ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

ТЕОРИЯ

ООП(Объектно-ориентированное программирование) – это способ организации кода, когда основу программы составляют объекты. Например, программа — это простой деревенский домик, части этого дома, такие как крыша, окно, дверь, будут являться объектами.

У каждого объекта всегда есть какое-нибудь свойство (у окна это цвет или его размер) или методы, которые объект выполняет(окно может открываться). КЛАСС - эскиз объекта программы, без которого нельзя создать объект.

```
classWindow:                                    # Класс, описывающий объект
color = "white"                                 # Задаём цвет окна
material = "wood"                               # Задаём материал для окна

defopen_window(self):
print("Окно открыто")                  # Метод, чтобы открыть окно


# Создаём объект нашего класса, с которым будем работать
my_window = Window();
print("Цвет окна: ", my_window.color,
"\nМатериал:", my_window.material);    # Мы можем получить свойства окна

my_window.open_window();                        # Или вызвать метод, который откроет окно
```

Что такоеself? Это слово помогает интерпретатору питона понятькакая переменная принадлежит текущему классу. Дополним предыдущий пример, добавив размеры окна и метод, который возвращает периметр оконной рамы.

```
classWindow:
color = "white"
material = "wood"

width = 0                                       # Задаём ещё два свойства: ширину и высоту
height = 0                                      # Пока оставляем параметры нулевыми

defopen_window(self):
print("Окнооткрыто")

        # Вметодемыбудемобращатьсякновым, измененнымпараметрамheight = 4,width = 5
defperimeter_of_window(self):
returnself.height * 2 + self.width * 2


my_window = Window();
my_window.width = 5                             # Устанавливаем параметры окна
my_window.height = 4
print(my_window.perimeter_of_window())     # Программавыведетзначение 18
```

Поговорим про создание объекта. Обращая внимание на предыдущие примеры можно заметить, что объект создаётся следующим образом: `my_window = Window()`.Сначала создаем переменную и ей присваиваем имя класса и круглые скобочки (по-умному: КОНСТРУКТОР)Конструктором в классе всегда является специальный метод `__init__(self)`

```
classWindow:
color = "white"
material = "wood"

# Конструктор
    # Если свойства объекта заданы в конструкторе, объявлять их отдельно не
обязательно
def__init__(self, width, height):
self.width = width       # Обратите внимание что self.width и width
self.height = height     # абсолютно разные переменные

defperimeter_of_window(self):
returnself.height * 2 + self.width * 2


my_window = Window(4, 5);
print(my_window.perimeter_of_window()) # Программа выведет значение 18
```

Отлично, у нас есть класс с конструктором, и мы уже можем создавать много разных окон, но что, если за работой мы случайно определим высоту окна как отрицательную величину, тогда работа всей программы будет не правильной. Чтобы такого не было мы организуем доступ к переменной через фильтры или публичные свойства, или, как их ещё называют геттеры и сеттеры (геттеры – для получения свойства вне класса, сеттеры – для изменения свойства) Возможность ограничить доступ к свойствам и методам объекта называется ИНКАПУЛЯЦИЕЙ. Кстати, инкапсуляцией так же называют объединение свойств и методов в пределах одного класса, для работы с ними.

```
classWindow:

def__init__(self, width, height):
self.__width = width# Чтобы "спрятать" свойство
self.__height = height# используем двойное подчеркивание

@property# Метод для возврата значения ширины окна
defwidth(self):
returnself.__width

@width.setter# Метод для изменения значения ширины окна
defwidth(self, width):
ifwidthinrange(1, 1000):     # Фильтр
self.__width = width
else:
print("вы пытаетесь установить неверный параметр")

defperimeter_of_window(self):
returnself.__height * 2 + self.__width * 2
```

```
my_window = Window(4, 5)
my_window.width = -9
print(my_window.perimeter_of_window())
# Программа выведет сообщение что была попытка установить неверный параметр, затем
выведет число 18

my_window.width = 10
print(my_window.perimeter_of_window())
# Программа выведет число 30
```

А теперь представим ситуацию что у нас появилось ещё одно окно, только другой формы (Например, это окно имеет форму равнобедренного треугольника). Так как программисты слишком ленивы, чтобы писать ещё один класс, а свойства и методы у нового окна почти совпадают со свойствами и методами старого, мы можем просто позаимствовать описание старого окна. Такой прием в ООП называется НАСЛЕДОВАНИЕМ.

```
# В скобочках указан класс, который будет унаследован
classNewWindow(Window):

definfo(self):
print("Это наше новое окно")


new_window = NewWindow(4, 5)
new_window.info()                    # Вызываем метод нашего нового класса
new_window.open_window()             # Вызываем метод старого класса
```

Мы описали новый класс на основе старого, но что делать с методом, который считает периметр оконной рамы. При вызове этого метода результат будет неверным. Чтобы не создавать новый метод мы просто можем переопределить(переписать) уже существующий. Этот способ обозначен в ООП как ПОЛИМОРФИЗМ.

```
classNewWindow(Window):

definfo(self):
print("Это наше новое окно")

# Переопределяем наш метод
defperimeter_of_window(self):
returnself.height * 2 + self.width


# Первый параметр - основание треугольника
# Второй параметр - длина его сторон
new_window = NewWindow(4, 5)
print(new_window.perimeter_of_window())
```

*Всё что выделено капсом надо понимать. Рекомендуем вам ещё почитать про ООП в следующих источниках:

https://metanit.com/python/tutorial/

https://habr.com/ru/post/463125/

https://habr.com/ru/post/87119/

*

ЗАДАНИЯ

1. Опишите класс ГЕОМЕТРИЧЕСКАЯ ФИГУРА
    a. Выделите основные свойства (то, что есть у каждой фигуры, например: цвет, координаты, форма)
    b. Сделайте свойства закрытыми и для каждого свойства пропишите геттеры и сеттеры для доступа.
    c. Создайте конструктор.
    d. Создайте в классе минимум три метод (Один из методов – вывод всей информации о фигуре)
    e. Создайте два разных объекта класса ГЕОМЕТРИЧЕСКАЯ ФИГУРА.
    f. Проверьте работу публичных свойств и методов на каждом объекте.

2. Создайте ещё два класса, которые будут наследовать класс ГЕОМЕТРИЧЕСКАЯ ФИГУРА (Это могут быть классы: ОКРУЖНОСТЬ, КВАДРАТ, ТРЕУГОЛЬНИК и т.д.)
    a. Создайте объекты новых классов и вызовите метод с выводом всей информации о фигуре на каждом объекте
    b. Добавьте в класс ГЕОМЕТРИЧЕСКАЯ ФИГУРА методы с подсчетом площади и периметра фигур. Переопределите эти методы в новых классах.
    c. Проверьте работу программы.

3. Создайте три новых класса: ЧЕЛОВЕК, УЧЕНИК и ПРЕПОДАВАТЕЛЬ. В классе ЧЕЛОВЕК обязательны свойства: имя, фамилия и возраст. Обязательно определить конструктор в каждом классе.
    a. Определите класс от которого остальные классы будут наследоваться.
    b. Создайте класс ГРУППА, у которого есть свойство имя_группы
    c. Создайте в классе ГРУППА массив из объектов класса УЧЕНИК, а так же добавьте объект класса ПРЕПОДАВАТЕЛЬ.
    d. Создайте в классе ГРУППА метод, который будет выводить список всех учеников.
    e. В программе создайте объект класса ГРУППА и вызовите метод который выведет список всех учеников группы.

4. Придумайте свой пример объяснения ООП. Представьте пример в виде небольшого кода с комментариями :)