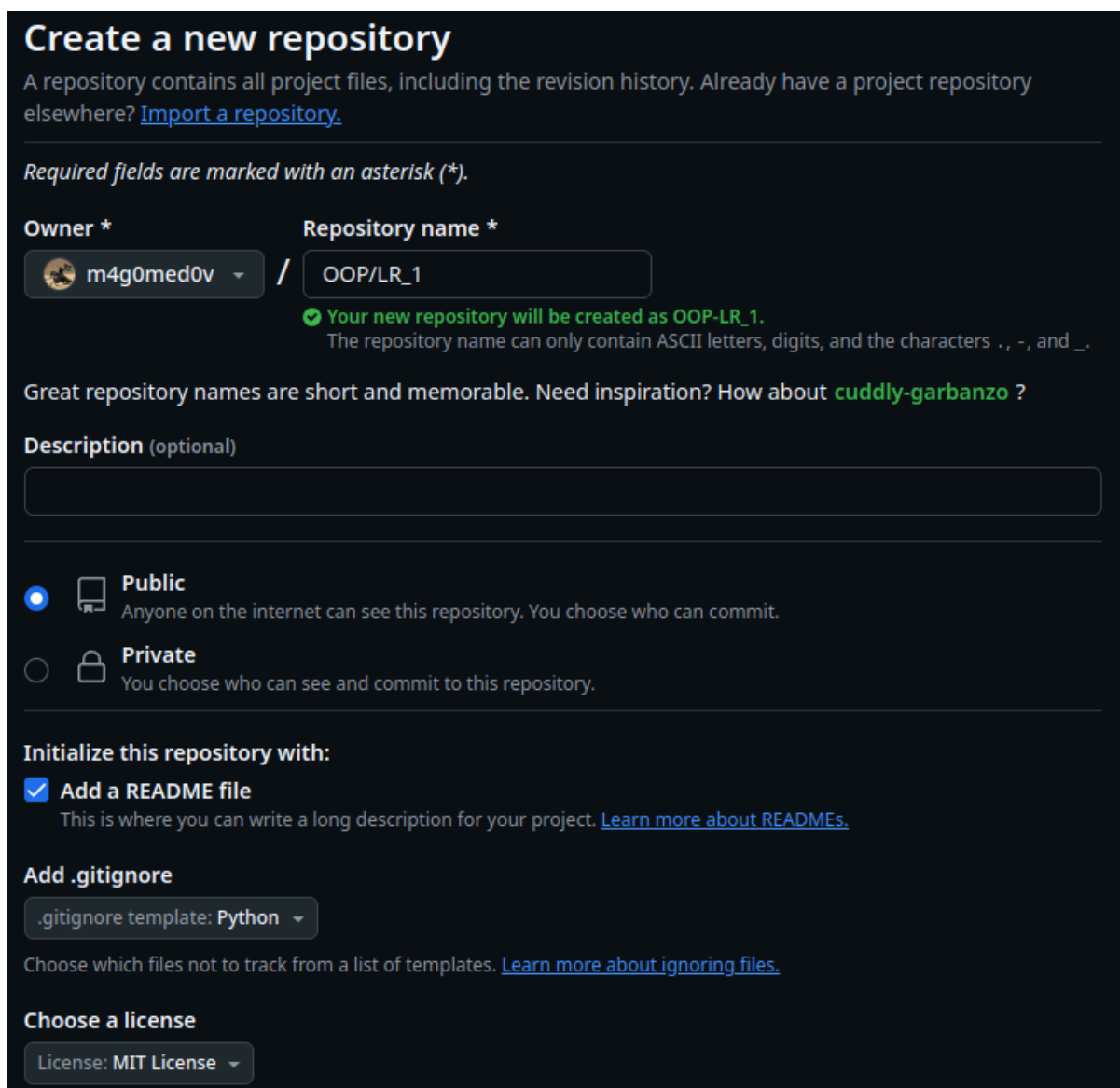


Тема: Элементы объектно-ориентированного
программирования в языке Python.

Цель работы: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Методика выполнения работы


1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk ().*


Owner *  m4g0med0v / **Repository name ***

✔ Your new repository will be created as OOP-LR_1.
The repository name can only contain ASCII letters, digits, and the characters ., -, and _.

Great repository names are short and memorable. Need inspiration? How about **cuddly-garbanzo** ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

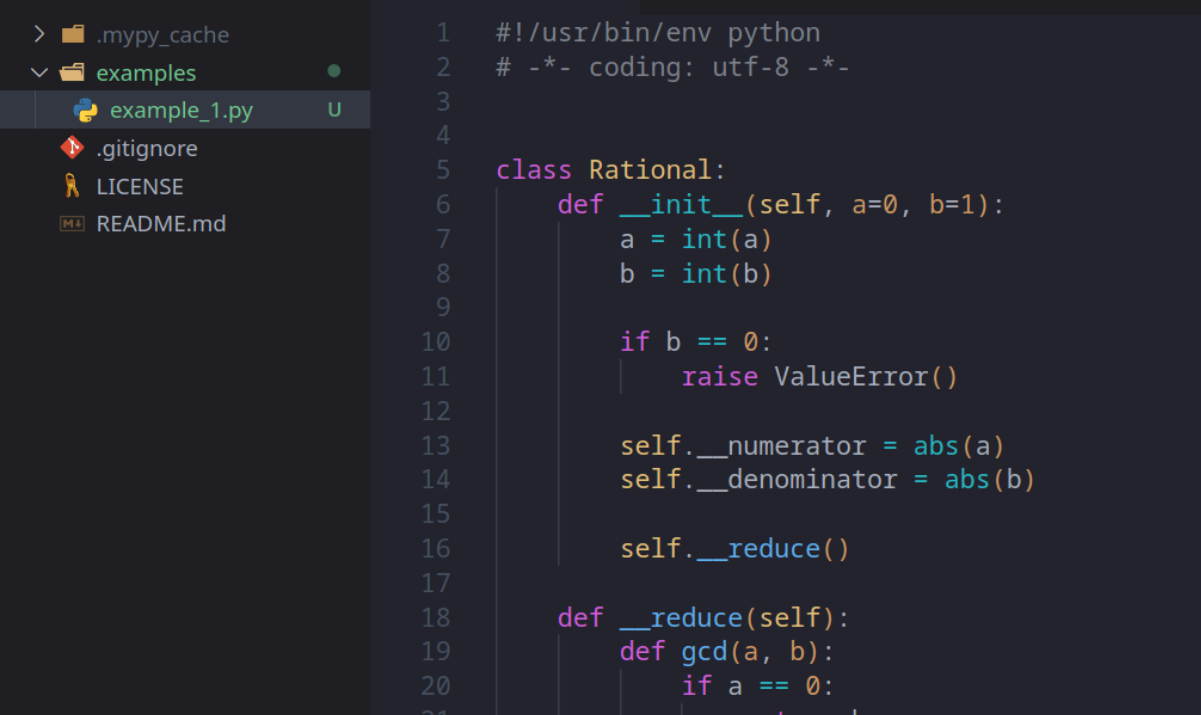
3. Выполните клонирование созданного репозитория.

```
~/Work/ncfu/object_oriented_programming/lab_work_4.1 git clone https://github.com/m4g0med0v/OOP-LR_1.git
Клонирование в «ООР-LR_1»...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Получение объектов: 100% (5/5), готово.
```

4. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
~/W/OOP-LR_1 main git branch develop
~/W/OOP-LR_1 main git branch
~/W/OOP-LR_1 main git checkout develop
Переключились на ветку «develop»
```

5. Проработайте примеры лабораторной работы.



```
> .mypy_cache
examples
example_1.py U
.gitignore
LICENSE
README.md

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4
5  class Rational:
6      def __init__(self, a=0, b=1):
7          a = int(a)
8          b = int(b)
9
10         if b == 0:
11             raise ValueError()
12
13         self.__numerator = abs(a)
14         self.__denominator = abs(b)
15
16         self.__reduce()
17
18     def __reduce(self):
19         def gcd(a, b):
20             if a == 0:
21                 return b
```

6. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

Задание 1.

Парой называется класс с двумя полями, которые обычно имеют имена first и second. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

- метод инициализации `__init__` ;метод должен контролировать значения аргументов на корректность;
- ввод с клавиатуры `read` ;
- вывод на экран `display` .

Реализовать внешнюю функцию с именем `make_тип()` , где `тип` — тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

Поле `first` — целое число, левая граница диапазона, включается в диапазон; поле `second` — целое число, правая граница диапазона, не включается в диапазон. Пара чисел представляет полуоткрытый интервал `[first, second)`. Реализовать метод `rangecheck()` — проверку заданного целого числа на принадлежность диапазону.

Решение

```

class Pair:
    def __init__(self, first: int, second: int):
        # Проверка на корректность диапазона
        if not (isinstance(first, int) and isinstance(second, int)):
            raise ValueError("Оба значения должны быть целыми числами.")
        if first >= second:
            raise ValueError("Значение first должно быть меньше значения second.")

        self.first = first
        self.second = second

    # Ввод данных с клавиатуры
    def read(self):
        try:
            self.first = int(input("Введите значение для first (целое число): "))
            self.second = int(input("Введите значение для second (целое число): "))
            if self.first >= self.second:
                raise ValueError("Значение first должно быть меньше значения second.")
        except ValueError as e:
            print(f"Ошибка ввода: {e}")
            return False
        return True

    # Вывод
    def display(self):

```

```

~/w/OOP-LR_1 > git develop ?2 > python3.12 src/individual_task_1.py
Пара чисел: [10, 20)
Число 15 принадлежит диапазону [10, 20)
Попробуйте ввести новую пару:
Введите значение для first (целое число): 12
Введите значение для second (целое число): 100
Пара чисел: [12, 100)
Число 15 принадлежит новому диапазону [12, 100)
~/work/OOP-LR_1 > git develop ?2 > python3.12 src/individual_task_1.py
Пара чисел: [10, 20)
Число 15 принадлежит диапазону [10, 20)
Попробуйте ввести новую пару:
Введите значение для first (целое число): 12
Введите значение для second (целое число): 1
Ошибка ввода: Значение first должно быть меньше значения second.
~/work/OOP-LR_1 > git develop ?2 > python3.12 src/individual_task_1.py
Пара чисел: [10, 20)
Число 15 принадлежит диапазону [10, 20)
Попробуйте ввести новую пару:
Введите значение для first (целое число): 12
Введите значение для second (целое число): a
Ошибка ввода: invalid literal for int() with base 10: 'a'

```

Задание 2.

Составить программу с использованием классов и объектов для решения задачи. Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы:

- метод инициализации `__init__` ;
- ввод с клавиатуры `read` ;

- вывод на экран `display` .

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

Создать класс `Time` для работы со временем в формате «час:минута:секунда». Класс должен включать в себя не менее четырех функций инициализации: числами, строкой (например, «23:59:59»), секундами и временем. Обязательными операциями являются: вычисление разницы между двумя моментами времени в секундах, сложение времени и заданного количества секунд, вычитание из времени заданного количества секунд, сравнение моментов времени, перевод в секунды, перевод в минуты (с округлением до целой минуты).

```
class Time:
    def initialize(self, hours=0, minutes=0, seconds=0):
        # Проверкой корректности времени
        if not (0 <= hours < 24 and 0 <= minutes < 60 and 0 <= seconds < 60):
            raise ValueError("Неверный формат времени: допустимые значения 0-23 для часов, 0-59 для минут, 0-59 для секунд")
        self.hours = hours
        self.minutes = minutes
        self.seconds = seconds

    # Инициализация объекта из строки вида "HH:MM:SS"
    def initialize_from_string(self, time_str):
        try:
            hours, minutes, seconds = map(int, time_str.split(':'))
            self.initialize(hours, minutes, seconds)
        except ValueError:
            raise ValueError("Неверный формат строки, ожидается 'HH:MM:SS'.")

    # Инициализация объекта с помощью секунд
    def initialize_from_seconds(self, total_seconds):
        if total_seconds < 0:
            raise ValueError("Количество секунд не может быть отрицательным.")

        hours = (total_seconds // 3600) % 24
        minutes = (total_seconds // 60) % 60
        seconds = total_seconds % 60
```

```

~/work/OOP-LR_1 git develop ?2 python3.12 src/individual_task_2.py
12:30:45
15:45:30
15:25:55
13:30:45
14:43:50
Разница между time1 и time2 в секундах: 4385
Время time1 в секундах: 48645
Время time1 в минутах: 811
time1 раньше, чем time2
Введите время в формате HH:MM:SS: 12:02:12
12:02:12

```

7. Зафиксируйте сделанные изменения в репозитории.

```

~/work/OOP-LR_1 git develop +2 git commit -m "added individual tasks"
[develop 9b010ee] added individual tasks
2 files changed, 235 insertions(+)
create mode 100644 src/individual_task_1.py
create mode 100644 src/individual_task_2.py

```

8. Выполните слияние ветки для разработки с веткой main / master.

```

~/work/OOP-LR_1 git develop git switch main
Переключились на ветку «main»
Эта ветка соответствует «origin/main».
~/work/OOP-LR_1 main git merge develop
Обновление cc58f37..9b010ee
Fast-forward
 examples/example_1.py | 132 +++++
 src/individual_task_1.py | 97 +++++
 src/individual_task_2.py | 138 +++++
 3 files changed, 367 insertions(+)
 create mode 100644 examples/example_1.py
 create mode 100644 src/individual_task_1.py
 create mode 100644 src/individual_task_2.py

```

9. Отправьте сделанные изменения на сервер GitHub.

```

~/work/OOP-LR_1 main +2 git push
Перечисление объектов: 10, готово.
Подсчет объектов: 100% (10/10), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (8/8), готово.
Запись объектов: 100% (9/9), 4.88 КиБ | 1.63 МиБ/с, готово.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/m4g0med0v/OOP-LR_1.git
 cc58f37..9b010ee main -> main

```

Контрольные вопросы

1. Как осуществляется объявление класса в языке Python?

Объявление класса осуществляется с использованием ключевого слова `class`, за которым следует имя класса, написанное по соглашению CapWords.

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибуты класса являются общими для всех экземпляров класса и задаются вне методов. Атрибуты экземпляра, в отличие от атрибутов класса, уникальны для каждого экземпляра и обычно определяются внутри метода `__init__` с использованием ключевого слова `self`.

3. Каково назначение методов класса?

Методы класса определяют поведение объектов, принадлежащих этому классу. Методы могут изменять состояние объекта, взаимодействовать с его атрибутами и выполнять действия на основе входных данных. Они создаются с использованием ключевого слова `def` и принимают первым параметром `self`, который представляет объект.

4. Для чего предназначен метод `__init__()` класса?

Метод `__init__()` служит конструктором класса и автоматически вызывается при создании нового экземпляра. Он используется для инициализации атрибутов объекта при его создании, задавая начальные значения экземпляру.

5. Каково назначение `self`?

`self` представляет собой ссылку на текущий экземпляр класса. Он используется для доступа к атрибутам и методам внутри класса, а также для различения атрибутов экземпляра от локальных переменных и параметров метода.

6. Как добавить атрибуты в класс?

Атрибуты класса добавляются, объявляя их внутри класса, но вне каких-либо методов. Атрибуты экземпляра добавляются внутри метода, чаще всего в `__init__`, с использованием `self`. Пример:

```
class MyClass:
    class_attr = 0 # Атрибут класса

    def __init__(self, value):
        self.instance_attr = value # Атрибут экземпляра
```

7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

В Python управление доступом к методам и атрибутам основывается на соглашениях, а не на жестких правилах, как в других языках (например, Java или C++). В Python используются следующие соглашения:

- Атрибуты и методы, доступные извне, имеют обычные имена без подчеркиваний.
- Для обозначения **приватных** (скрытых) атрибутов и методов используется одно подчеркивание перед именем (например, `_private_attr`). Это указывает, что этот элемент не предназначен для использования за пределами класса, хотя технически доступ к нему возможен.
- Если имя метода или атрибута начинается с двух подчеркиваний (например, `__private_method`), Python выполняет "**манглинг**" имен — автоматически изменяет имя, чтобы затруднить его доступ извне (атрибут становится, например, `_ClassName__private_method`). Однако даже такие атрибуты можно вызвать, зная правильное имя.

8. Каково назначение функции `isinstance` ?

Функция `isinstance()` в Python используется для проверки, является ли объект экземпляром определенного класса или его подкласса. Она принимает два аргумента: объект и класс, и возвращает `True`, если объект

является экземпляром указанного класса (или любого его подкласса), и False в противном случае. Это позволяет проверять типы объектов в процессе выполнения программы и, например, предотвращать ошибки при выполнении операций над объектами неверного типа.