--a  
create or replace function *inc*(San integer) returns integer  
 language plpgsql  
as  
$$  
BEGIN RETURN San + 1; END;  
$$;  
select *inc*(40);  
--b  
create or replace function *sum*(val1 numeric,val2 numeric)  
 returns numeric as $$ begin  
 return val1 + val2;  
 end; $$  
 language plpgsql;  
select sum(50,50);  
--c  
  
create or replace function *check\_divis*(num numeric) returns BOOLEAN  
 language plpgsql  
as  
$$  
BEGIN  
 IF ( num%2=0)  
  
 then return true;  
 else  
 return false;  
 END IF;  
  
END;  
$$;  
select *check\_divis*('1');  
  
--d  
create function *check\_pass*(pass text) returns BOOLEAN  
 language plpgsql  
as  
$$  
BEGIN  
 IF (length(pass) > 10)  
  
 then return true;  
 else  
 return false;  
 END IF;  
  
END;  
$$;  
select \* from *check\_pass*('1');  
--e  
create or replace function *splitting*(text varchar(30)) returns record  
 language plpgsql  
as  
$$  
declare text1 record;  
begin  
 select split\_part(text, ',', 1) ,  
 split\_part(text, ',', 2)  
 into text1;  
 return text1;  
end;  
$$;  
select *splitting*('Hello,world');  
--task 2  
--a  
create table igra(  
 id serial primary key,  
 aty varchar(50) not null,  
 changed timestamp  
);  
  
insert into igra(aty) values('Spiderman');  
insert into igra(aty) values('Ironman');  
insert into igra(aty) values('Superman');  
  
create or replace function *changing*() returns trigger as $$  
begin  
 new.changed = now();  
 return new;  
end;  
$$ language plpgsql;  
  
create trigger igra\_changed before insert or update on igra  
 for each row execute procedure *changing*();  
  
select \* from igra where id=1;  
  
insert into igra(aty) values ('Tor');  
  
update igra  
set aty='Jumandgy'  
where id=1;  
--b  
create table person(  
 id serial primary key,  
 name varchar(20),  
 age integer,  
 year\_of\_birth integer not null  
);  
create or replace function *age\_calculate*()  
 returns trigger  
 language plpgsql  
 as  
$$  
 begin  
 new.age = extract(year from current\_date) - new.year\_of\_birth;  
 return new;  
 end;  
$$;  
create trigger age1 before insert or update on person  
 for each row execute procedure *age\_calculate*();  
  
  
insert into person(name, year\_of\_birth) values ('John', 2003);  
insert into person(name, year\_of\_birth) values ('Jek', 1993);  
insert into person(name, year\_of\_birth) values ('Grigory', 1964);  
insert into person(name, year\_of\_birth) values ('Saken', 2005);  
  
select \* from person;  
-- c  
CREATE table foods(  
 id integer primary key,  
 name varchar(80),  
 price integer  
);  
  
create or replace FUNCTION *total*()  
returns trigger  
 language plpgsql  
 as  
  
 $$  
 BEGIN  
 update foods  
 set price=price+0.12\*price  
 where id = new.id;  
 return new;  
 end;  
 $$;  
  
  
create trigger cost after insert on foods  
 for each row execute procedure *total*();  
insert into foods(id, name,price) values (1, 'Airan', 300);  
insert into foods(id,name,price) values (3, 'Chips',419);  
  
select \* from foods;  
-- d  
create or replace function *reset*() returns trigger language plpgsql  
 as $$  
 begin  
 insert into foods(id,name,price) values(old.id,old.name,old.price);  
 return old;  
 end;  
 $$;  
  
create trigger back  
 after delete  
 on foods  
 for each row  
 execute procedure *reset*();  
delete from foods where id=2;  
select \* from foods;  
  
  
  
  
-- e  
--1e  
create trigger func  
 after insert  
 on igra  
 for each row  
 execute function *age\_calculate*();  
  
create trigger paswalid  
 after insert  
 on foods  
 execute function *total*();  
  
  
  
  
  
  
  
--task 3  
-- The function must return a value but in Stored Procedure it is optional. Even a procedure can return zero or n values.  
--  
-- Functions can have only input parameters for it whereas Procedures can have input or output parameters.  
--  
-- Functions can be called from Procedure whereas Procedures cannot be called from a Function.  
  
--task 4  
--a  
Create table employee(  
 id int primary key,  
 name varchar(120),  
 date\_of\_birth date,  
 age int,  
 salary int,  
 workexperience int,  
 discount int  
);  
  
drop table employee;  
insert into employee(id, name, date\_of\_birth, age, salary, workexperience, discount) values (1,'Muha','2003.01.22',18,1000000,3,4000);  
insert into employee(id, name, date\_of\_birth, age, salary, workexperience, discount) values (2,'Jako','2002.02.23',19,200000,3,6000);  
  
create or replace procedure *salary*()  
 as $$  
 begin  
 update employee set salary = salary\*1.1 where workexperience>=2;  
 update employee set discount=discount\*1.1 where workexperience>=2;  
 update employee set discount = discount\*1.01 where workexperience>=5;  
 end; $$  
 language plpgsql;  
call *salary*();  
select \* from employee;  
-- b  
create or replace procedure *sal*()  
 as $$  
 begin  
 update employee set salary = salary\*1.15 where age>=40;  
 update employee set salary = salary\*1.15 where workexperience>=8;  
 update employee set discount = discount\*1.20 where workexperience>=8;  
 end; $$  
 language plpgsql;  
call *sal*();  
select \* from employee;  
  
-- task 5  
create table members(  
 memid integer,  
 surname varchar(200),  
 firstname varchar(200),  
 address varchar(300),  
 zipcode integer,  
 telephone varchar(20),  
 recommendedby integer,  
 joindate timestamp  
);  
create table bookings(  
 facid integer,  
 memid integer,  
 starttime timestamp,  
 slots integer  
);  
create table facilities(  
 facid integer,  
 name varchar(200),  
 membercost numeric,  
 guestcost numeric,  
 initialoutlay numeric,  
 monthlymaintenance numeric  
);  
with recursive recom(recommender, member) as (  
 select recommendedby, memid  
 from members  
 union all  
 select members.recommendedby, recom.member  
 from recom  
 inner join members  
 on members.memid = recom.recommender  
)  
select recom.member member, recom.recommender, members.firstname, members.surname  
 from recom  
 inner join members  
 on recom.recommender = members.memid  
 where recom.member = 22 or recom.member = 12  
order by recom.member asc, recom.recommender desc