Learning Django Framework, Python, Postgresql and what I have done so far.

Installing Python and pip

Downloaded and installed Python 3.6.0 for Windows 10.

Tried out pip by installing beautifulsoup4, requests and csvkit.

http://www.anthonydebarros.com/2015/08/16/setting-up-python-in-windows-10/

While coding with Visual Studio Code I installed pylint that added integrated python terminal to it.

I still need to figure out better way to run the code than copy paste.

Pyexcel and datasheet file conversion

I wanted to learn way to convert Microsoft Office Excel (xsl), LibreOffice Calc (ods) and CSV-files so data could be unified and later saved for database. I had an excel file that contains list of post addresses of my acquaintances. I used it for this.

https://pyexcel.readthedocs.io/en/latest/design.html
https://pypi.python.org/pypi/pyexcel-ods/0.3.0

I had to install additional packages besides pyexcel for each new file extension. There were also conversions to Json, Html and Mediawikia that could be used.
https://pyexcel.readthedocs.io/en/latest/design.html#data-format

Pyexcel handles conversion based on file extension so I ended up with following code:

```
import pyexcel as pe
folder = "pyexcel/files/"
filename = folder + "osoitteet_original.ods"
destfilename = folder + "osoitteet_original"

def saveFileAsOtherType(fname, destfname):
    "Save file in other format"
    pe.save_as(file_name=fname, dest_file_name=destfname)
    return

# save file as csv
saveFileAsOtherType(filename, destfilename+".csv")
```

PostgreSQL as database

Installed PostgreSQL using this website:
http://www.postgresqltutorial.com/install-postgresql/

I had to reinstall PostgreSQL twice, because I had not created postgres-user for installation on first time.

Psycopg2 and database connection

I used Pyscopg2 to setup database connection.
https://wiki.postgresql.org/wiki/Psycopg2_Tutorial

I had to remind myself with these links. It has been a while since I wrote SQL without additional tools, and my last project was using MongoDB which lead to new set of unexpected challenges.
https://www.postgresql.org/docs/9.1/static/sql-createtable.html
http://www.postgresqltutorial.com/postgresql-python/create-tables

The python code that uses Psycogp2 for CRUD commands is added as **appendix 1.**

I think I have learned basics of how to use Psycopg2 for basic CRUD commands. I also learned one way to setup PostgreSQL connection with Python.

ConEmu and batch file (as side project)

Installed ConEmu and setup it to launch my Angular project. First startup MongoDB, then after 10 seconds Mongo and Angular project with Node.
https://conemu.github.io/

After that I created batch file that would do the same and then opening http:/localhost:3000 on browser. By using that I could start project server with one click.

```
rem Start mongod for database and wait 10 seconds
start "MongoDB" "%ProgramFiles%\MongoDB\Server\3.2\bin\mongod.exe" --dbpath "c:\data"
timeout 10

rem Start mongo for database command line
start "Mongo" cmd /k "%ProgramFiles%\MongoDB\Server\3.2\bin\mongo.exe" & title Mongo

rem Start our chosen node.js app (TaMa, TaskManager App)
start "Node.js TaMa" cmd /k node C:\nodeprojects\Ryhmatyo\TaMa\app.js

rem Open TaskManager App in browser
start "" http://localhost:3000
```

Setting up Django

Install and test that Django is working.
https://docs.djangoproject.com/en/1.10/intro/tutorial01/

Then I created models (models.py) for question and choice, and then migrated models.

Cmd commands used:

python manage.py makemigrations polls	Tells made changes in models to Django for migration.
python manage.py sqlmigrate polls 0001	Shows SQL that migration would run.
python manage.py check	Checks for any problems in project without migrating.
python manage.py migrate	Takes all undone migrations and applies them to database.

Next step:
Playing with the API

https://docs.djangoproject.com/en/1.10/intro/tutorial02/

Appendix 1:

```
#!
import psycopg2
from config import config

def connect():
    """Connect to the PostgreSQL database"""
    conn = None
    try:
        # read connection parameters
        params = config()
        # connect to server
        print('Connecting to PostgreSQL database..')
        conn = psycopg2.connect(**params)
        # create cursor
        cur = conn.cursor()
        # execute a statement
        print('PostgreSQL database version:')
        cur.execute('SELECT version()')
        # display the PostgreSQL database server version
        db_version = cur.fetchone()
        print(db_version)
        # close the communication with the PostgreSQL
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
            print('Database connection closed.')

def create_tables():
    """Create table Addresses"""
    command = (
        """
        CREATE TABLE Addresses (
            aid SERIAL PRIMARY KEY,
            name VARCHAR(50) NOT NULL,
            address VARCHAR(50) NOT NULL,
            postnumber VARCHAR(5) NOT NULL,
            postoffice VARCHAR(50) NOT NULL,
            phonenumber VARCHAR(10),
            comments VARCHAR(90)
        )
        """
    )
    try:
        # read connection parameters
```

```
        params = config()
        # connect to server
        print('Connecting to PostgreSQL database..')
        conn = psycopg2.connect(**params)
        # create cursor
        cur = conn.cursor()
        # execute
        print('Creating table Addresses.')
        cur.execute(command)
        # close communication
        cur.close()
        # commit the changes
        conn.commit()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
            print('Database connection closed.')

# CRUD, Create Read Update Delete

# insert data
def insert_address(name, address, postnumber, postoffice, phonenumber, comments):
    """Insert a new address to Addresses table"""
    sql = """INSERT INTO Addresses(name, address, postnumber, postoffice, phonenumber, comments)
             VALUES(%s, %s, %s, %s, %s, %s) RETURNING aid;"""
    conn = None
    aid = None
    try:
        # read database configuration
        params = config()
        # connect to the PostgreSQL database
        conn = psycopg2.connect(**params)
        # create a new cursor
        cur = conn.cursor()
        # execute the INSERT statement
        cur.execute(sql, (name, address, postnumber, postoffice, phonenumber, comments))
        # get the generated id back
        aid = cur.fetchone()[0]
        # commit the changes to the database
        conn.commit()
        # close communication with the database
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
```

```
    return aid

# insert multiple data
def insert_multiple_addresses(address_list):
    """Insert a new address to Addresses table"""
    insert_query = """INSERT INTO Addresses(name, address, postnumber, postoffice, phonenumber,
comments)
            VALUES {0}"""
    conn = None
    aid = None
    try:
        # read database configuration
        params = config()
        # connect to the PostgreSQL database
        conn = psycopg2.connect(**params)
        # create a new cursor
        cur = conn.cursor()
        # execute the INSERT statement
        args = address_list
        records_list_template = ','.join(['%s'] * len(args))
        insert_query = 'INSERT INTO Addresses(name, address, postnumber, postoffice, phonenumber,
comments) VALUES {0}'.format(records_list_template)
        cur.execute(insert_query, args)
        # commit the changes to the database
        conn.commit()
        # close communication with the database
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()

# select data
def get_addresses():
    """ query data from the Addresses table """
    conn = None
    try:
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()
        cur.execute("SELECT aid, name, address, postnumber, postoffice FROM Addresses ORDER BY
name")
        print("The number of addresses: ", cur.rowcount)
        row = cur.fetchone()
        # loop through addresses
        while row is not None:
            print(row)
            row = cur.fetchone()
```

```
        # close connection
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()

# update data
def update_address(address_id, name, address):
    """ update Address name based on the address id """
    sql = """ UPDATE Addresses
                SET name = %s, address = %s
                WHERE aid = %s"""
    conn = None
    updated_rows = 0
    try:
        # read database configuration
        params = config()
        # connect to the PostgreSQL database
        conn = psycopg2.connect(**params)
        # create a new cursor
        cur = conn.cursor()
        # execute the UPDATE  statement
        cur.execute(sql, (name, address, address_id))
        # get the number of updated rows
        updated_rows = cur.rowcount
        # Commit the changes to the database
        conn.commit()
        # Close communication with the PostgreSQL database
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
    return updated_rows


# delete data
def delete_address(address_id):
    """Delete address by address id"""
    conn = None
    rows_deleted = 0
    try:
        # read database configuration
        params = config()
        # connect to the PostgreSQL database
        conn = psycopg2.connect(**params)
```

```
        # create a new cursor
        cur = conn.cursor()
        # execute the UPDATE  statement
        cur.execute("DELETE FROM Addresses WHERE aid = %s", (address_id,))
        # get the number of updated rows
        rows_deleted = cur.rowcount
        # Commit the changes to the database
        conn.commit()
        # Close communication with the PostgreSQL database
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()
    return rows_deleted

if __name__ == '__main__':
    # create table
    create_tables()
    # insert one row
    inserted_id = insert_address("Joni Viholainen", "Kauppakatu 2", "40251", "Jyväskylä", "05921052",
"omat tiedot")
    print('The id of inserted row: ', inserted_id)
    # insert multiple rows
    insert_multiple_addresses([
        ("Joni Viholainen", "Ankeriastie 30", "54921", "Taipalsaari", "2357999999", "lomaosoite"),
        ("Matti Meikäläinen", "Helsingintie 13", "53022", "Lappeenranta", "3525255555", "all ok"),
        ("Heimo Huima", "Alvar Aallon katu 92", "32365", "Mikkeli", "3333333336", "virheelliset
tiedot")
    ])
    # query addresses
    get_addresses()
    # update row
    updated_rows = update_address(14, "Jani Virolainen", "Virokatu 11")
    print('The number of updates rows: ', updated_rows)
    # delete row
    deleted_rows = delete_address(14)
    print('The number of deleted rows: ', deleted_rows)
```