

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА** - **Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**«Онтология»**

**по дисциплине**

**«Системный анализ данных СППР»**

Студент группы: ИКБО-04-22 Основин А.И. *(Ф. И.О.студента)*

Преподаватель \_\_Железняк Л.М.\_\_ *(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc181777729)

[1 ОНТОЛОГИЯ 4](#_Toc181777730)

[1.1 Цель практической работы 4](#_Toc181777731)

[1.2 Постановка задачи 5](#_Toc181777732)

[1.3 Описание онтологии 5](#_Toc181777733)

[1.4 Построение онтологии в Protégé 6](#_Toc181777734)

[1.5 Выполнение запросов в Protege 9](#_Toc181777735)

[1.6 Результаты выполнения программного кода 10](#_Toc181777736)

[ЗАКЛЮЧЕНИЕ 11](#_Toc181777737)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 12](#_Toc181777738)

[ПРИЛОЖЕНИЯ 13](#_Toc181777739)

# ВВЕДЕНИЕ

Возникновение онтологий и их стремительное развитие связано с проявлением в нашей реальности следующих новых факторов:

* колоссальный рост объемов информации, предъявляемых для обработки (анализа, использования) специалистам самых различных областей деятельности;
* чрезвычайная зашумленность этих потоков (повторы, противоречивость, разноуровневость, и т.п.);
* острая необходимость в использовании одних и тех же знаний разными специалистами в разных целях;
* всеобщая интернетизация нашей жизни и острая необходимость в структуризации информации для её представления пользователям и более эффективного поиска;
* необходимость сокращения времени на поиск нужной информации и повышения качества информационных услуг в Интернете.

Онтологии – это базы знаний специального типа, которые могут читаться и пониматься, отчуждаться от разработчика и/или физически разделяться их пользователями.

Существует много видов онтологий, однако одним из самых широко применяемых видов являются онтологии предметных областей, содержащие понятия определённой области знаний или входящих в неё областей.

# 1 онтология

Построение баз знаний является основой формализации экспертных систем, и одним из наиболее четких примеров баз знаний является онтология. Онтология направлена на стандартизацию и унификацию представления знаний. В онтологии описываются основные элементы предметной области, организованные в виде тройки: где:

X — конечное множество концептов (понятий, терминов) предметной области;

R — конечное множество отношений между этими концептами;

F — конечное множество функций интерпретации (аксиом), которые определяют взаимосвязи и правила, действующие на концептах и отношениях.

В формализованном виде онтологии представляются через:

Классы — абстракции, описывающие категории объектов или понятий;

Слоты (атрибуты) — характеристики или свойства, которыми обладают классы;

Экземпляры — конкретные объекты, являющиеся представителями классов.

Такое формальное представление онтологий позволяет моделировать знания в системах искусственного интеллекта и обеспечивать их совместное использование и обработку.

## 1.1 Цель практической работы

Целью данной практической работы является ознакомление с построением некоторой базы знаний на основе подобия фреймовой системы, а также ее реализация, анализ и применение на практике.

## 1.2 Постановка задачи

Необходимо разработать онтологию выбранной предметной области – «Онлайн магазин одежды». Данная предметная область была выбрана из-за личного интереса к теме.

## 1.3 Описание онтологии

Рассматриваемый онлайн магазин создан для продажи мужской одежды стиля кэжуал (офисной, официальной). Такой одеждой являются штаны, рубашки и туфли. Ассортимент одежды предоставляют магазины (продавцы), зарегистрированные на площадке. У каждого продавца собственный ассортимент товаров. Пользователи площадки могут выбрать товары, исходя из ассортимента всего онлайн магазина. У каждого пользователя есть список приобретённых товаров.

На основе этого описания можно составить онтологию, состоящую из следующих классов:

* «Онлайн магазин одежды» — общий базовый класс для всех классов;
* «Продукт» — базовый класс для разных видов товаров, представленных в онлайн магазине, содержит общие слоты «Бренд», «Модель», «Цена», «Цвет», «Размер»;
* «Штаны» — класс для описания штанов, содержит слоты «Тип посадки» и «Хорошо сидит с», ссылающийся на экземпляр класса «Туфли»;
* «Рубашка» — класс для описания рубашек, содержит слоты «Количество пуговиц» и «Стильно смотрится с», ссылающийся на экземпляр класса «Штаны»;
* «Туфли» — класс для описания туфель, содержит слот «Высота каблука»;
* «Пользователь» — базовый класс для всех видов пользователей, содержит общие слоты «Имя» и «Активен» (для продавца – продаёт ли он сейчас товары; для покупателя – покупает ли он товары);
* «Продавец» — класс для описания продавца (магазина с собственным ассортиментом), содержит слот «Список товаров», ссылающийся на несколько экземпляров класса «Продукт»;
* «Покупатель» — класс для описания покупателя, содержит слот «Выкуп» и «Список покупок», ссылающийся на несколько экземпляров класса «Продукт».

Данное описание использовано для построения графической схемы онтологии (Рисунок 1.3.1).

Изображение выглядит как круг, снимок экрана, луна, ночь

Автоматически созданное описание

Рисунок 1.3.1 – Схема онтологии «Онлайн магазин одежды»

## 1.4 Построение онтологии в Protégé

Для подробного изучения составленной онтологии использован инструмент для построения, редактирования онтологий и работы с ними Protégé. Сначала созданы классы (Рисунок 1.4.1), а затем в них описаны слоты (Рисунок 1.4.2–1.4.6).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 1.4.1 – Составленная иерархия классов

Изображение выглядит как снимок экрана, текст, линия, число

Автоматически созданное описание

Рисунок 1.4.2 – Слоты класса «Штаны»

Изображение выглядит как снимок экрана, текст, линия, число

Автоматически созданное описание

Рисунок 1.4.3 – Слоты класса «Рубашка»

Изображение выглядит как снимок экрана, текст, линия, число

Автоматически созданное описание

Рисунок 1.4.4 – Слоты класса «Туфли»

Изображение выглядит как текст, снимок экрана, линия, число

Автоматически созданное описание

Рисунок 1.4.5 – Слоты класса «Продавец»

Изображение выглядит как снимок экрана, текст, линия, число

Автоматически созданное описание

Рисунок 1.4.6 – Слоты класса «Покупатель»

После составления и описания классов созданы экземпляры каждого из классов (Рисунок 1.4.7-1.4.11).

Изображение выглядит как текст, линия, число, программное обеспечение

Автоматически созданное описание

Рисунок 1.4.7 – Экземпляры класса «Штаны»

Изображение выглядит как текст, линия, число, программное обеспечение

Автоматически созданное описание

Рисунок 1.4.8 – Экземпляры класса «Рубашка»

Изображение выглядит как текст, линия, число, программное обеспечение

Автоматически созданное описание

Рисунок 1.4.9 – Экземпляры класса «Туфли»

Изображение выглядит как текст, линия, программное обеспечение, снимок экрана

Автоматически созданное описание

Рисунок 1.4.10 – Экземпляры класса «Продавец»

Изображение выглядит как текст, программное обеспечение, линия, число

Автоматически созданное описание

Рисунок 1.4.11 – Экземпляры класса «Покупатель»

## 1.5 Выполнение запросов в Protege

Программа Protégé позволяет составлять запросы на получение объектов по определённым условиям, а также вытаскивать связанные объекты для уже полученных объектов. Проделаны обычные запросы на получение экземпляров (Рисунок 1.5.1), а также сделаны цепные запросы на получение связанных объектов (Рисунок 1.5.2–1.5.3).

Изображение выглядит как текст, программное обеспечение, снимок экрана

Автоматически созданное описание

Рисунок 1.5.1 – Одинарный запрос на получение экземпляров класса «Штаны»

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.5.2 – Цепной запрос на получение рубашек, смотрящихся стильно со штанами, которые хорошо сидят с туфлями

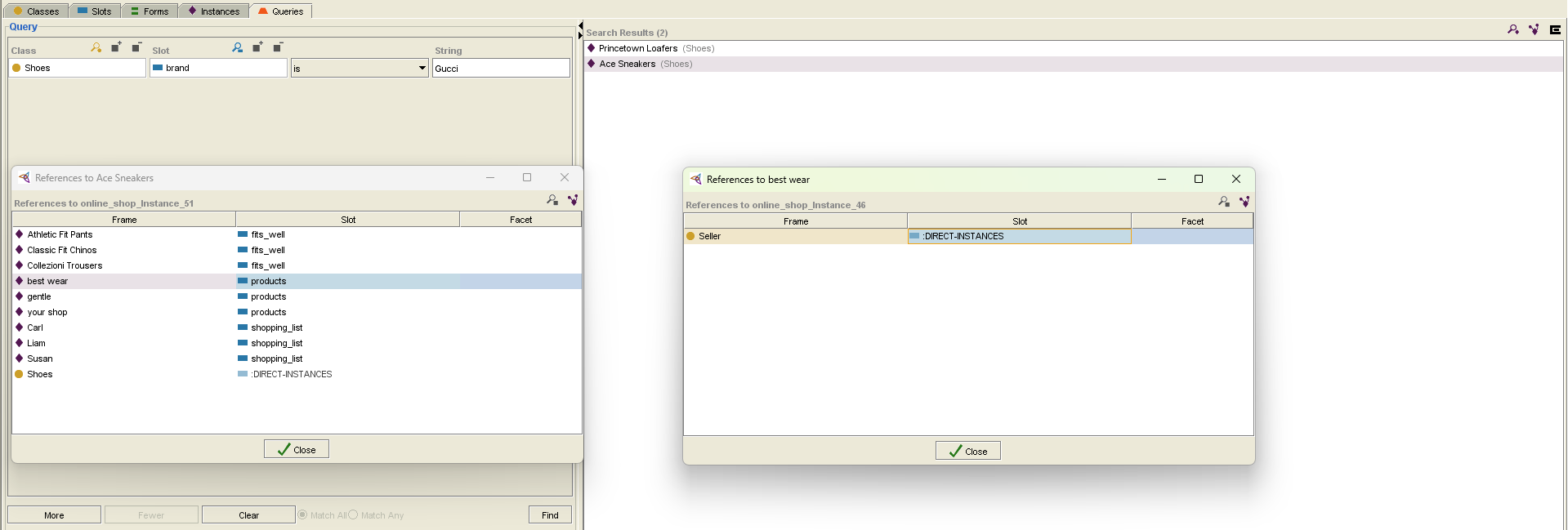


Рисунок 1.5.3 – Цепной запрос на получение магазина, продающего туфли

## 1.6 Результаты выполнения программного кода

Для работы с онтологиями написана программа на языке программирования Python, которая запускается в консоли и поддерживает выполнение запросов на получение экземпляров. Работы программы продемонстрирована на Рисунках 1.6.1–1.6.2.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.6.1 – Результат выполнения программы

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 1.6.2 – Результат выполнения программы

# Заключение

В ходе выполнения данной практической работы изучены теоретические основы системного анализа и использования онтологий в широком ряде задач, получены навыки построения онтологий и работы с ними, включая создание классов для описания выбранной предметной области, создание слотов в классах и создание экземпляров. С помощью инструменты работы с онтологиями Protégé выполнены запросы на получение объектов по различным запросам.

В качестве закрепления полученных знаний написана программа на языке программирования Python, способная работать с онтологией выбранной предметной области. В её функционал входит возможность писать запросы на получение экземпляров и связанных объектов.

# Список информационных источников

1. Сорокин А.Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие — М.: Московский технологический университет (МИРЭА), 2019. — 1 электрон. опт. диск (CD-ROM)
2. Лапшин В. А. Онтологии в компьютерных системах. — М.: Научный мир, 2010.
3. Карпенко А. П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко. – Москва: Издательство МГТУ им. Н. Э. Баумана, 2014. – 448 c.
4. Noy, Natalya F.; [McGuinness, Deborah L.](https://en.wikipedia.org/wiki/Deborah_McGuinness) (March 2001). ["Ontology Development 101: A Guide to Creating Your First Ontology"](https://web.archive.org/web/20100714172301/http:/www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html). Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, Stanford Medical Informatics Technical Report SMI-2001-0880
5. Добров Б. В., Иванов В.В., Лукашевич Н.В., Соловьев В.Д. [Онтологии и тезаурусы: модели, инструменты, приложения](http://www.intuit.ru/department/expert/ontoth/). — М.: Бином. Лаборатория знаний, 2009. — 173 с. — [ISBN 978-5-9963-0007-5](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785996300075).

# приложения

Приложение А — Код реализации онтологии на языке Python

### Приложение А

Код реализации онтологии на языке Python

Листинг А.1 – Реализация онтологии

from typing import Any

import random

import re

class OntologyObject():

    shown\_field = 'name'

    def \_\_str\_\_(self) -> str:

        if isinstance(self, Product):

            self.shown\_field = 'model'

        return getattr(self, self.shown\_field)

class Product(OntologyObject):

    instances = []

    def \_\_init\_\_(self, brand: str, model: str, color: str, price: int, size: float) -> None:

        self.brand = brand

        self.model = model

        self.color = color

        self.price = price

        self.size = size

        Product.instances.append(self)

class User(OntologyObject):

    instances = []

    def \_\_init\_\_(self, name: str, is\_active: bool):

        self.name = name

        self.is\_active = is\_active

        User.instances.append(self)

class Shoes(Product):

    instances = []

    def \_\_init\_\_(self, brand: str, model: str, color: str, price: int, size: float, heel\_height: str):

        super().\_\_init\_\_(brand, model, color, price, size)

        self.heel\_height = heel\_height

        Shoes.instances.append(self)

class Pants(Product):

    instances = []

    def \_\_init\_\_(self, brand: str, model: str, color: str, price: int, size: float, fit\_type: str, fits\_well: list[Shoes]):

        super().\_\_init\_\_(brand, model, color, price, size)

        self.fit\_type = fit\_type

        self.fits\_well = fits\_well

        Pants.instances.append(self)

class Shirt(Product):

    instances = []

Продолжение Листинга А.1

    def \_\_init\_\_(self, brand: str, model: str, color: str, price: int, size: float, buttons: int, fits\_style: list[Pants]):

        super().\_\_init\_\_(brand, model, color, price, size)

        self.buttons = buttons

        self.fits\_style = fits\_style

        Shirt.instances.append(self)

class Seller(User):

    instances = []

    def \_\_init\_\_(self, name: str, is\_active: bool, products: list[Product]):

        super().\_\_init\_\_(name, is\_active)

        self.products = products

        Seller.instances.append(self)

class Customer(User):

    instances = []

    def \_\_init\_\_(self, name: str, is\_active: bool, redemption: int,shopping\_list: list[Product]):

        super().\_\_init\_\_(name, is\_active)

        self.redemption = redemption

        self.shopping\_list = shopping\_list

        Customer.instances.append(self)

def find\_related\_objects\_by\_value(cls: OntologyObject.\_\_class\_\_,

                                  lookup\_field: str, value: Any):

    instances = cls.instances

    result = []

    for instance in instances:

        if isinstance(getattr(instance, lookup\_field), OntologyObject):

            if not isinstance(value, OntologyObject):

                value = find\_object\_by\_name(

                    getattr(instance, lookup\_field).\_\_class\_\_, value)

                if value is None:

                    return None

        if isinstance(getattr(instance, lookup\_field), bool):

            value = bool(value)

        elif isinstance(getattr(instance, lookup\_field), int):

            value = int(value)

        elif isinstance(getattr(instance, lookup\_field), float):

            value = float(value)

        if isinstance(getattr(instance, lookup\_field), list):

            for subfield in getattr(instance, lookup\_field):

                if subfield == value:

                    result.append((instance, type(instance).\_\_name\_\_))

        else:

            if getattr(instance, lookup\_field) == value:

                result.append((instance, type(instance).\_\_name\_\_))

    return result

def find\_object\_by\_name(cls: OntologyObject.\_\_class\_\_, name: str):

    instances = cls.instances

    for instance in instances:

        if instance.name == name:

            return instance

    return None

Продолжение Листинга А.1

def get\_class(class\_name: str):

    classes = {

        'shirt': Shirt,

        'pants': Pants,

        'shoes': Shoes,

        'seller': Seller,

        'customer': Customer

    }

    class\_name = class\_name.lower().strip()

    if class\_name in classes:

        return classes[class\_name]

    else:

        return None

def get\_random\_class\_instance(cls: OntologyObject.\_\_class\_\_):

    instances = cls.instances

    return random.choice(instances)

def get\_related\_class(obj: OntologyObject):

    classes = [Shirt, Pants, Shoes, Seller, Customer]

    class\_types = set()

    for cls in classes:

        cls\_instances = cls.instances

        for instance in cls\_instances:

            fields = [

                key for key in instance.\_\_dict\_\_.keys()

                if not re.match(r"\_\_\w\*\_\_", key)

            ]

            for field in fields:

                field\_value = getattr(instance, field)

                if field\_value == obj:

                    class\_types.add((cls, field))

                elif isinstance(field\_value, list):

                    for subfield\_value in field\_value:

                        if subfield\_value == obj:

                            class\_types.add((cls, field))

    return list(class\_types)

def main():

    shoes = [

        Shoes('Gucci', 'Ace Sneakers', 'black', 4390, 42., 'low'),

        Shoes('Manolo Blahnik', 'BB Pumps', 'black', 5290, 40., 'flat'),

        Shoes('Jimmy Choo', 'Lockett Petite Boots', 'pink', 4890, 42., 'high'),

        Shoes('Gucci', 'Princetown Loafers', 'brown', 5290, 44., 'flat')

        ]

    pants = [

        Pants('Tommy Hilfiger', 'Athletic Fit Pants', 'blue', 4990, 56., 'slim', [shoes[0], shoes[1]]),

        Pants('Tommy Hilfiger', 'Classic Fit Chinos', 'black', 3890, 58., 'regular', [shoes[0], shoes[3]]),

        Pants('Armani', 'Collezioni Trousers', 'black', 4590, 50., 'regular', [shoes[0], shoes[1], shoes[2], shoes[3]]),

        Pants('Calvin Klein', 'Modern Cotton', 'brown', 2990, 52., 'slim', [shoes[3]])

        ]

    shirts = [

        Shirt('Brooks Brothers', 'Non-Iron Dress', 'white', 2590, 54., 10, [pants[1], pants[2]]),

Продолжение Листинга А.1

        Shirt('Hugo Boss', 'The Boss Green Shirt', 'white', 2690, 56., 10, [pants[0], pants[1], pants[2], pants[3]]),

        Shirt('Hugo Boss', 'The Boss Selection Shirt', 'black', 3290, 54., 12, [pants[0], pants[3]]),

        Shirt('Thomas Pink', 'The Oxford', 'pink', 2290, 58., 12, [pants[3]])

        ]

    sellers = [

        Seller('best wear', True, [pants[0], pants[1], pants[2], pants[3], shoes[0], shoes[3]]),

        Seller('gentle', True, [shoes[0], shoes[2], shoes[3]]),

        Seller('your shop', False, [shirts[0], shirts[1], shirts[2], shoes[1]]),

        ]

    customers = [

        Customer('Bob', False, 54, [pants[0], pants[1]]),

        Customer('Carl', True, 79, [shoes[0], shoes[3]]),

        Customer('Liam', True, 98, [shoes[0], shirts[2], pants[1]]),

        Customer('Susan', True, 97, [pants[0], pants[2], shirts[2]]),

        ]

    while True:

        while True:

            class\_name = input('Введите класс получаемых объектов: ')

            cls: OntologyObject.\_\_class\_\_ | None = get\_class(class\_name)

            if cls is None:

                print('Такого класса не существует\n\n')

            else:

                break

        while True:

            instance = get\_random\_class\_instance(cls)

            available\_fields = [

                key for key in instance.\_\_dict\_\_.keys()

                if not re.match(r"\_\_\w\*\_\_", key)

            ]

            field = input(

                f'Введите требуемое поле ( {", ".join(available\_fields)} ): ')

            try:

                getattr(instance, field)

                break

            except Exception:

                print('Такого поля в классе не существует\n\n')

        value = input('Введите значение поля: ')

        res = find\_related\_objects\_by\_value(cls, field, value)

        while True:

            flag = False

            if res is None or len(res) == 0:

                print('Объекты по введённому запросу не найдены')

                while True:

                    \_type = input(

                        '1 - повторить ввод запроса\nq - завершить выполнение программы\n'

                    )

                    if \_type == '1':

                        break

                    elif \_type == 'q':

                        exit(0)

Продолжение Листинга А.1

            else:

                str\_objects = [

                    "\033[32m" + str(obj) + "\033[0m (\033[33m" +

                    str(obj\_type) + "\033[0m)" for obj, obj\_type in res

                ]

                print(f'Полученные объекты: {", ".join(str\_objects)}')

                while True:

                    \_type = input('1 - повторить ввод запроса\n'

                                  '2 - посмотреть связанные объекты\n'

                                  'q - завершить выполнение программы\n')

                    if \_type == '1':

                        flag = False

                        break

                    elif \_type == '2':

                        flag = True

                        break

                    elif \_type == 'q':

                        exit(0)

                if flag:

                    if len(res) == 1:

                        obj\_number = 1

                    else:

                        while True:

                            obj\_number = int(

                                input(

                                    f'Выберите номер нужного объекта (1-{len(res)}): '

                                ))

                            if obj\_number not in range(1, len(res) + 1):

                                print('Введён неправильный номер')

                            else:

                                break

                    instance = res[obj\_number - 1][0]

                    related\_classes = get\_related\_class(instance)

                    res = []

                    for rel\_class, field\_name in related\_classes:

                        res.extend(

                            find\_related\_objects\_by\_value(

                                rel\_class, field\_name, instance))

                else:

                    break

            if not flag:

                break

if \_\_name\_\_ == "\_\_main\_\_":

    main()