Лабораторная работа №4.

Задание №1. Создайте класс **Laptop**, у которого есть:

- Конструктор __init__, принимающий 3 обязательных аргумента: бренд, модель и цену ноутбука.

На основании этих аргументов нужно для экземпляра создать атрибуты brand, model, price, а также атрибут laptop_name - строковое значение, следующего вида: "Стоимость <brand> <model> составляет <price> рублей".

- Укажите экземпляр lenovo для класса Laptop с параметрами: 'lenovo', 'z-570-dx', 25_000
- Затем вызовите атрибут laptop_name для экземпляр класса lenovo.

Задание №2. Создайте класс **SoccerPlayer**, у которого есть:

1. конструктор __init__, принимающий 2 аргумента: name, surname. Также во время инициализации необходимо создать 2 атрибута экземпляра: goals и assists - общее количество голов и передач игрока, изначально оба значения должны быть равны нулю;
2. метод score, который принимает количество голов, забитых игроком, по умолчанию данное значение равно единице. Метод должен увеличить общее количество забитых голов игрока на переданное значение;
3. метод make_assist, который принимает количество передач, сделанных игроком за матч, по умолчанию данное значение равно единице. Метод должен увеличить общее количество сделанных передач игроком на переданное значение;
4. метод statistics, который вывод на экран статистику игрока в виде: <Фамилия> <Имя> - голы: <goals>, передачи: <assists>

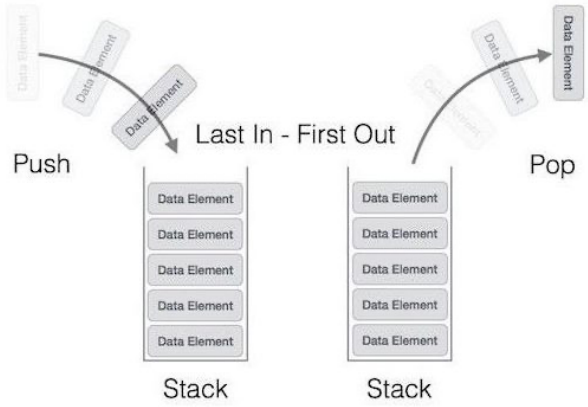
```
leo = SoccerPlayer('Leo', 'Messi')
leo.score(700)
leo.make_assist(500)
leo.statistics() # выводит "Messi Leo - голы: 700, передачи: 500"
kokorin = SoccerPlayer('Alex', 'Kokorin')
kokorin.score()
kokorin.statistics() # выводит "Kokorin Alex - голы: 1, передачи: 0"
```

Задание №3. Создайте класс **Person**, у которого есть:

- конструктор __init__, принимающий имя, фамилию и возраст. Их необходимо сохранить в атрибуты экземпляра first_name , last_name, age
- метод full_name, который возвращает строку в виде "<Фамилия> <Имя>"
- метод is_adult, который возвращает True, если человек достиг 18 лет и False в противном случае

```
p1 = Person('Jimi', 'Hendrix', 55)
print(p1.full_name())  # выводит "Hendrix Jimi"
print(p1.is_adult()) # выводит "True"
```

Задание №4. В этом задании вам предстоит реализовать свой стек(stack) - это упорядоченная коллекция элементов, организованная по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).



Ваша задача реализовать класс **Stack**, у которого есть:

- метод __init__создаёт новый пустой стек. Параметры данный метод не принимает. Создает атрибут экземпляра values, где будут в дальнейшем храниться элементы стека в виде списка (list), изначально при инициализации задайте значение атрибуту values равное пустому списку;
- метод push(item) добавляет новый элемент на вершину стека, метод ничего не возвращает.
- метод pop() удаляет верхний элемент из стека. Параметры не требуются, метод возвращает элемент. Стек изменяется. Если пытаемся удалить элемент из пустого списка, необходимо вывести сообщение "Empty Stack";
- метод peek() возвращает верхний элемент стека, но не удаляет его. Параметры не требуются, стек не модифицируется. Если элементов в стеке нет, распечатайте сообщение "Empty Stack", верните None после этого;
- метод is_empty() проверяет стек на пустоту. Параметры не требуются, возвращает булево значение.
- метод size() возвращает количество элементов в стеке. Параметры не требуются, тип результата - целое число.

```
s = Stack()
s.peek()  # распечатает 'Empty Stack'
print(s.is_empty())  # распечатает True
s.push('cat')  # кладем элемент 'cat' на вершину стека
s.push('dog')  # кладем элемент 'dog' на вершину стека
print(s.peek())  # распечатает 'dog'
s.push(True)  # кладем элемент True на вершину стека
print(s.size())  # распечатает 3
print(s.is_empty())  # распечатает False
s.push(777)  # кладем элемент 777 на вершину стека
print(s.pop())  # удаляем элемент 777 с вершины стека и печатаем его
print(s.pop())  # удаляем элемент True с вершины стека и печатаем его
print(s.size())  # распечатает 2
```

***Задание №5**. Создайте класс **Worker**, у которого есть:

- метод __init__, принимающий 4 аргумента: имя, зарплата, пол и паспорт. Необходимо сохранить их в следующих атрибутах: name, salary, gender и passport.
- метод get_info, которое распечатает информацию о сотруднике в следующем виде: «Worker {name}; passport-{passport}»

Пример использования класса Worker

```
bob = Worker('Bob Moore', 330000, 'M', '1635777202')
bob.get_info() # печатает "Worker Bob Moore; passport-1635777202"
```

Ниже имеется список кортежей persons, содержащий информацию о десяти работниках.

```
persons = [
   ('Allison Hill', 334053, 'M', '1635644202'),
   ('Megan Mcclain', 191161, 'F', '2101101595'),
   ('Brandon Hall', 731262, 'M', '6054749119'),
   ('Michelle Miles', 539898, 'M', '1355368461'),
   ('Donald Booth', 895667, 'M', '7736670978'),
   ('Gina Moore', 900581, 'F', '7018476624'),
   ('James Howard', 460663, 'F', '5461900982'),
   ('Monica Herrera', 496922, 'M', '2955495768'),
   ('Sandra Montgomery', 479201, 'M', '5111859731'),
   ('Amber Perez', 403445, 'M', '0602870126')
]
```

На основании этих данных необходимо создать 10 экземпляров класса Worker и добавить их в список worker_objects. Работников в списке следует разместить в том же порядке, в каком они встречаются в списке persons. В этом же порядке для каждого объекта в списке worker_objects вызовите метод get_info