-- 1a  
create function *increment*(val int)  
returns INT as $$  
 Begin  
 return val+1;  
 end;$$  
LANGUAGE plpgsql;  
  
drop function *increment*(val int);  
  
select *increment*(7);  
select *increment*(*increment*(7));  
  
-- 1b  
create function *cube*(inout a int)  
as $$  
 Begin  
 a:=a\*a;  
 end; $$  
language plpgsql;  
  
select *cube*(2);  
  
-- 1c  
create function *sum*(val1 int, val2 int, out sum int)  
as $$  
 Begin  
 sum:= val1+val2;  
 end; $$LANGUAGE plpgsql;  
  
select *sum*(2, 5);  
  
-- 1.d  
create function *is\_even*(val int)  
returns bool as $$  
Begin  
 if val%2=0 then  
 return true;  
 else  
 return false;  
 end if;  
end;$$  
language plpgsql;  
  
drop function *is\_even*;  
  
select *is\_even*(8);  
  
-- 1.e  
create function *sum\_avg*(VARIADIC list int[], out total int, out average numeric)  
as $$  
Begin  
Select into total *sum*(list[i])  
from *generate\_subscripts*(list, 1) g(i);  
select into average *avg*(list[i])  
 from *generate\_subscripts*(list, 1) g(i);  
end; $$  
language plpgsql;  
  
select\* from *sum\_avg*(10, 20, 30);  
  
-- 1f  
create function *count\_fun*(variadic list int[], out cnt int)  
as $$  
 begin  
 Select into cnt *count*(list[i])  
 from *generate\_subscripts*(list, 1) g(i);  
 end;$$  
language plpgsql;  
  
select\* from *count\_fun*(1, 2, 3, 4);  
  
-- 1g  
create function *check\_pass*(password varchar)  
returns bool as $$  
 begin  
 if password similar to '\w+\d\*' then  
 return true;  
 else  
 return false;  
 end if;  
end;$$  
language plpgsql;  
  
select *check\_pass*('nurtas');  
  
-- 1h  
create type pair AS(pass varchar, stat varchar);  
create function *two\_one*(password varchar)  
returns pair as $$  
 declare  
 res pair;  
 stat varchar;  
 begin  
 if *check\_pass*(password) then stat='valid';  
 else stat = 'invalid';  
 end if;  
 select password, stat into res;  
 return res;  
 end;$$  
language plpgsql;  
  
-- task2  
CREATE TABLE person(  
 id INT PRIMARY KEY,  
 name VARCHAR,  
 last\_name VARCHAR,  
 date\_of\_birth DATE,  
 age INT,  
 insertion\_time timestamp  
);  
CREATE TABLE item(  
 item\_id INT PRIMARY KEY,  
 item\_name VARCHAR,  
 price\_initial NUMERIC,  
 price\_final NUMERIC  
);  
  
CREATE TABLE accounts(  
 acc\_id INT PRIMARY KEY,  
 acc\_name VARCHAR,  
 password VARCHAR  
);  
  
-- 2a  
CREATE FUNCTION *trig\_func*()  
RETURNS TRIGGER AS $$  
 BEGIN  
 new.insertion\_time = *now*();  
 RETURN new;  
 END;$$  
LANGUAGE PLPGSQL;  
  
CREATE TRIGGER time\_of\_action BEFORE INSERT OR UPDATE  
 ON person  
 FOR EACH ROW  
 EXECUTE PROCEDURE *trig\_func*();  
  
DROP FUNCTION *trig\_func*;  
DROP TRIGGER time\_of\_action ON person;  
  
-- 2b  
create function *calc\_age*() returns trigger as $$  
 declare year\_c INT;  
 begin  
 select *date\_part*('year', *age*(new.date\_of\_birth)) into year\_c;  
 new.age=year\_c;  
 return new;  
 end;$$  
language plpgsql;  
  
create trigger calc\_age  
 before insert or update  
 on person  
 for each row  
 execute procedure *calc\_age*();  
  
DROP TRIGGER calc\_age ON person;  
DROP FUNCTION *calc\_age*;  
  
INSERT INTO person VALUES(1,'Nurtas','Serikkanov','2003-10-17');  
DELETE FROM person WHERE id = 1;  
  
-- 2c  
create function *add\_tack*()  
 returns trigger as $$  
 begin  
 new.price\_final = new.price\_initial\*1.12;  
 return new;  
 end;$$  
language plpgsql;  
  
create trigger add\_tack  
 before insert or update  
 on item  
 for each row  
 execute procedure *add\_tack*();  
  
  
INSERT INTO item VALUES(3,'something3',1000);  
INSERT INTO item VALUES(4,'smth',1000);  
  
drop trigger add\_tack on item;  
drop function *add\_tack*();  
  
-- 2d  
CREATE FUNCTION *prev\_del*()  
RETURNS TRIGGER AS $$  
 BEGIN  
 INSERT INTO item VALUES(old.item\_id,old.item\_name,old.price\_initial);  
 RETURN old;  
 END;$$  
LANGUAGE PLPGSQL;  
  
CREATE TRIGGER prevent\_deletion AFTER DELETE  
 ON item  
 FOR EACH ROW  
 EXECUTE PROCEDURE *prev\_del*();  
  
DROP TRIGGER prevent\_deletion ON item;  
DROP FUNCTION *prev\_del*;  
  
DELETE FROM item WHERE item\_id = 3;  
  
-- 2e  
CREATE FUNCTION *launch*()  
RETURNS TRIGGER AS $$  
 DECLARE pass\_val pair;  
 BEGIN  
 pass\_val = validity(new.password);  
 IF pass\_val.stat = 'valid' THEN RETURN new;  
 ELSE RAISE 'INVALID PASSWORD';  
 END IF;  
 END;$$  
LANGUAGE PLPGSQL;  
  
CREATE TRIGGER launch\_trig BEFORE INSERT  
 ON accounts  
 FOR EACH ROW  
 EXECUTE PROCEDURE *launch*()  
  
-- task3  
-- a)Increases salary by 10% for every 2 years of work  
-- experience and provides 10% discountand after5 yearsadds 1% to the discoun  
create or replace procedure *calc*()  
 language plpgsql  
 as $$  
 declare  
 row record;  
 new\_salary int;  
 new\_disc int;  
 begin  
 for row in select \* from worker  
 loop  
 if(row.duration%2 = 0) then  
 new\_salary = row.salary + (row.salary\*10/100);  
 new\_disc = row.discount + (row.discount\*10/100);  
 else  
 new\_salary = row.salary;  
 new\_disc = row.discount;  
 end if;  
  
 if(row.duration%5 = 0) then  
 new\_disc = row.discount + (row.discount\*1/100);  
 new\_salary = row.salary;  
 else  
 new\_salary = row.salary;  
 new\_disc = row.discount;  
 end if;  
 update worker set salary = new\_salary where id = row.id;  
 update worker set discount = new\_disc where id = row.id;  
 end loop;  
 end;  
 $$  
  
-- 2b  
create or replace procedure *calc2*()  
 language plpgsql  
 as $$  
 declare  
 row record;  
 new\_salary int;  
 new\_disc int;  
 begin  
 for row in select \* from worker  
 loop  
 if(*extract*(year from *age*(row.date\_of\_birth)) >= 40) then  
 new\_salary = new\_salary + (new\_salary\*15/100);  
 end if;  
 if(row.duration > 8) then  
 new\_salary = new\_salary + (new\_salary\*15/100);  
 new\_disc = 20;  
 end if;  
 update worker set salary = new\_salary where id = row.id;  
 update worker set discount = new\_disc where id = row.id;  
 end loop;  
 end;  
 $$