

МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота № 1**

з дисципліни “ Компоненти програмної інженерії 1. Основи розроблення ПЗ”

|  |  |  |
| --- | --- | --- |
| Виконали  студенти II курсу  групи КП-72  Арнаутова Ганна Сергіївна,  Демченко Владислав Миколайович,  Чумак Михайло Сергійович  (*прізвище, ім’я, по батькові*)  варіант № 10 |  | Перевірив  “\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.  доцент  Замятін Денис Станіславович  (*прізвище, ім’я, по батькові*) |

Київ 2019

**Мета роботи**

Оволодіти процедурними конструкціями мови Python та основним інструментарієм розробника.

**Постановка завдання**

1. Розробити програмні засоби мовою програмування Python 3 згідно з обраним варіантом завдання. Розроблений програмний код повинен складатися з окремих функцій, кожна з яких має не більше 10 рядків та не більше 3 рівнів вкладеності. Розроблені засоби мають бути інтерактивними та надавати користувачу можливість взаємодії з ними через консоль. Дані для роботи програми повинні зберігатися у файлах в форматі JSON.

2. Виконати декомпозицію розробленого програмного коду на окремі модулі, які відповідають за бізнес-логіку, введення-виведення та управління порядком виконання (контролер).

3. Додати до кожної функції документуючий коментар та набір док-тестів. Покриття програмного коду тестами повинно складати не менше 80%.

4. За допомогою засобів статичного аналізу pep8 та pyflakes забезпечити сумісність програмного коду із стандартами кодування.

5. Використовуючи фреймворк flask-restful додати реалізацію REST-інтерфейсу для розроблених програмних засобів. Написати тести до REST-інтерфейсу, використовуючи бібліотеки pytest та requests.

6. Розмістити розроблені програмні засоби у публічному сховищі програмного коду (github, bitbucket тощо).

* **Варіант 10:**

*Програма планування графіку поливу кімнатних рослин.*

* **Посилання на репозиторій:**

[*https://github.com/aarnautova/Python*](https://github.com/aarnautova/Python)*\_Lab1*

1. **Приклад сесії взаємодії з розробленими програмними засобами**

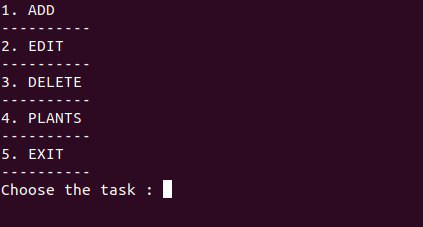


Рис. 1 Головне Меню

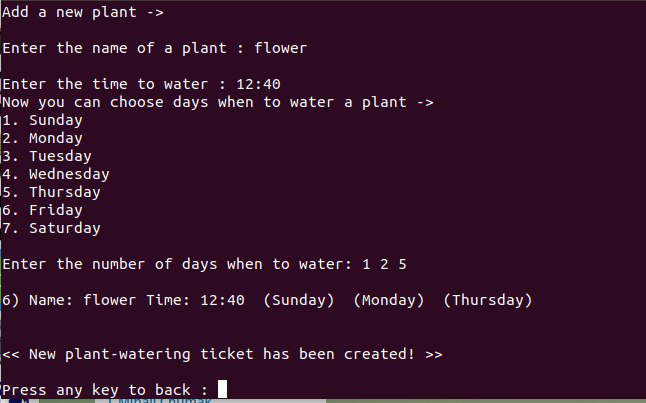


Рис. 2 Меню Додавання

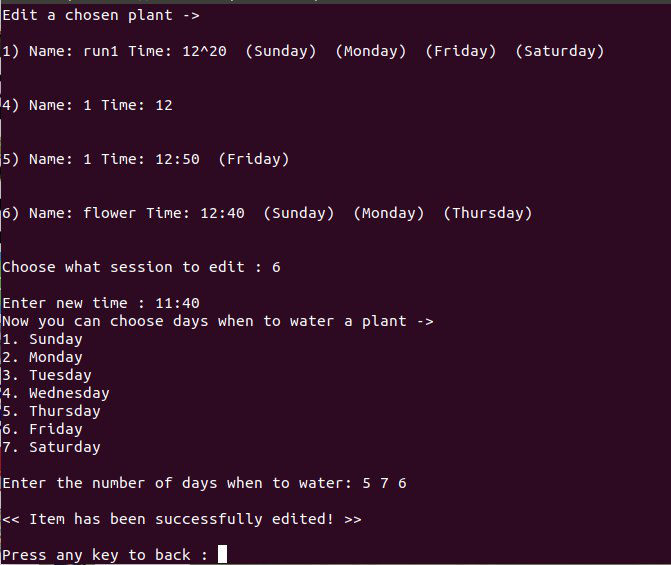


Рис. 3 Меню Редагування

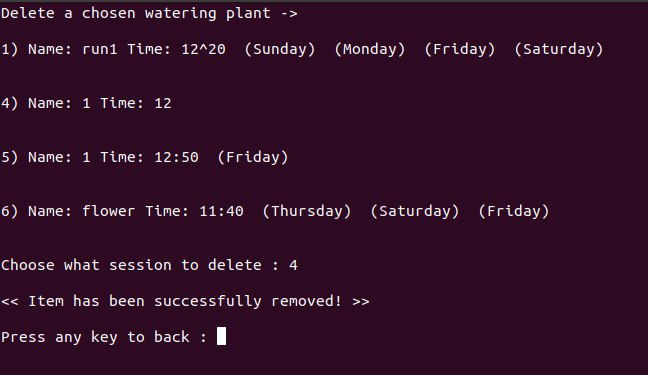


Рис. 4 Меню Видалення

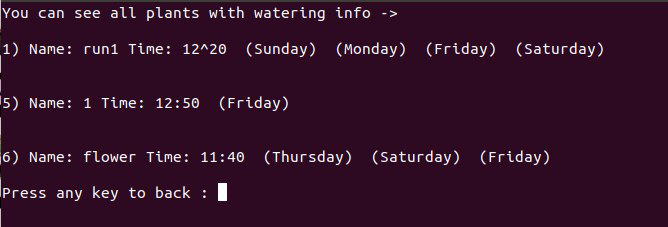


Рис. 5 Меню Перегляду Інформації

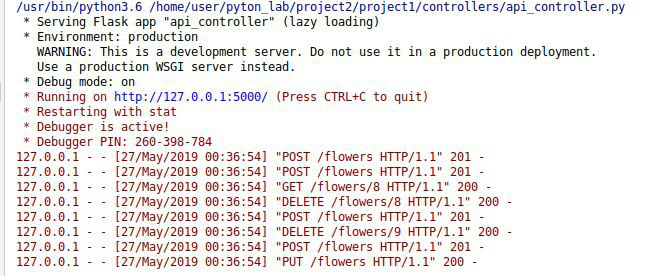


Рис. 6 Робота з REST-інтерфейсом

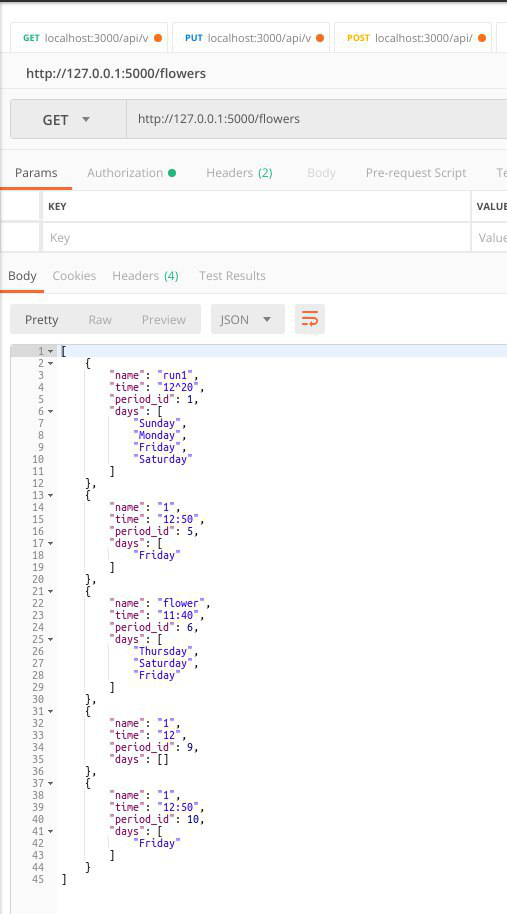


Рис. 7 Відповідь на GET-запит

1. Згенерована документація

|  |
| --- |
| import json  class WateringPeriod:  """ Class of watering period information """ def \_\_init\_\_(self, name, time, days, period\_id=0):  """Create object of WateringPeriod class  >>> wp1 = WateringPeriod("TEST", "12:00", ["Monday", "Tuesday"])  >>> wp2 = WateringPeriod("TEST", "12:00", ["Monday", "Tuesday"])  >>> wp3 = WateringPeriod("TEST1", "17:00", [])  >>> wp1.\_\_eq\_\_(wp2)  True  >>> wp1.\_\_eq\_\_(wp3)  False  """  self.period\_id = period\_id  self.name = name  self.time = time  self.days = days  def info(self):  """Prints information about watering period object.  >>> wp = WateringPeriod("TEST", "12:00", [])  >>> wp.info()  '0) Name: TEST Time: 12:00 '  >>> wp1 = WateringPeriod("TEST", "12:00", ["Friday", "Monday"])  >>> wp1.info()  '0) Name: TEST Time: 12:00 (Friday) (Monday) '  """  result = "{}) Name: {} Time: {} ".format(self.period\_id, self.name, self.time)  for day in self.days:  result += "(" + day + ") "  return result  def \_\_dict\_\_(self):  """Convert WateringPeriod object to dict  >>> wp = WateringPeriod("TEST", "12:00", [])  >>> wp.\_\_dict\_\_()  {'name': 'TEST', 'time': '12:00', 'period\_id': 0, 'days': []}  >>> wp = WateringPeriod("TEST", "12:00", ["Thursday", "Friday"])  >>> wp.\_\_dict\_\_()  {'name': 'TEST', 'time': '12:00', 'period\_id': 0, 'days': ['Thursday', 'Friday']}  """  return {"name": self.name,  "time": self.time,  "period\_id": self.period\_id,  "days": self.days}  def \_\_eq\_\_(self, other):  """Compare objects of WateringPeriod class  >>> wp1 = WateringPeriod("TEST", "12:00", [])  >>> wp2 = WateringPeriod("TEST", "12:00", [])  >>> wp3 = WateringPeriod("TEST1", "12:00", [])  >>> wp1.\_\_eq\_\_(wp2)  True  >>> wp1.\_\_eq\_\_(wp3)  False  """  return self.name == other.name and self.days == other.days and self.time == other.time  class WateringPeriodEncoder(json.JSONEncoder):  def default(self, obj):  if isinstance(obj, WateringPeriod):  return obj.\_\_dict\_\_  return json.JSONEncoder.default(self, obj)  def ob\_from\_dict(d):  """Convert dict to WateringPeriod object  >>> d1 = {"name": "TEST", "time": 0, "period\_id": 0, "days": []}  >>> wp1 = WateringPeriod("TEST", "12:00", [])  >>> wp1.\_\_eq\_\_(ob\_from\_dict(d1))  False  >>> d2 = {"name": "TEST", "time": "12:00", "period\_id": 0, "days": ["Friday", "Thursday"]}  >>> wp2 = WateringPeriod("TEST", "12:00", ["Friday", "Thursday"])  >>> wp2.\_\_eq\_\_(ob\_from\_dict(d2))  True  """  return WateringPeriod(d.get('name'),  d.get('time'),  d.get('days'),  d.get('period\_id'))  def list\_from\_dict(dictionary):  """Convert dict to list of WateringPeriod objects >>> d1 = [ {'name': "TEST", 'time': "23:00", 'period\_id': 0, 'days': []}] >>> list\_from\_dict(d1) == [WateringPeriod("TEST", "23:00", [])] True """ ob\_list = list() for item in dictionary:  ob\_list.append(ob\_from\_dict(item))  return ob\_list  # python -m doctest -v watering\_period.py if \_\_name\_\_ == "\_\_main\_\_":  import doctest doctest.testmod |

|  |
| --- |
| import data.json\_reader import watering\_period from watering\_period import WateringPeriod  class Storage:  """ Class for data storage operations """ def \_\_init\_\_(self, filename):  """ Create an object of Storage class >>> s1 = Storage("./data/data.json") >>> s2 = Storage("./data/data.json") >>> s3 = Storage("./data/data1.json") >>> s1.\_\_eq\_\_(s2) True >>> s1.\_\_eq\_\_(s3) False """ self.filename = filename self.file\_object = data.json\_reader.load\_dict(filename) self.localScheduleList = watering\_period.list\_from\_dict(  self.file\_object.get("items"))  self.nextId = self.file\_object.get("nextId")  def \_\_eq\_\_(self, other):  """Compare objects of Storage class  >>> s1 = Storage("./data/data.json")  >>> s2 = Storage("./data/data.json")  >>> s3 = Storage("./data/data1.json")  >>> s1.\_\_eq\_\_(s2)  True  >>> s1.\_\_eq\_\_(s3)  False  """  return self.filename == other.filename  def create(self, wp: WateringPeriod):  """ Adds new object to local list  >>> wp = WateringPeriod("Test", 0, 0, [])  >>> s = Storage("./data/data1.json")  >>> s.create(wp).period\_id == s.nextId - 1  True  >>> s.create(wp).period\_id == s.nextId  False  """  wp.period\_id = self.nextId  self.nextId = self.nextId + 1  self.localScheduleList.append(wp)  return wp  def read\_all(self):  """ Returns local list >>> s = Storage("./data/data1.json") >>> wp\_list = [WateringPeriod("Test", "0", []), WateringPeriod("Test1", "12:20", [])] >>> s.read\_all() == wp\_list True  """ return self.localScheduleList  def read(self, period\_id):  """ Returns local list entity with id "period\_id" >>> s = Storage("./data/data1.json") >>> wp1 = WateringPeriod("Test1", "12:20", []) >>> wp2 = WateringPeriod("Test", "12:20", []) >>> s.read(2).\_\_eq\_\_(wp1) True >>> s.read(2).\_\_eq\_\_(wp2) False """ try:  return next(  filter(  lambda x: x.period\_id == period\_id, self.localScheduleList))  except StopIteration:  raise Exception("there aren't such element")  @staticmethod def update\_ob(to\_upd, kwargs):  if kwargs.get('name'):  to\_upd.name = kwargs.get('name')  if kwargs.get('time'):  to\_upd.time = kwargs.get('time')  if kwargs.get('days'):  to\_upd.days = kwargs.get('days') return to\_upd.period\_id  def update(self, period\_id, [\*\*](http://localhost:63342/project1/doc1.html?_ijt=6tmldj77l1k4ltr3e38sodds8i#id1)kwargs):  """ Updates local list entity with id "period\_id"  >>> s = Storage("./data/data1.json")  >>> upd = s.update(1, time="12:20", days=[])  >>> wp = WateringPeriod("Test", "12:20", [])  >>> wp.\_\_eq\_\_(upd)  True  """ try:  to\_upd = next(  filter(  lambda x: x.period\_id == period\_id, self.localScheduleList))  self.update\_ob(to\_upd, kwargs) return to\_upd  except StopIteration:  raise Exception("there aren't such element")  def delete(self, period\_id):  """ Deletes from local list entity with id "period\_id"  >>> s1 = Storage("./data/data1.json")  >>> s2 = Storage("./data/data1.json")  >>> s1.delete(1)  >>> len(s1.read\_all()) == len(s2.read\_all())  False  >>> len(s1.read\_all()) == 1  True  >>> len(s2.read\_all()) == 2  True  """ try:  self.localScheduleList.remove(next(  filter(  lambda x: x.period\_id == period\_id, self.localScheduleList)))  except StopIteration:  raise Exception("there aren't such element")  def delete\_all(self):  """ Clears local list  >>> s1 = Storage("./data/data1.json")  >>> s2 = Storage("./data/data1.json")  >>> s1.delete\_all()  >>> len(s1.read\_all()) == len(s2.read\_all())  False  >>> len(s1.read\_all()) == 0  True  >>> len(s2.read\_all()) == 2  True  """ self.localScheduleList.clear() self.nextId = 0  def save\_session(self):  """ Saves local list to file """ data.json\_reader.write\_file(  self.filename, {"nextId": self.nextId,  "items": [ob.\_\_dict\_\_() for ob in self.localScheduleList]})  # python -m doctest -v ./data/storage.py if \_\_name\_\_ == "\_\_main\_\_":  import doctest doctest.testmod() |

|  |
| --- |
| import json  def load\_file(filename):  with open(filename) as json\_file:  return json\_file  def load\_dict(filename):  try:  with open(filename) as json\_file:  return json.load(json\_file)  except FileNotFoundError:  with open(filename, 'w') as json\_file:  json.dump({"nextId": 0,  "items": []}, json\_file, indent=4)  return json.load(json\_file)  def write\_empty\_file(filename):  with open(filename, 'w') as json\_file:  json.dump({"nextId": 0,  "items": []}, json\_file, indent=4)  def write\_file(filename, data):  with open(filename, 'w') as json\_file:  json.dump(data, json\_file, indent=4) |

|  |
| --- |
| from data.storage import Storage import watering\_period from flask import Flask, request from flask\_restful import Resource, Api  app = Flask(\_\_name\_\_) api = Api(app) storage = Storage("../data/data.json")  class GetAll(Resource):  """ Class of getting all information about watering periods """ @staticmethod def get():  """ Method for HTTP-get request """ list\_ob = storage.read\_all() response = [ob.\_\_dict\_\_() for ob in list\_ob] return response  class GetById(Resource):  """ Class of getting all information about current watering period """ @staticmethod def get(period\_id):  """ Method for HTTP-get request """ return storage.read(period\_id).\_\_dict\_\_()  class Add(Resource):  """ Class of adding information about new watering period """ @staticmethod def post():  """ Method for HTTP-post request """ period = request.get\_json() result = storage.create(  watering\_period.ob\_from\_dict(period)).\_\_dict\_\_()  storage.save\_session() return result, 201  class Update(Resource):  """ Class of adding information about new watering period """ @staticmethod def put():  """ Method for HTTP-put request """ period = request.get\_json() if period.get("period\_id") is None:  return "", 404  storage.update(  period.get("period\_id"), days=period.get("days"), time=period.get("time"))  storage.save\_session() return 200  class Delete(Resource):  """ Class for deleting some watering period """ @staticmethod def delete(period\_id):  """ Method for HTTP-delete request """ storage.delete(period\_id) storage.save\_session() return 200  api.add\_resource(GetAll, '/flowers') api.add\_resource(GetById, '/flowers/<int:id>') api.add\_resource(Add, '/flowers') api.add\_resource(Update, '/flowers') api.add\_resource(Delete, '/flowers/<int:id>')  app.run(debug=True) |

|  |
| --- |
| from data.storage import Storage import watering\_period  storage = Storage("./data/data.json")  def get\_all():  """ Method for getting all information about watering periods"""  list\_ob = storage.read\_all() response = [ob.info() for ob in list\_ob] return response  def get\_by\_id(period\_id):  """ Method for getting all information about watering period by id  >>> get\_by\_id(9)  '9) Name: 1 Time: 12 '  """  return storage.read(period\_id).info()  def add(name, time, days):  """ Method for adding information about new watering period >>> r = add("1", "12", []) >>> r.split(' ', 1)[1] 'Name: 1 Time: 12 ' """ result = storage.create(watering\_period.WateringPeriod(name, time, days)) storage.save\_session() return result.info()  def update(period\_id, time, days):  """ Method for updating information of watering period """ storage.update(period\_id, days=days, time=time) storage.save\_session()  def delete(period\_id):  """ Method for deleting information about watering period """ storage.delete(period\_id) storage.save\_session()  # python -m doctest -v console\_controller.py if \_\_name\_\_ == "\_\_main\_\_":  import doctest doctest.testmod() |

**2. Звіт програм pep8, pyflakes та об’єм покриття тестами**

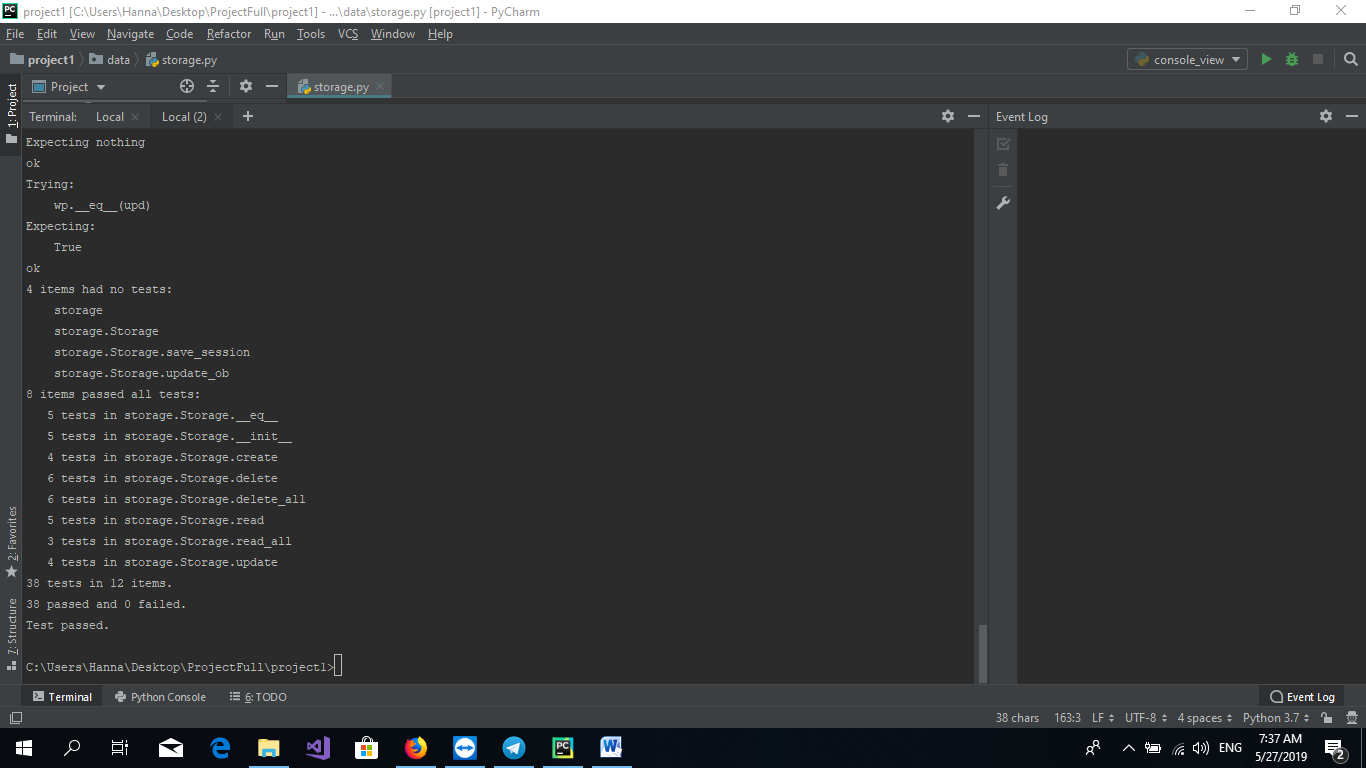


Рис. 8 Покриття тестами модуля storage

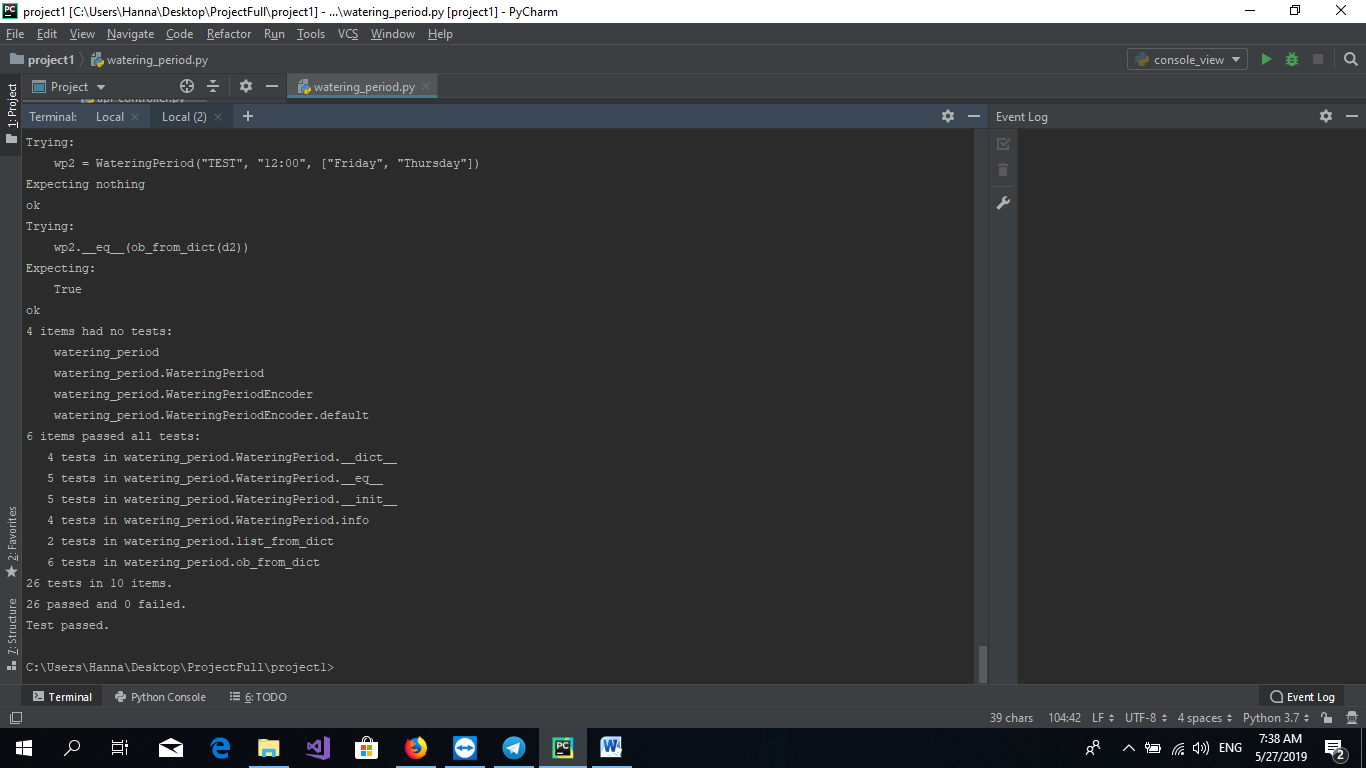


Рис. 9 Покриття тестами модуля watering\_period

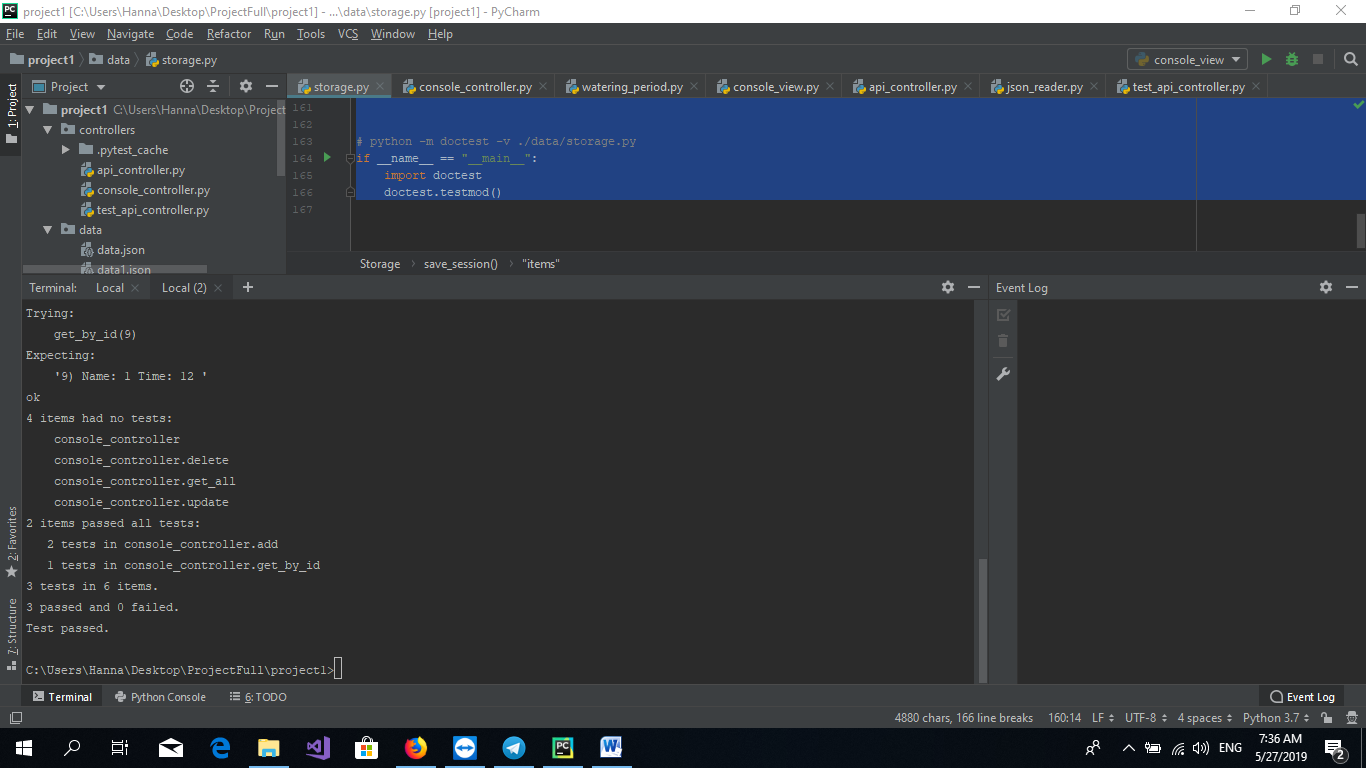


Рис. 10 Покриття тестами модуля console\_controller

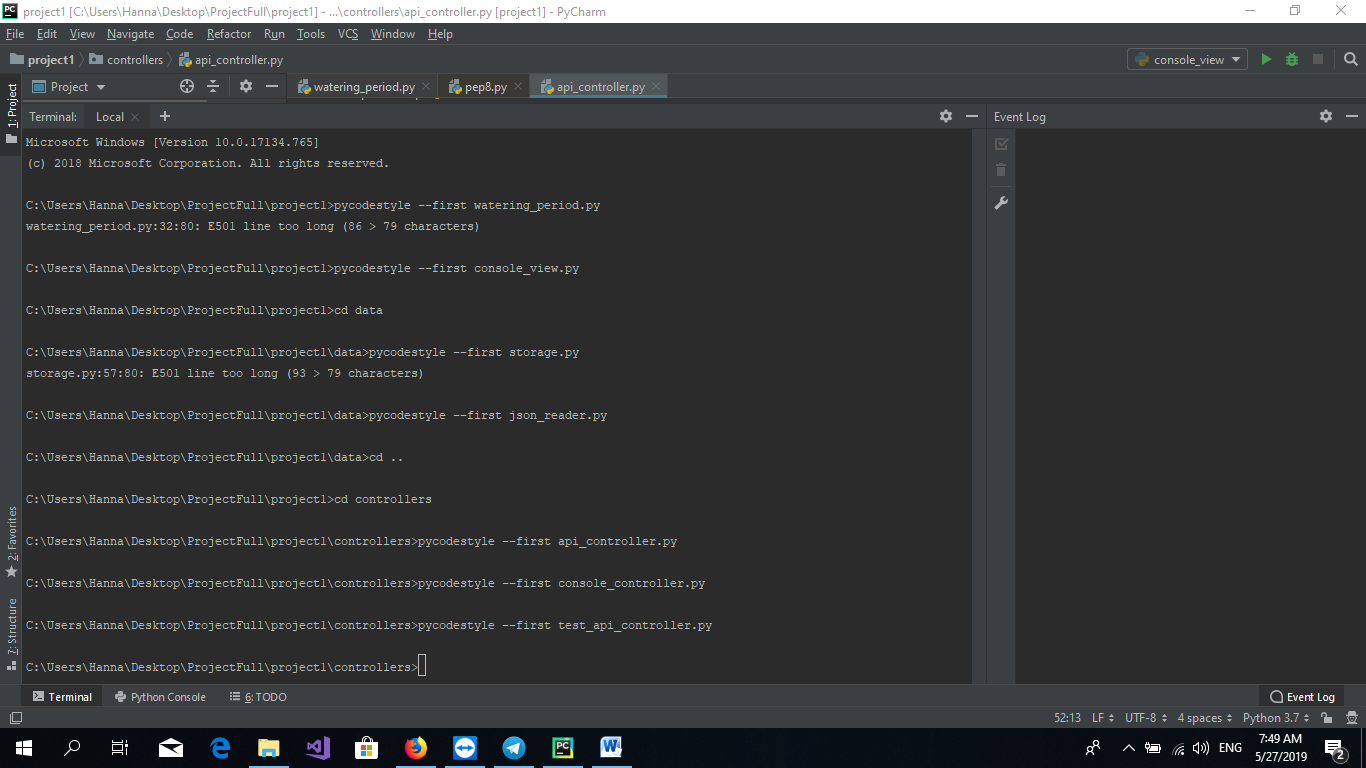


Рис.11 Звіт pep8

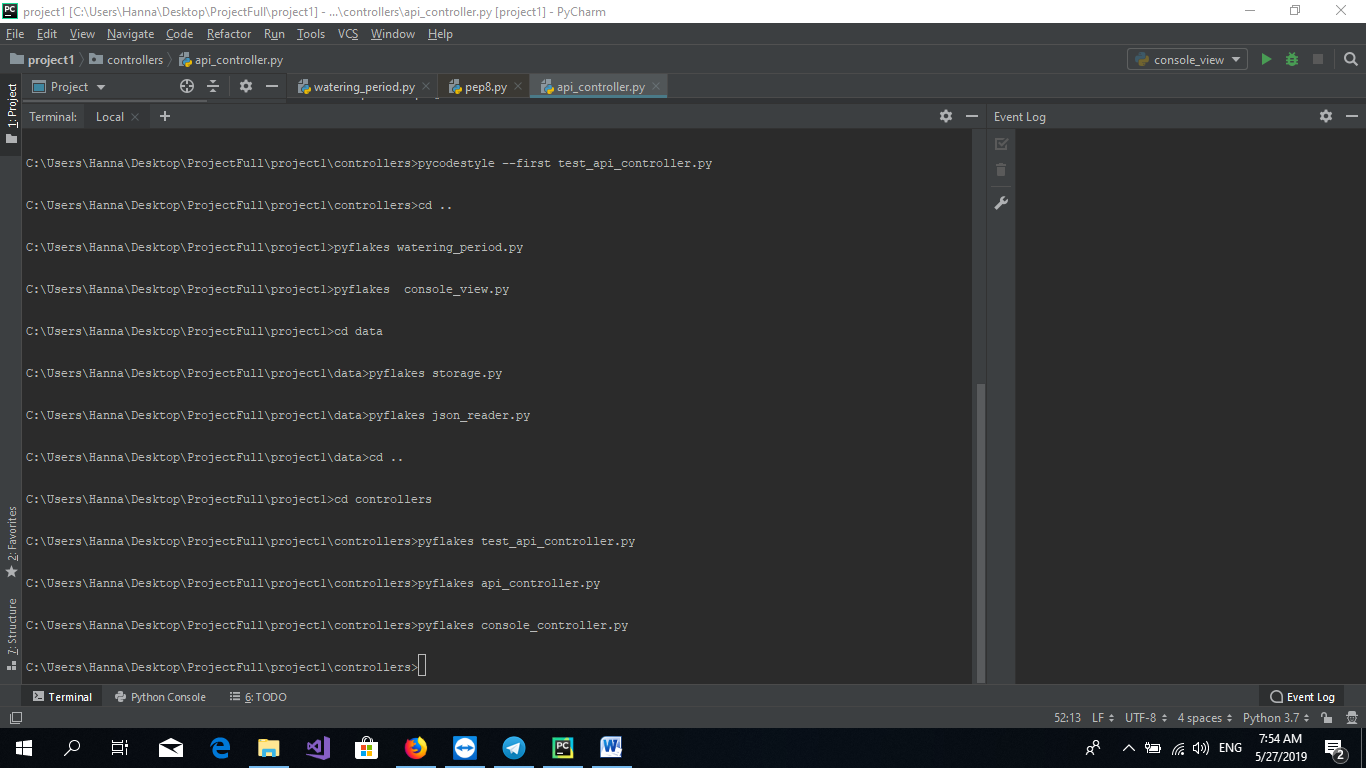


Рис.12 Звіт pyflakes

**Висновки**

Отже, в результаті виконання цієї лабораторної роботи, було опрацьовано процедурні конструкції мови Python та застосовано на практиці основний інструментарій розробника.

Було створено ПЗ, що має консольний інтерфейс, співпрацює з JSON документами та маэ REST-api.

Використано фреймворк flask-restful для реалізації REST-інтерфейсу для розроблених програмних засобів, до якого написано тести, за допомогою бібліотек pytest та requests.

За допомогою засобів статичного аналізу pep8 та pyflakes забезпечено сумісність програмного коду із стандартами кодування.

Розроблені програмні засоби розміщено у публічному сховищі програмного коду github.