_emp.newsalary_1_v2 (4500) as Result_1
, a_emp.newsalary_1_v2 (null) as Result_2
, a_emp.newsalary_1_v2 (-35) as Result3#
```

```

+-----+-----+-----+
| Result_1 | Result_2 | Result |
+-----+-----+-----+
| 4800.00 | 300.00 | 0.00 |
+-----+-----+-----+

```

1.2. Another method for dealing with nulls

We want a function named **FutureDate** that has four parameters. The first is a datetime value, the 2nd, 3rd, and 4th are integer arguments that are expected to be the number of years, months, and days. The function will calculate and return a date that adds the indicated number of years, months and days. The function will use only the year-month-day components of the first parameter- ignoring any time components and returns a date type. If the first argument is null, then return a null. The other arguments can be positive or negative numbers; if they are null then do not add anything for that component. For example:

FutureDate(mydate, 2, 3, 4) will add 2 years, 3 months and 4 days to mydate

FutureDate(mydate, 2, null, 44) will add 2 years, no months and 44 days to mydate

Hopefully you are thinking of the Data_dd function which can add years, month and days to a date value.

We have two other considerations. (1) We need to ignore the time components. The easiest way to do that is assign the datetime parameters to a local date variable. (2) we need to handle any possible null parameter for the year, month, day. If the parameter for year is null, then we want to add 0 years. You might be thinking of If tests, but there is an easier way- use the coalesce function. `set p_yr :=coalesce(p_yr, 0);`

But what if the first parameter for the date is null; the Date_Add function will handle that and return a null if the date value passed to it is null; your code does not need to handle that.

Demo 04: FutureDate version 1

```

drop function if exists a_testbed.FutureDate#

create function a_testbed.FutureDate (
  p_date datetime
, p_yr int
, p_mn int
, p_day int)
  returns date
BEGIN
  declare return_date date ;
  set return_date := p_date;

  set p_yr :=coalesce(p_yr, 0);
  set p_mn :=coalesce(p_mn, 0);
  set p_day :=coalesce(p_day, 0);

  set return_date := date_add(return_date, interval p_yr year);
  set return_date := date_add(return_date, interval p_mn month);
  set return_date := date_add(return_date, interval p_day day);

  return return_date;
end;
#

```

Demo 05: If you are very comfortable with functions you can do this

```

drop function if exists a_testbed.FutureDate2#

```

```
create function a_testbed.FutureDate2 (  
  p_date datetime  
  , p_y int  
  , p_m int  
  , p_d int)  
  returns date  
BEGIN  
  return p_date +  
    interval coalesce(p_y, 0) year +  
    interval coalesce(p_m, 0) month +  
    interval coalesce(p_d, 0) day;  
END;  
#
```