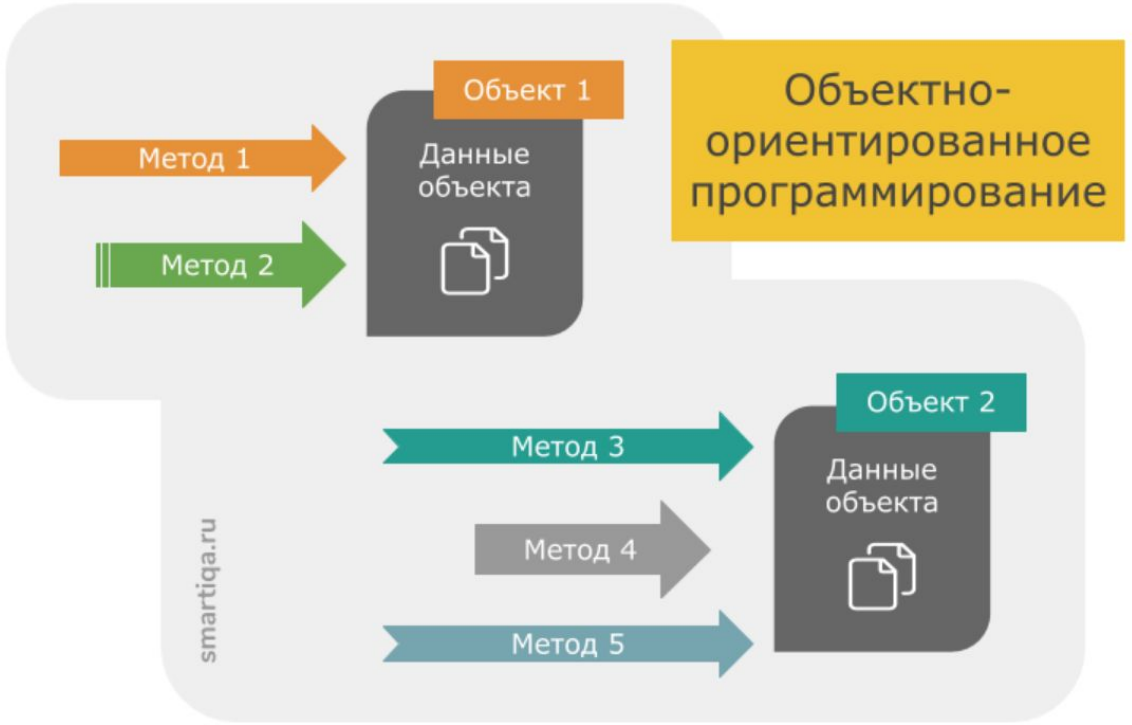
Моя IT Школа

сертифицированные IT курсы

ООП



Проверка домашнего задания

Два метода в классе, один принимает в себя либо строку, либо число.
Если я передаю строку, то смотрим:
если произведение гласных и согласных букв меньше или равно длине слова, выводить все гласные, иначе согласные;
если число то, произведение суммы чётных цифр на длину числа.
Длину строки и числа искать во втором методе.

Решение

Моя Школа

```
class TheExample:
    def __init__(self):
        self.h = 0
        self.d = 0
        self.g = 0
        self.gl = []
        self.sgl = []

    def func(self, a):
        if type(a) is str:
            for i in a:
                if i in "aeoiu":
                    self.h += 1
                    self.gl.append(i)
                else:
                    self.d += 1
                    self.sgl.append(i)
            print('Кол-во гласных', self.h)
            print('Кол-во согласных', self.d)
            print('Длина слова', self.func1(a))
            if (self.h * self.d) <= self.func1(a):
                print('Гласные: ', self.gl)
            else:
                print('Согласные: ', self.sgl)
```

```
        elif type(a) is int:
            for i in str(a):
                i = int(i)
                if (i % 2) == 0:
                    self.g += i
            print('Произведение: ', self.g * self.func1(a))

    def func1(self, a):
        return len(str(a))


example = TheExample()
c = input()
if c.isalpha():
    example.func(c)
elif c.isdigit():
    example.func(int(c))
```

Метод str

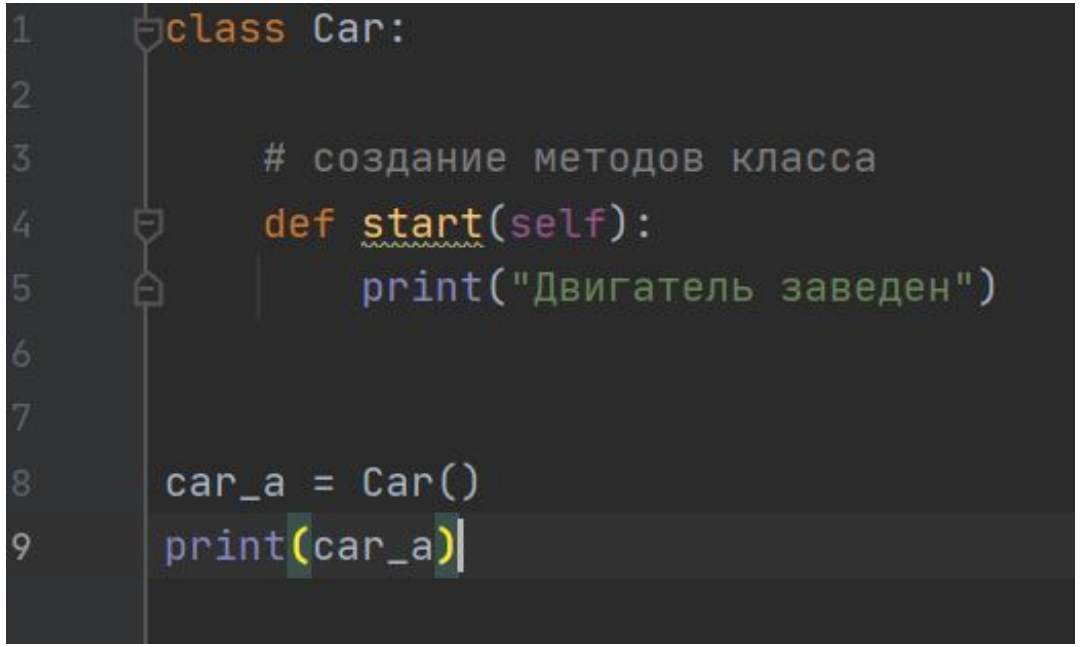
Моя IT Школа

Метод str

До этого момента мы выводили атрибуты при помощи метода print(). Посмотрим, что случится, если мы выведем объект класса.

Для этого нам нужно создать простой класс Car с одним методом и попытаться вывести объект класса в консоль

Результат покажет локацию памяти, где хранится наш объект

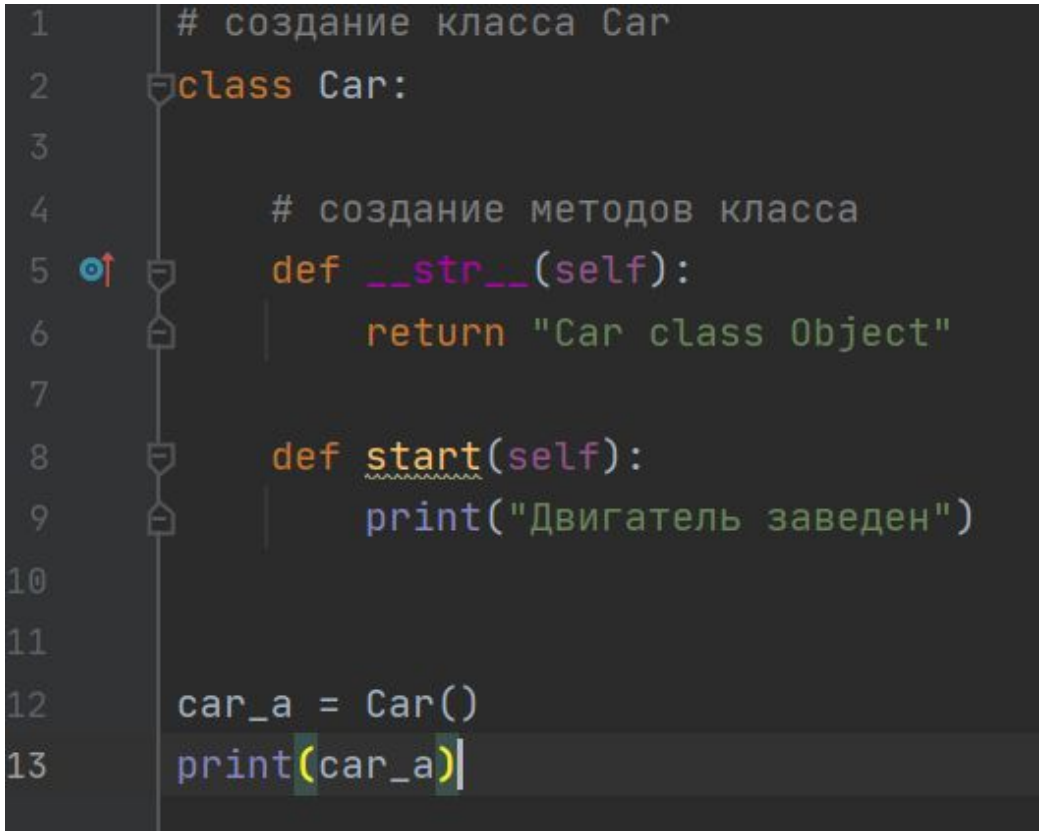


Метод str

Моя Школа

Каждый объект Python по умолчанию содержит метод __str__ . Когда вы используете объект в качестве строки, вызывается метод __str__ , который по умолчанию выводит локацию памяти объекта. Однако, вы также можете предоставить собственное определение метода __str__ .

В данной программе, мы переопределили метод __str__ , предоставив наше собственное определение метода. Теперь, если вы выведите объект car_a, вы увидите сообщение «Car class Object» в консоли. Это сообщение, которое мы внесли в наш пользовательский метод __str__

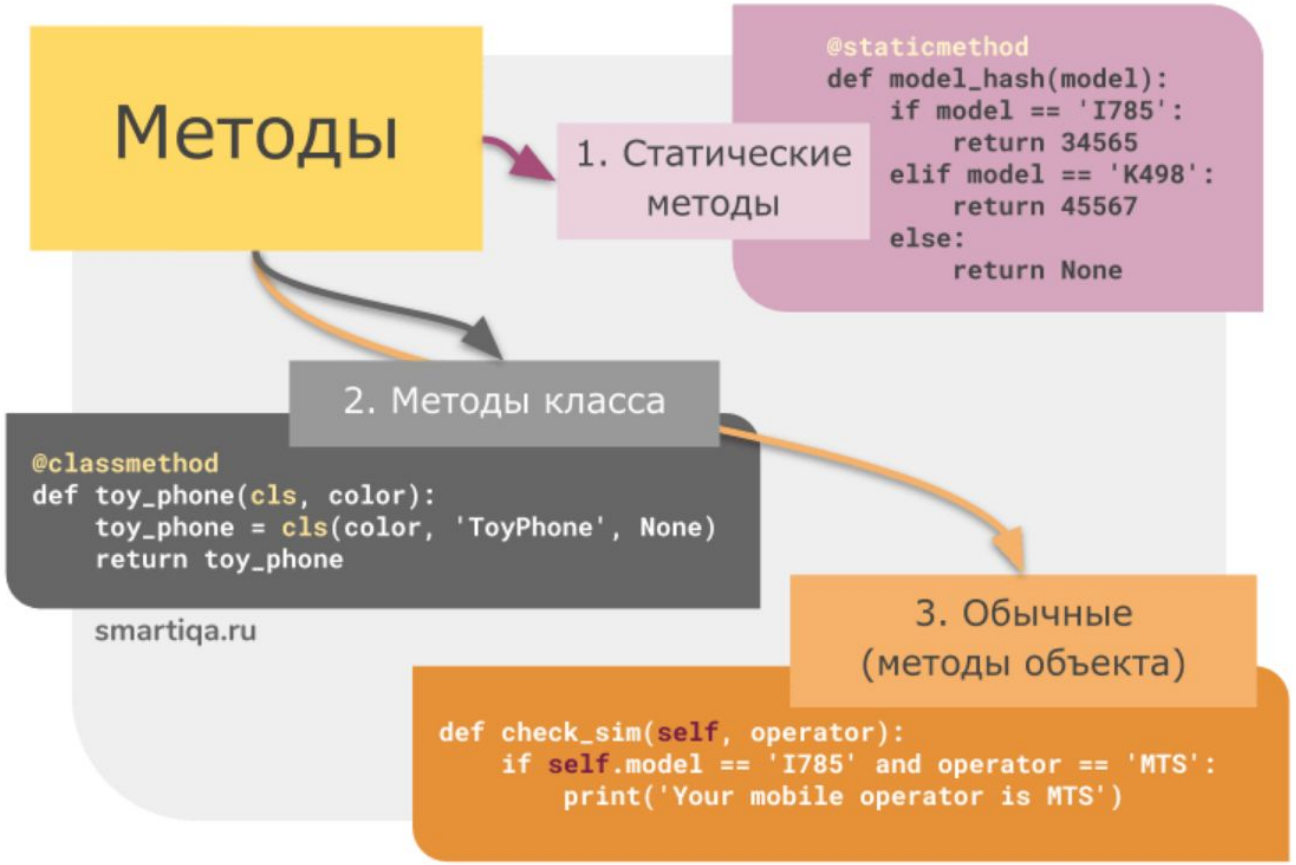


Методы (функции) класса

Моя Школа

Как вы уже знаете, функции внутри класса называются методами. Методы также бывают разными, а именно - их можно разделить на 3 группы:

1. Методы экземпляра класса (они же обычные методы)
2. Статические методы
3. Методы класса



Методы экземпляра класса

Моя IT Школа

Это группа методов, которые становятся доступны только после создания экземпляра класса, то есть чтобы вызвать такой метод, надо обратиться к экземпляру. Как следствие - первым параметром такого метода является слово **self**. И как мы уже обсудили выше, с помощью данного параметра в метод передается ссылка на объект класса, для которого он был вызван

```
class Phone:

    def __init__(self, color, model):
        self.color = color
        self.model = model

    # Обычный метод
    # Первый параметр метода - self
    def check_sim(self, mobile_operator):
        if self.model == 'I785' and mobile_operator == 'MTS':
            print('Your mobile operator is MTS')


my_phone = Phone('red', 'I785')
my_phone.check_sim('MTS')
```

Статические методы класса

Моя IT Школа

Статические методы — это обычные функции, которые помещены в класс для удобства и тем самым располагаются в области видимости этого класса. Чаще всего это какой-то вспомогательный код.

Чтобы создать статический метод в Python, необходимо воспользоваться специальным декоратором - **@staticmethod**

```
26      # Статический метод справочного характера
27      # Возвращает хэш по номеру модели
28      # self внутри метода отсутствует
29      @staticmethod
30      def model_hash(model):
31          if model == 'I785':
32              return 34565
33          elif model == 'K498':
34              return 45567
35          else:
36              return None
37
38  Phone.model_hash('I785')
39  my_phone = Phone('red', 'I785')
40  my_phone.check_sim('MTS')
```

Методы класса

Моя Школа

Методы класса являются чем-то средним между обычными методами (привязаны к объекту) и статическими методами (привязаны только к области видимости). Как легко догадаться из названия, такие методы тесно связаны с классом, в котором они определены.
Чтобы создать метод класса, необходимо воспользоваться соответствующим декоратором - **@classmethod**. При этом в качестве первого параметра такого метода передается служебное слово **cls**, которое в отличие от **self** является ссылкой на сам класс (а не на объект)

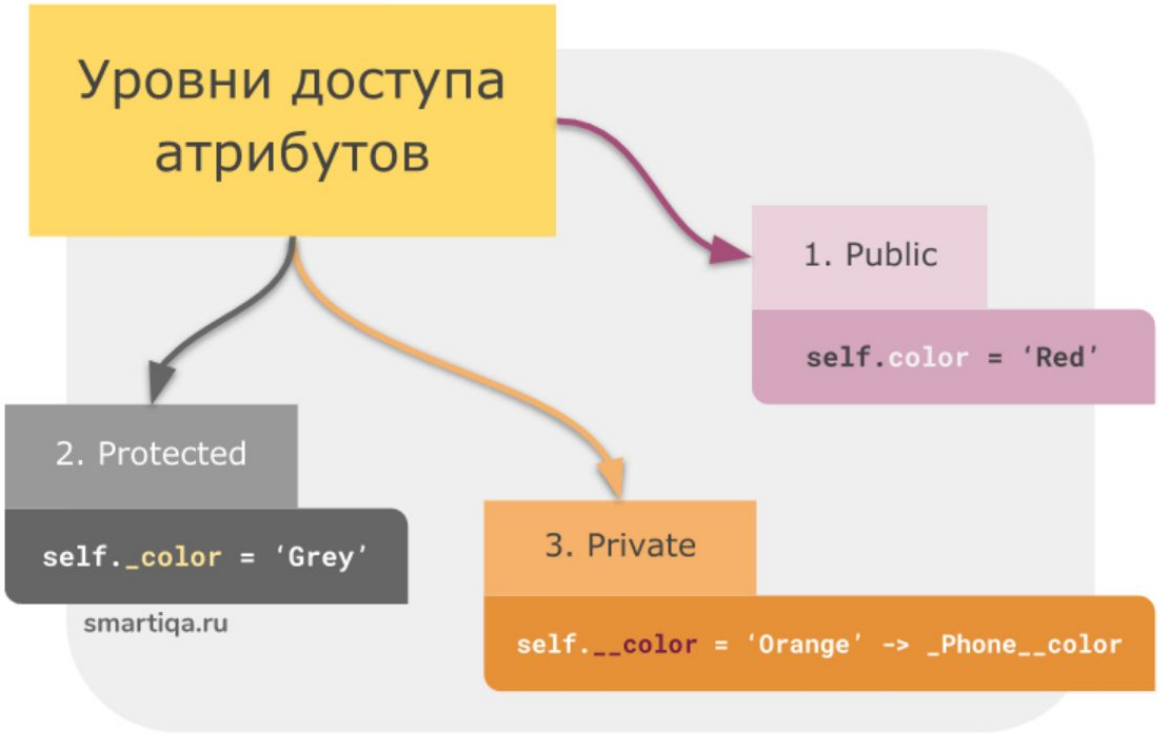
```
4       def __init__(self, os, color, model):
5           self.color = color
6           self.model = model
7           self.os = os
8
9       # Метод класса
10      # Принимает 1) ссылку на класс Phone и 2) цвет в качестве параметров
11      # Создает специфический объект класса Phone(особенность объекта в том, что это игрушечный телефон)
12      # При этом вызывается инициализатор класса Phone
13      # которому в качестве аргументов мы передаем цвет и модель,
14      # соответствующую созданию игрушечного телефона
15          @classmethod
16          def toy_phone(cls, color):
17              toy_phone = cls(color, 'ToyPhone', None)
18              return toy_phone
```

Уровни доступа атрибутов в Python

1. **Private.** Приватные члены класса недоступны извне - с ними можно работать только внутри класса.

2. **Public.** Публичные методы наоборот - открыты для работы снаружи и, как правило, объявляются публичными сразу по-умолчанию.

3. **Protected.** Доступ к защищенным ресурсам класса возможен только внутри этого класса и также внутри унаследованных от него классов (иными словами, внутри классов-потомков). Больше никто доступа к ним не имеет



Класс Human

1. Создайте класс Human.
2. Определите для него два статических поля: default_name и default_age.
3. Создайте метод __init__(), который помимо self принимает еще два параметра: name и age. Для этих параметров задайте значения по умолчанию, используя свойства default_name и default_age. В методе __init__() определите четыре свойства: Публичные - name и age. Приватные - money и house.
4. Реализуйте справочный метод info(), который будет выводить поля name, age, house и money.
5. Реализуйте справочный статический метод default_info(), который будет выводить статические поля default_name и default_age.
6. Реализуйте метод earn_money(), увеличивающий значение свойства money.

Тесты

1. Вызовите справочный метод default_info() для класса Human()
2. Создайте объект класса Human
3. Выведите справочную информацию о созданном объекте (вызовите метод info()).
4. Поправьте финансовое положение объекта - вызовите метод earn_money()
5. Посмотрите, как изменилось состояние объекта класса Human

```python
class Human:

    # Статические поля
    default_name = 'No name'
    default_age = 0

    def __init__(self, name=default_name, age=default_age):
        # Динамические поля
        # Публичные
        self.name = name
        self.age = age
        # Приватные
        self.__money = 0
        self.__house = None

    def info(self):
        print(f'Name: {self.name}')
        print(f'Age: {self.age}')
        print(f'Money: {self.__money}')
        print(f'House: {self.__house}')

```

```
23      @staticmethod
24      def default_info():
25          print(f'Default Name: {Human.default_name}')
26          print(f'Default Age: {Human.default_age}')
27
28      def earn_money(self, amount):
29          self.__money += amount
30          print(f'Earned {amount} money! Current value: {self.__money}')
31
32
33  if __name__ == '__main__':
34
35      Human.default_info()
36
37      alexander = Human('Sasha', 27)
38      alexander.info()
39
40      alexander.earn_money(5000)
41      alexander.earn_money(20000)
42      alexander.info()
```

Наследование в Python

Моя Школа

Как мы уже выяснили ранее , механизм наследования позволяет создать новый класс на основе уже существующего. При этом новый класс включает в себя как свойства и методы родительского класса, так и новые (собственные) атрибуты. Эти новые атрибуты и отличают свежесозданный класс от его родителя.

Для того, чтобы в Python создать новый класс с помощью механизма наследования, необходимо воспользоваться следующим синтаксисом:

class <имя_нового_класса>(<имя_родителя>):

В инициализаторе (метод **__init__**) наследуемого класса вызывается метод **super()**.

Главная задача этого метода - дать возможность наследнику **обратиться к родительскому классу**. В классе родителе **Phone** есть свой инициализатор, и когда в потомке **MobilePhone** мы так же создаем инициализатор (а он нам действительно нужен, так как внутри него мы хотим объявить новое свойство) - мы его **перегружаем**. Иными словами, мы заменяем родительский метод **__init__()** собственным одноименным методом. Это чревато тем, что родительский метод просто в принципе не будет вызван, и мы потеряем его функционал в классе наследнике. В конкретном случае, потеряем свойство **is_on**.

Чтобы такой потери не произошло, мы можем:

1. Внутри инициализатора класса-наследника вызвать инициализатор родителя (для этого вызываем метод **super().__init__()**)
2. А затем просто добавить новый функционал

```python
# Родительский класс
class Phone:

    # Инициализатор
    def __init__(self):
        self.is_on = False

    # Включаем телефон
    def turn_on(self):
        self.is_on = True

    # Если телефон включен, делаем звонок
    def call(self):
        if self.is_on:
            print('Making call...')
```

```python
# Унаследованный класс
class MobilePhone(Phone):

    # Добавляем новое свойство battery
    def __init__(self):
        super().__init__()
        self.battery = 0

# Заряжаем телефон на величину переданного значения
def charge(self, num):
    self.battery = num
    print(f'Charging battery up to ... {self.battery}%')

my_mobile_phone = MobilePhone()
dir(my_mobile_phone)
```

Множественное наследование Python

Моя Школа

В Python, родительский класс может иметь несколько дочерних, и, аналогично, дочерний класс может иметь несколько родительских классов.
Давайте рассмотрим первый сценарий.

```python
# создаем класс Vehicle
class Vehicle:
    def vehicle_method(self):
        print("Это родительский метод из класса Vehicle")


# создаем класс Car, который наследует Vehicle
class Car(Vehicle):
    def car_method(self):
        print("Это дочерний метод из класса Car")


# создаем класс Cycle, который наследует Vehicle
class Cycle(Vehicle):
    def cycleMethod(self):
        print("Это дочерний метод из класса Cycle")


car_a = Car()
car_a.vehicle_method()  # вызов метода родительского класса
car_b = Cycle()
car_b.vehicle_method()  # вызов метода родительского класса
```

В этой программе, родительский класс Vehicle наследуется двумя дочерними классами — Car и Cycle. Оба дочерних класса будут иметь доступ к vehicle_method() родительского класса.
Запустите программу, чтобы увидеть это лично

Множественное наследование Python

Моя IT Школа

Вы можете видеть, как родительский класс наследуется двумя дочерними классами. Таким же образом, дочерний класс может иметь несколько родительских.

```python
class Camera:
    def camera_method(self):
        print("Это родительский метод из класса Camera")


class Radio:
    def radio_method(self):
        print("Это родительский метод из класса Radio")


class CellPhone(Camera, Radio):
    def cell_phone_method(self):
        print("Это дочерний метод из класса CellPhone")


cell_phone_a = CellPhone()
cell_phone_a.camera_method()
cell_phone_a.radio_method()
```

В данной программе мы создали три класса: Camera, Radio, и CellPhone. Классы Camera и Radio наследуются классом CellPhone. Это значит, что класс CellPhone будет иметь доступ к методам классов Camera и Radio. Запустите программу, чтобы увидеть это лично

В предыдущем задании допишите:

Класс House

1. Создайте класс House

2. Создайте метод __init__() и определите внутри него два динамических свойства: _area и _price. 3. Свои начальные значения они получают из параметров метода __init__()

4. Создайте метод final_price(), который принимает в качестве параметра размер скидки и возвращает цену с учетом данной скидки.

Класс SmallHouse

1. Создайте класс SmallHouse, унаследовав его функционал от класса House

2. Внутри класса SmallHouse переопределите метод __init__() так, чтобы он создавал объект с площадью 40м2

Класс Human

1. Реализуйте приватный метод make_deal(), который будет отвечать за техническую реализацию покупки дома: уменьшать количество денег на счету и присваивать ссылку на только что купленный дом. В качестве аргументов данный метод принимает объект дома и его цену.

2. Реализуйте метод buy_house(), который будет проверять, что у человека достаточно денег для покупки, и совершать сделку. Если денег слишком мало - нужно вывести предупреждение в консоль. Параметры метода: ссылка на дом и размер скидки

Тесты

1. Создайте объект класса SmallHouse
2. Попробуйте купить созданный дом, убедитесь в получении предупреждения
3. Снова попробуйте купить дом, после поправки финансового положения

```
32      def buy_house(self, house, discount):
33          price = house.final_price(discount)
34          if self.__money >= price:
35              self.__make_deal(house, price)
36          else:
37              print('Not enough money!')
38
39      # Приватный метод
40      def __make_deal(self, house, price):
41          self.__money -= price
42          self.__house = house
```

```
45  class House:
46
47      def __init__(self, area, price):
48          self._area = area
49          self._price = price
50
51      def final_price(self, discount):
52          final_price = self._price * (100 - discount) / 100
53          print(f'Final price: {final_price}')
54          return final_price
55
56
57  class SmallHouse(House):
58
59      default_area = 40
60
61      def __init__(self, price):
62          super().__init__(SmallHouse.default_area, price)
63
```

```
65  if __name__ == '__main__':
66
67      Human.default_info()
68
69      alexander = Human('Sasha', 27)
70      alexander.info()
71
72      small_house = SmallHouse(8500)
73
74      alexander.buy_house(small_house, 5)
75
76      alexander.earn_money(5000)
77      alexander.buy_house(small_house, 5)
78
79      alexander.earn_money(20000)
80      alexander.buy_house(small_house, 5)
81      alexander.info()
```

Полиморфизм в Python

Моя Школа

Как вы уже знаете, полиморфизм позволяет перегружать одноименные методы родительского класса в классах-потомках. Что в свою очередь дает возможность использовать перегруженный метод в случаях, когда мы еще не знаем, для какого именно класса он будет вызван. Мы просто указываем имя метода, а объект класса, к которому он будет применен, определится по ходу выполнения программы

```
19      # Унаследованный класс
20      class MobilePhone(Phone):
21
22          def __init__(self):
23              super().__init__()
24              self.battery = 0
25
26          # Такой же метод, который выводит короткую сводку по классу MobilePhone
27          # Обратите внимание, что названия у методов совпадают - оба метода называются info()
28          # Однако их содержимое различается
29          def info(self):
30              print(f'Class name: {MobilePhone.__name__}')
31              print(f'If mobile phone is ON: {self.is_on}')
32              print(f'Battery level: {self.battery}')
33
```

Полиморфизм в Python

Моя Школа

```
35  # Демонстрационная функция
36
37  # Создаем список из классов
38  # В цикле перебираем список и для каждого элемента списка(а элемент - это класс)
39  # Создаем объект и вызываем метод info()
40  # Главная особенность: запись object.info() не дает информацию об объекте, для которого будет вызван метод info()
41  # Это может быть объект класса Phone, а может - объект класса MobilePhone
42  # И только в момент исполнения кода становится ясно, для какого именно объекта нужно вызывать метод info()
43  def show_polymorphism():
44      for item in [Phone, MobilePhone]:
45          print('-------')
46          object = item()
47          object.info()
```

Перегрузка метода

Моя IT Школа

Перегрузка метода относится к свойству метода вести себя по-разному, в зависимости от количества или типа параметров. Взглянем на очень простой пример перегрузки метода.

```
# создаем класс Car
class Car:
    def start(self, a, b=None):
        if b is not None:
            print(a + b)
        else:
            print(a)


car_a = Car()
car_a.start(10)
```

В данной программе, если метод start() вызывается передачей одного аргумента, параметр будет выведен на экран. Однако, если мы передадим 2 аргумента методу start(), он внесет оба аргумента и выведет результат суммы.

Инкапсуляция

Моя Школа

Чтобы предоставить контролируемый доступ к данным класса в Python, используются модификаторы доступа и свойства. Мы уже ознакомились с тем, как действуют модификаторы доступа. Сейчас посмотрим, как действуют свойства.

Предположим, что нам нужно убедиться в том, что модель автомобиля должна датироваться между 2000 и 2018 годом. Если пользователь пытается ввести значение меньше 2000 для модели автомобиля, значение автоматически установится как 2000, и если было введено значение выше 2018, оно должно установиться на 2018. Если значение находится между 2000 и 2018 — оно остается неизменным.

```python
# создаем класс Car
class Car:

    # создаем конструктор класса Car
    def __init__(self, model):
        # Инициализация свойств.
        self.model = model

    # создаем свойство модели.
    @property
    def model(self):
        return self.__model

    # Сеттер для создания свойств.
    @model.setter
    def model(self, model):
        if model < 2000:
            self.__model = 2000
        elif model > 2018:
            self.__model = 2018
        else:
            self.__model = model
```

Свойство имеет три части.
Вам нужно определить атрибут, который является моделью в скрипте выше.
Затем, вам нужно определить свойство атрибута, используя декоратор @property.
Наконец, вам нужно создать установщик свойства, который является дескриптором @model.setter в примере выше.

Теперь, если вы попробуете ввести значение выше 2018 в атрибуте модели, вы увидите, что значение установлено на 2018.

```python
    def getCarModel(self):
        return "Год выпуска модели " + str(self.model)


carA = Car(2088)
print(carA.getCarModel())
```

Класс Alphabet

1. Создайте класс Alphabet
2. Создайте метод __init__(), внутри которого будут определены два динамических свойства:
1) lang - язык и 2) letters - список букв. Начальные значения свойств берутся из входных параметров метода.
3. Создайте метод print(), который выведет в консоль буквы алфавита
4. Создайте метод letters_num(), который вернет количество букв в алфавите

Класс EngAlphabet

1. Создайте класс EngAlphabet путем наследования от класса Alphabet
2. Создайте метод __init__(), внутри которого будет вызываться родительский метод __init__(). В качестве параметров ему будут передаваться обозначение языка(например, 'En') и строка, состоящая из всех букв алфавита(можно воспользоваться свойством ascii_uppercase из модуля string).
3. Добавьте приватное статическое свойство __letters_num, которое будет хранить количество букв в алфавите.
4. Создайте метод is_en_letter(), который будет принимать букву в качестве параметра и определять, относится ли эта буква к английскому алфавиту.
5. Переопределите метод letters_num() - пусть в текущем классе классе он будет возвращать значение свойства __letters_num.
6. Создайте статический метод example(), который будет возвращать пример текста на английском языке.

Тесты

1. Создайте объект класса EngAlphabet
2. Напечатайте буквы алфавита для этого объекта
3. Выведите количество букв в алфавите
4. Проверьте, относится ли буква F к английскому алфавиту
5. Проверьте, относится ли буква Щ к английскому алфавиту
6. Выведите пример текста на английском языке

```python
import string

# Алфавит
class Alphabet:

    def __init__(self, language, letters_str):
        self.lang = language
        self.letters = list(letters_str)

    # Печатаем все буквы алфавита
    def print(self):
        print(self.letters)

    # Возвращаем количество букв в алфавите
    def letters_num(self):
        len(self.letters)


```

```python
# Английский алфавит
class EngAlphabet(Alphabet):

    __letters_num = 26

    def __init__(self):
        super().__init__('En', string.ascii_uppercase)

    # Проверяем, относится ли буква к английскому алфавиту
    def is_en_letter(self, letter):
        if letter.upper() in self.letters:
            return True
        return False

    # Возвращаем количество букв в алфавите
    def letters_num(self):
        return EngAlphabet.__letters_num

    # Печатаем пример текста на английском языке
    @staticmethod
    def example():
        print("English Example:\nDon't judge a book by it's cover.")
```

```
44  if __name__ == '__main__':
45      eng_alphabet = EngAlphabet()
46      eng_alphabet.print()
47      print(eng_alphabet.letters_num())
48      print(eng_alphabet.is_en_letter('F'))
49      print(eng_alphabet.is_en_letter('Щ'))
50      EngAlphabet.example()
51
```

Домашнее задание

Класс Tomato:

1. Создайте класс Tomato
2. Создайте статическое свойство states, которое будет содержать все стадии созревания помидора
3. Создайте метод __init__(), внутри которого будут определены два динамических protected свойства: 1) _index - передается параметром и 2) _state - принимает первое значение из словаря states
4. Создайте метод grow(), который будет переводить томат на следующую стадию созревания
5. Создайте метод is_ripe(), который будет проверять, что томат созрел (достиг последней стадии созревания)

Класс TomatoBush

1. Создайте класс TomatoBush
2. Определите метод __init__(), который будет принимать в качестве параметра количество томатов и на его основе будет создавать список объектов класса Tomato. Данный список будет храниться внутри динамического свойства tomatoes.
3. Создайте метод grow_all(), который будет переводить все объекты из списка томатов на следующий этап созревания
4. Создайте метод all_are_ripe(), который будет возвращать **True**, если все томаты из списка стали спелыми
5. Создайте метод give_away_all(), который будет чистить список томатов после сбора урожая

Класс Gardener

1. Создайте класс Gardener
2. Создайте метод __init__(), внутри которого будут определены два динамических свойства: 1) name - передается параметром, является публичным и 2) _plant - принимает объект класса Tomato, является protected
3. Создайте метод work(), который заставляет садовника работать, что позволяет растению становиться более зрелым
4. Создайте метод harvest(), который проверяет, все ли плоды созрели. Если все - садовник собирает урожай. Если нет - метод печатает предупреждение.
5. Создайте статический метод knowledge_base(), который выведет в консоль справку по садоводству.

Тесты

1. Вызовите справку по садоводству
2. Создайте объекты классов TomatoBush и Gardener
3. Используя объект класса Gardener, поухаживайте за кустом с помидорами
4. Попробуйте собрать урожай
5. Если томаты еще не дозрели, продолжайте ухаживать за ними
6. Соберите урожай