**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

# Курс «Базовые компоненты интернет-технологий**»**

Отчет по лабораторной работе №4

«Основные конструкции языка Python»

| Выполнил: |  | Проверил: |
| --- | --- | --- |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Ткаченко В. Л. |  | Гапонюк Ю. Е. |
| Подпись и дата |  | Подпись и дата |

Москва, 2021 г.

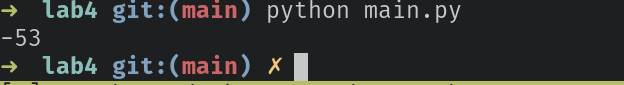
### **Задание:**

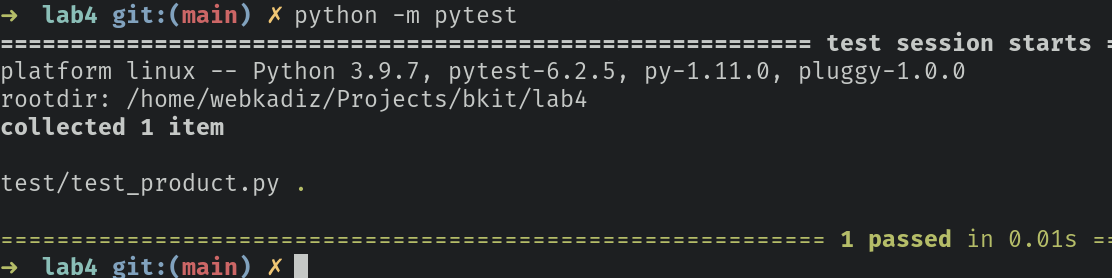
1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать [следующий каталог.](https://refactoring.guru/ru/design-patterns/catalog) Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. Вместо реализации паттерна Вы можете написать тесты для своей программы решения биквадратного уравнения. В этом случае, возможно, Вам потребуется доработать программу решения биквадратного уравнения, чтобы она была пригодна для модульного тестирования.
3. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.

**Листинг программы:**

| ***# compositor.py* from abc import ABC, abstractmethod  class IDietTreeComponent(ABC):  def getPrice(self):  price = 0   for child in self.children:  price += child.getPrice()   return price   @abstractmethod  def children(): pass  *# diet.py* from compositor import IDietTreeComponent from iterator import TreeIterator from dish import Dish  class Diet(IDietTreeComponent):  def \_\_init\_\_(self, category):  self.\_category = category  self.\_children = []   children = property(lambda self: self.\_children)   def addDish(self, dish):  self.children.append(dish)   def \_\_iter\_\_(self):  return TreeIterator().depthDetour(self)   def eatBreakfast(self, client):  breakfast = self.getBreakfast()  breakfast.isEated = True  price = breakfast.getPrice()  client.balance -= price   def getBreakfast(self):  for child in self:  if type(child) == Dish and child.isBreakfast() and not child.isEated:  return child   *# dish.py* from compositor import IDietTreeComponent  class Dish(IDietTreeComponent):  def \_\_init\_\_(self, category):  self.\_children = []  self.\_category = category  self.\_isEated = False   children = property(lambda self: self.\_children)   isEated = property(lambda self: self.\_isEated, lambda self, val: setattr(self, '\_isEated', val))   def isBreakfast(self):  return self.\_category == 'breakfast'   def addProduct(self, product):  self.\_children.append(product)  return self%   *# facade.py* from diet import Diet from dish import Dish from product import Product   products = [  oatgroats:= Product(dict(name="Овсяная крупа", priceForWeight=30, weight=1)),  water:= Product(dict(name="Вода", priceForWeight=23, weight=1)),  cabbage:= Product(dict(name="Капуста", priceForWeight=123, weight=1)),  beet:= Product(dict(name="Свекла", priceForWeight=333, weight=1)),  fish:= Product(dict(name="Рыба", priceForWeight=500, weight=1)), ]  class DietKeto():  def \_\_init\_\_(self):  self.\_diet = Diet('keto')  self.\_diet.addDish(  Dish('breakfast')  .addProduct(oatgroats.clone())  .addProduct(water.clone()),  )  self.\_diet.addDish(  Dish('launch')  .addProduct(water.clone())  .addProduct(cabbage.clone())  .addProduct(beet.clone()),  )  self.\_diet.addDish(  Dish('dinner')  .addProduct(fish.clone())  )   def eatBreakfast(self, client):  self.\_diet.eatBreakfast(client)%   *# iterator.py* from product import Product  class TreeIterator():  def depthDetour(self, tree):  for child in tree.children:  yield child  yield from self.depthDetour(child)%   *# product.py* from compositor import IDietTreeComponent from prototype import IPrototype  class Product(IDietTreeComponent, IPrototype):  def \_\_init\_\_(self, params):  self.\_params = params.copy()  self.\_name = params.get('name')  self.\_category = params.get('category')  self.\_proteins = params.get('proteins')  self.\_fats = params.get('fats')  self.\_carbs = params.get('carbs')  self.\_priceForWeight = params.get('priceForWeight')  self.\_weight = params.get('weight')   children = property(lambda self: [])   def getPrice(self):  return self.\_priceForWeight \* self.\_weight   def clone(self):  return Product(self.\_params)%  *# prototype.py* from abc import ABC, abstractmethod  class IPrototype(ABC):  @abstractmethod  def clone(): pass%  *# client.py*  class Client():  def \_\_init\_\_(self):  self.\_diet = None  self.\_balance = 0   @property  def diet(self):  return self.\_diet   @diet.setter  def diet(self, value):  self.\_diet = value   @property  def balance(self):  return self.\_balance   @balance.setter  def balance(self, value):  self.\_balance = value   def eatBreakfast(self):  self.diet.eatBreakfast(self)%   *# main.py* from client import Client from facade import DietKeto  client = Client() keto = DietKeto()  client.diet = keto  client.eatBreakfast()  print(client.balance)%  *# test/test\_product.py* import pytest from product import Product  def test\_check\_get\_price():  product = Product(dict(name="Картошка", weight=2, priceForWeight=30))   res = product.getPrice()   assert res == 60  def test\_check\_get\_price2():  product = Product(dict(name="Картошка", weight=2, priceForWeight=40))   res = product.getPrice()   assert res == 70** |
| --- |

**Пример работы:**

****

****