Лабораторная работа №5-3.

Основные требования:

- каждая функция должна иметь docstring - множ-й комментарий (по типу что
  делает данная функция)
- написание функций должно быть компактным (не в 100 строчек кода)
- именования функций должны быть нормальными и четко отражать смысл самой
  функции
- код должен соответствовать стандарту языка Python (PEP).
- классы и объекты должны соответствовать стандартам языка программирования Python.

Общее задание к Л/Р:

Работа с (*argv, **kwargs).
Основная цель работы получить надпись на экране “Access Granted!” путем изменения передаваемых значений argv или kwargs. (возможно потребуется и то и то использовать)

Немного теории.
***args** и ****kwargs** - это соглашения в Python для передачи переменного числа аргументов в функцию.

***args** используется для передачи переменного числа позиционных аргументов. Когда вы видите ***args** в определении функции, это означает, что функция может принимать любое количество позиционных аргументов, и они будут доступны внутри функции как кортеж.

Простой пример:

```
def example_function(*args):
    for arg in args:
        print(arg)
example_function(1, 2, 3)  # Вывод: 1 2 3
```

****kwargs** используется для передачи переменного числа именованных (ключевых) аргументов. Когда вы видите ****kwargs** в определении функции, это означает, что функция может принимать любое количество именованных аргументов, и они будут доступны внутри функции как словарь.

Простой пример:

```
def example_function(**kwargs):
    for key, value in kwargs.items():
        print(key, value)
example_function(a=1, b=2, c=3)  # Вывод: a 1, b 2, c 3
```

Что нужно сделать?
В прошлой лабораторной вы работали с адресами, вам нужно в call_object записать определенный адрес (значение по адресу автоматически уже будет храниться в параметре value, **НЕ ЗАБУДЬТЕ УКАЗАТЬ В КОДЕ ПУТЬ К ФАЙЛУ memory_addresses.txt ИНАЧЕ РАБОТАТЬ НЕ БУДЕТ!!!**).

Далее нужно создать вывод (out) приравняв функции `__call__access__ ()` в которую предварительно необходимо что-то передать (вот это что-то вы и должны подобрать так чтобы выдавало при запуске надпись Access Granted!).

Как подбирать параметры в __call__access__()?
Подбор параметров осуществляется путем анализа кода класса Caller() или по блок схеме которую я прикрепил тут.

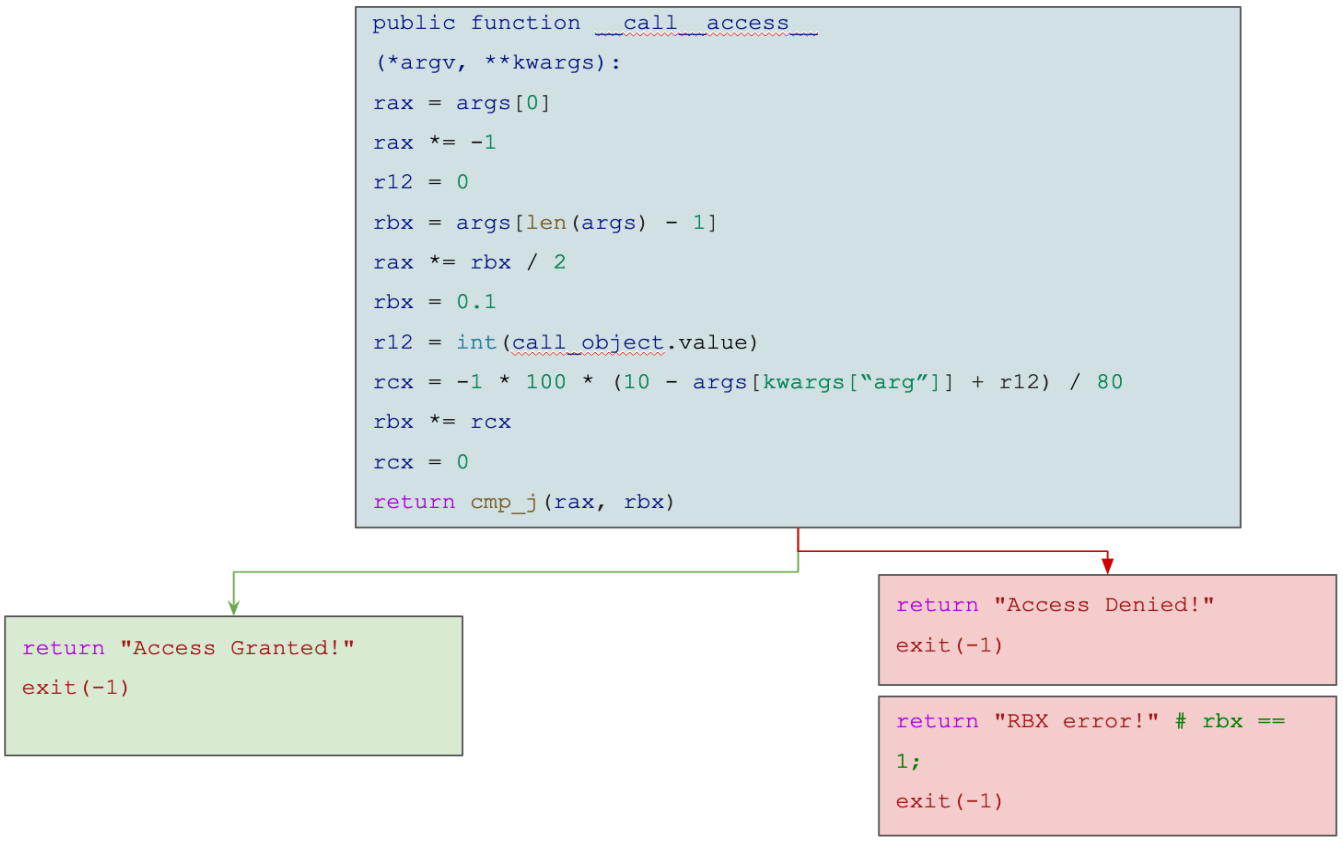
Итак **дан код** (Caller.py класс):

ВНИМАНИЕ!!! КОД МЕНЯТЬ СТРОГО ЗАПРЕЩЕНО (КРОМЕ УКАЗАНИЯ ПУТИ К ФАЙЛ .txt)

```
class Caller:
    def __init__(self, base_address: any) -> None:
        self.base_address = base_address
        self.value =
self.read_file("/your_path_to_file/memory_addresses.txt
")

    def read_file(self, file_path):
        result_dict = {}
        with open(file_path, 'r') as file:
            data = file.read()
            data = data.split("\n")
            for line in data:
                parts = line.strip().split("-")
                if len(parts) == 2:
                    key, value = parts
                    result_dict[key] = value
        if self.base_address in result_dict:
            print(result_dict[self.base_address])
            return result_dict[self.base_address]
        else:
            return f"No value found for key:
{self.base_address}"

    @staticmethod
    def cmp_j(rax, rbx) -> str:
        if rbx == -1:
            return "RBX error!"
        if rax > rbx:
```

```
            return "Access Denied!"
        else:
            return "Access Granted!"

    @staticmethod
    def __call__access__(call_object, *args, **kwargs)
-> str:
        rax = args[0]
        rax *= -1
        r12 = 0
        rbx = args[len(args) - 1]
        rax *= rbx / 2
        rbx = 0.1
        r12 = int(call_object.value)
        rcx = -1 * 100 * (10 - args[kwargs["arg"]] +
r12) / 80
        rbx *= rcx
        rcx = 0
        return call_object.cmp_j(rax, rbx)
```

Блок схема (зеленая (нужное) - если rax <= rbx, красная если rax > rbx или rbx= -1):
cmp_j для сравнения регистров **rax** и **rbx** используется.



Основной код (решения) будет находиться например в файле (main.py):

```
from Caller import Caller

call_object = Caller(...)
out = call_object.__call__access__(call_object, a1,
a2,..)
print(out)
```