```sql
-- 1.
-- a
CREATE FUNCTION increment(val integer) RETURNS integer AS $$
BEGIN
RETURN val + 1;
END; $$
LANGUAGE plpgsql;

SELECT increment(5);


-- b
CREATE FUNCTION sum(x numeric, y numeric) RETURNS numeric AS $$
BEGIN
RETURN x + y;
END; $$
LANGUAGE plpgsql;

SELECT sum(57, 13);


-- c
CREATE FUNCTION div_two(val numeric) RETURNS boolean AS $$
BEGIN
    IF(val % 2 = 0) THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
END IF;
END; $$
LANGUAGE plpgsql;

SELECT div_two(26);

-- d
CREATE FUNCTION check_passwor(passw varchar(255)) RETURNS BOOLEAN AS $$
BEGIN
    IF (passw LIKE '%[a-z]%[a-z]%' AND
        passw LIKE '%[A-Z]%[A-Z]%' AND
        passw LIKE '%[0-9]%[0-9]%' AND
        passw LIKE '%[~!@#$%^&]%[~!@#$%^&]%' AND
        length(passw) >= 8) THEN RETURN TRUE;
    ELSE
        RETURN FALSE;
END IF;
END; $$
LANGUAGE plpgsql;

SELECT check_passwor('HOLIDAY8');


-- e
CREATE FUNCTION square_cub(in val integer, out square integer, out cub
integer) AS $$
BEGIN
    square := val * val;
    cub := val * val * val;
END; $$
LANGUAGE PLPGSQL;

SELECT square_cub(2);
```

```sql
-- 2.
-- a
CREATE FUNCTION trigger_function() RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TABLE table1 (
  id SERIAL NOT NULL PRIMARY KEY,
  content TEXT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  completed_at TIMESTAMPTZ
);

CREATE TRIGGER set_timestamp
BEFORE UPDATE ON table1
FOR EACH ROW
EXECUTE PROCEDURE trigger_function();

INSERT INTO table1 (content)
VALUES ('Hey') RETURNING *;

UPDATE table1
SET completed_at = now()
WHERE content = 'Hey' RETURNING *;



-- b
CREATE TABLE patient (
   id INT GENERATED ALWAYS AS IDENTITY,
   date_of_birth date,
   age integer,
   PRIMARY KEY(id)
);

CREATE OR REPLACE FUNCTION ages() RETURNS TRIGGER AS $$
BEGIN
    NEW.age = extract(years from age(current_date, new.date_of_birth ));
RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER compute_age
Before INSERT
ON patient
FOR EACH ROW
EXECUTE PROCEDURE ages();

INSERT INTO patient (date_of_birth)
VALUES ('2000-10-05');

-- C
CREATE TABLE table3 (
  price INT NOT NULL
);

CREATE FUNCTION tax_function() RETURNS TRIGGER AS $$
BEGIN
    new.price = new.price * 0.12;
    RETURN new;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER set_tax
BEFORE INSERT ON table3
FOR EACH ROW
EXECUTE PROCEDURE tax_function();
INSERT INTO table3(price)
VALUES (50);

DROP TABLE table3;
DROP FUNCTION tax_function();
DROP TRIGGER set_tax ON table3;


-- d
CREATE TABLE table4 (
   id INT GENERATED ALWAYS AS IDENTITY,
   name VARCHAR NOT NULL,
   date_of_birth date,
   age integer,
   PRIMARY KEY(id)
);

CREATE FUNCTION del_function() RETURNS TRIGGER AS $$
BEGIN
    RETURN null;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER del_trigger
BEFORE INSERT ON table4
FOR EACH ROW
EXECUTE PROCEDURE del_function();
DELETE FROM table4 WHERE name= 'Khalida';



-- e
CREATE TABLE t_passw (
   id INT GENERATED ALWAYS AS IDENTITY,
   name VARCHAR NOT NULL,
   password VARCHAR,
   PRIMARY KEY(id)
);

CREATE FUNCTION check_passw() RETURNS TRIGGER AS $$
BEGIN
    IF (new.password LIKE '%[a-z]%[a-z]%' AND
        new.password LIKE '%[A-Z]%[A-Z]%' AND
        new.password LIKE '%[0-9]%[0-9]%' AND
        new.password LIKE '%[~!@#$%^&]%[~!@#$%^&]%' AND
        length(new.password) >= 8) THEN RETURN TRUE;
    ELSE
        RETURN FALSE;
END IF;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER p_trigger
BEFORE INSERT ON t_passw
FOR EACH ROW
EXECUTE PROCEDURE check_passw();
```

```sql
INSERT INTO t_passw(password)
VALUES ('HOLIDAY8');

DROP TABLE t_passw;
DROP FUNCTION check_passw();
DROP TRIGGER p_trigger ON t_passw;



-- 3
-- What is the difference between procedure and function
-- Function is used to calculate something from a given input.
-- While procedure is the set of commands, which are executed in a order.


-- 4
CREATE TABLE employee(id int, name varchar, date_of_birth date, age int,
salary int, workexperince int, discount int);
-- a) Increases salary by 10% for every 2 years of work experience and
provides
-- 10% discount and after 5 years adds 1% to the discount.
CREATE OR REPLACE PROCEDURE bonus()
AS $$
BEGIN
    UPDATE employee
    SET salary = employee.salary * (employee.workexperince/2 + 1),
    discount = 10
    WHERE workexperince >= 2;

    UPDATE employee SET discount = discount + (workexperince / 5)
    WHERE workexperince >= 5;

    COMMIT ;
END; $$
LANGUAGE plpgsql;

CALL bonus();

INSERT INTO employee VALUES (1,'Khalida', '2002-12-05', 20, 50000,12, 0);

-- b) After reaching 40 years, increase salary by 15%. If work experience is
more
-- than 8 years, increase salary for 15% of the already increased value for
work
-- experience and provide a constant 20% discount.
CREATE OR REPLACE PROCEDURE bonus2()
AS $$
BEGIN
    UPDATE employee
    SET salary = employee.salary * 1.15
    WHERE age >= 40;

    UPDATE employee SET discount = 20,
                        salary = salary * 1.15
    WHERE workexperince >= 8 AND age >= 40;

    COMMIT ;
END;$$
LANGUAGE plpgsql;
CALL bonus2();

INSERT INTO employee VALUES ( 2, 'Khamit', '1965-08-03', 51, 40000, 9,0);
```

```sql
-- 5
CREATE TABLE members(memid integer,
                        surname character varying(200),
                        firstname character varying(200),
                        address character varying(300),
                        zipcode integer,
                        telephone character varying(20),
                        recommendedby integer,
                        joindate timestamp);

CREATE TABLE bookings(facid integer,
                        memid integer,
                        starttime timestamp,
                        slots integer);

CREATE TABLE facilities (facid integer,
                            name character varying(100),
                            memercost numeric,
                            guestcost numeric,
                            initialoutlay numeric,
                            monthlymaintenance numeric);


WITH RECURSIVE recommenders(recommender, member) AS (
SELECT recommendedby, memid
FROM members
UNION ALL
SELECT mems.recommendedby, recs.member
FROM recommenders recs
INNER JOIN members mems
ON mems.memid = recs.recommender
)

SELECT recs.member member, recs.recommender, mems.firstname, mems.surname
FROM recommenders recs
INNER JOIN members mems
ON recs.recommender = mems.memid
WHERE recs.member = 22 OR recs.member = 12
ORDER BY recs.member ASC, recs.recommender DESC
```