АКАДЕМИЯ АЙТИ

День 21

academyit.ru

ООП

Функционалы, функциональное программирование, функции map, filter, генераторы, декораторы

Турашова Анна Николаевна
Преподаватель
anna1turashova@gmail.com
Telegram: @anna1tur

Проверка домашнего задания

Задание 1.

Преобразуйте класс Time, написанный на вебинаре, в класс DateTime. Он должен хранить в себе информацию о дне, часах и минутах. Дни могут быть неограниченно большими, а также отрицательными.
Измените методы для сложения и вычитания так, чтобы они корректно складывали время и дни.

Дополнительно: Добавьте месяцы и года, учтите високосный год, добавьте как можно больше методов перегрузки.

Пример – класс Time (время)

```
import functools
# на основе < и == делает все сравнения
@functools.total_ordering
class Time:
    """ Время в часах и минутах """
    def __init__(self, h=0, m=0):
        total = h * 60 + m
        self.h = total // 60 % 24
        self.m = total % 60

    def __str__(self):
        return f"{self.h:02}:{self.m:02}"

    def __add__(self, other):  # +
        return Time(self.h + other.h, self.m + other.m)

    def __mul__(self, other):  # t * x
        if isinstance(other, int):
            return Time(self.h * other, self.m * other)

    def __rmul__(self, other):  # x * t
        return self * other

    def __sub__(self, other):  # a - b
        return self + (other * -1)

    def __neg__(self):  # -b
        return self * -1

    def __int__(self):
        return self.h * 60 + self.m

    def __lt__(self, other):  # <
        return int(self) < int(other)

    def __eq__(self, other):  # ==
        return int(self) == int(other)
```

ООП

Полиморфизм

- Объектно-ориентированный подход позволяет писать код, который будет правильно работать с экземплярами различных классов, которые возможно даже еще не созданы.
- Достаточно, чтобы у разных классов были одинаковые по названию и смыслу методы, которые могут при этом работать по-разному.
- Это называется **полиморфизм.**

Пример – геометрические фигуры

```
from math import pi

class Circle:
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return pi * self.radius ** 2
    def perimeter(self):
        return 2 * pi * self.radius
```

Пример – геометрические фигуры

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)
```

Полиморфная функция

```
def print_shape_info(shape):
    print(f"Area = {shape.area()}, perimeter = {
shape.perimeter()}.")


square =  Rectangle(10, 10)
print_shape_info(square)
# Area = 100, perimeter = 40.

circle = Circle(10)
print_shape_info(circle)
# Area = 314.1592653589793, perimeter = 62.83185
307179586.
```

Утиная типизация

- Код использует то, что в Python принята утиная типизация. Название происходит от шутки «Если нечто выглядит как утка, плавает как утка и крякает как утка, это, вероятно, утка и есть».
- В программах на Python это означает, что если какой-то объект поддерживает все требуемые от него операции, с ним и будут работать с помощью этих операций, не заботясь о том, какого он на самом деле типа.
- Утиная типизация позволяет написать функцию, которая будет работать со всеми экземплярами любых классов — даже еще не существующих.

Проверка типа объекта

- Функция isinstance функция принимает два параметра: isinstance(object, type)
- Первый параметр представляет объект, а второй — тип, на принадлежность к которому выполняется проверка. Если объект представляет указанный тип, функция возвращает True.

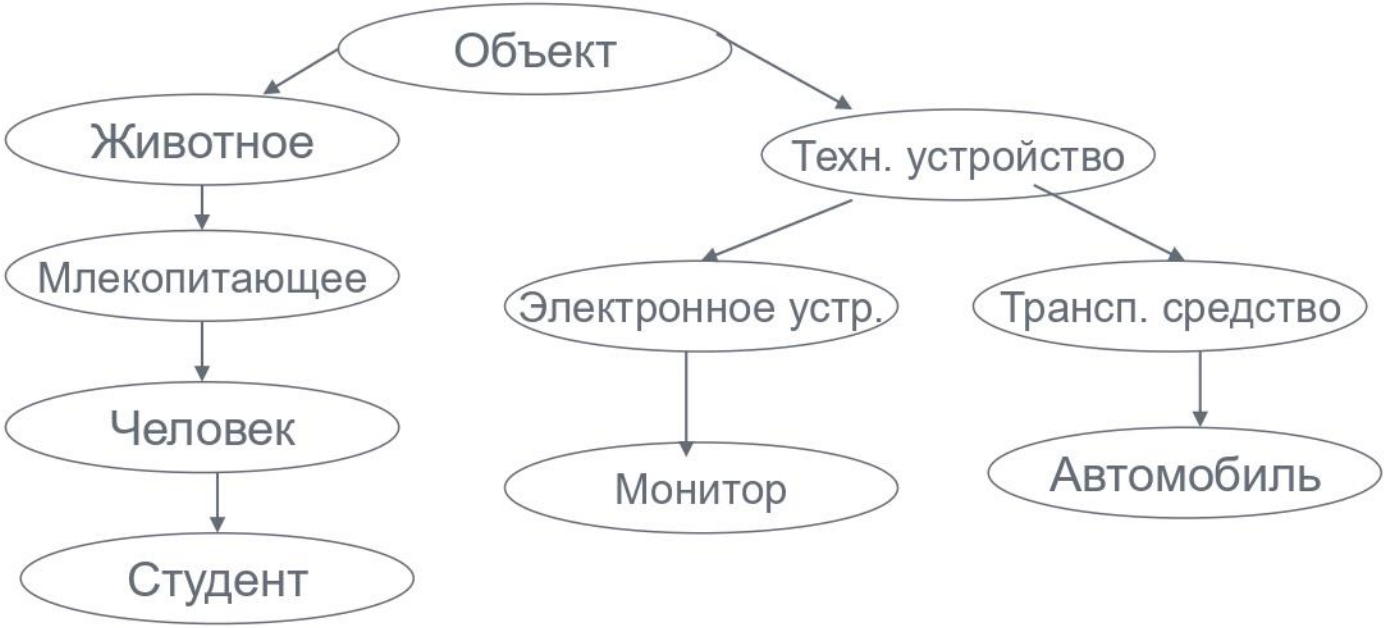
Пример: список фигур.

Наследование классов

- Python поддерживает наследование классов, что позволяет создавать новые классы с расширенным и/или измененным функционалом базового класса.
- Новый класс, созданный на основе базового класса - называется производный класс или просто подкласс.
- Подкласс наследует атрибуты и методы из родительского класса. Он так же может переопределять методы родительского класса.
- Если подкласс не определяет свой конструктор __init__, то он наследует конструктор родительского класса.

Преимущества наследования

- Наследование описывает отношения, которые напоминают отношения реального мира.
- Механизм наследования предоставляет возможность повторного использования, которая позволяет пользователю добавлять дополнительные функции в производный класс, не изменяя его.
- Если класс Y наследуется от класса X, то автоматически все подклассы Y будут наследовать от класса X.



Пример: геометрические фигуры

```
class Shape:
    def describe(self):
        print(f"Класс: {self.__class__.__name__}")
class Circle(Shape):
    def __init__(self, radius):
        self.r = radius
    def area(self):
        return pi * self.r ** 2

class Rectangle(Shape):
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def area(self):
        return self.a * self.b

shape = Shape()
shape.describe()
circle = Circle(1)
circle.describe()
rectangle = Rectangle(1, 2)
rectangle.describe()
```

Пример: расширение метода

```
class Square(Rectangle):
    def __init__(self, size):
        print('Создаем квадрат')
        super().__init__(size, size)


sq = Square(2)
print(sq.area())
print(sq.perimeter())
print(sq.a)
```

Вызов методов наследников в базовом классе

```
class Shape:
    def describe(self):
        # Добавим еще и название класса
        print(f"Класс: {self.__class__.__name__}\n"
              f"Периметр: {self.perimeter()}\n"
              f"Площадь: {self.area()}")

    def area(self):
        return None

    def perimeter(self):
        return None
```

Переопределение методов в производном классе

```
class Rectangle(Shape):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def area(self):
        return self.a * self.b
```

Абстрактные методы

Абстрактным называется класс, который содержит один и более **абстрактных методов**. **Абстрактные классы** - это классы, которые предназначены для наследования, но избегают реализации конкретных методов, оставляя только сигнатуры методов, которые должны реализовывать подклассы. Создавать объекты таких классов нельзя!

Абстрактным называется объявленный, но не реализованный метод.

В Python отсутствует встроенная поддержка абстрактных классов, для этой цели используется модуль abc (Abstract Base Class)

```
from abc import ABC, abstractmethod
```

Пример: абстрактный класс Фигура

```
class Shape(ABC):
    def describe(self):
        # Добавим еще и название класса
        print(f"Класс: {self.__class__.__name__}\n"
              f"Периметр: {self.perimeter()}\n"
              f"Площадь: {self.area()}")

    @abstractmethod
    def area(self):
        return None

    @abstractmethod
    def perimeter(self):
        return None
```

Пример: животные

```
class Animal:
      pass

class Cat(Animal):
      pass


class Dog(Animal):
      pass
```

Множественное наследование

Python предоставляет возможность наследоваться сразу от нескольких классов.
Такой механизм называется **множественное наследование**, и он позволяет вызывать в производном классе методы разны базовых классов.

```
class Base1:
    def tic(self):
        print("tic")

class Base2:
    def tac(self):
        print("tac")

class Derived(Base1, Base2):
    pass

d = Derived()
d.tic()  # метод, наследованный от Base1
d.tac()  # метод, наследованный от Base2
```

Задачи

Задача 3.

Евгения создала класс KgToPounds с параметром kg, куда передается определенное количество килограмм, а с помощью метода to_pounds() они переводятся в фунты. Чтобы закрыть доступ к переменной “kg” она реализовала методы set_kg() - для задания нового значения килограммов, get_kg() - для вывода текущего значения кг. Из-за этого возникло неудобство: нам нужно теперь использовать эти 2 метода для задания и вывода значений. Помогите ей переделать класс с использованием функции property() и свойств-декораторов. Код приведен ниже.

```
class KgToPounds:

    def __init__(self, kg):
        self.__kg = kg

    def to_pounds(self):
        return self.__kg * 2.205

    def set_kg(self, new_kg):
        if isinstance(new_kg, (int, float)):
            self.__kg = new_kg
        else:
            raise ValueError('Килограммы задаются только числами')

    def get_kg(self):
        return self.__kg
```

Задача 4.

Николай – оригинальный человек.
Он решил создать класс Nikola, принимающий при инициализации 2 параметра: имя и возраст.

Но на этом он не успокоился.
Не важно, какое имя передаст пользователь при создании экземпляра, оно всегда будет содержать “Николая”.

В частности - если пользователя на самом деле зовут Николаем, то с именем ничего не произойдет, а если его зовут, например, Максим, то оно преобразуется в “Я не Максим, а Николай”.

Более того, никаких других атрибутов и методов у экземпляра не может быть добавлено, даже если кто-то и вздумает так поступить (т.е. если некий пользователь решит прибавить к экземпляру свойство «отчество» или метод «приветствие», то ничего у такого хитреца не получится).

Задача 5.

Строки в Питоне сравниваются на основании значений символов.
Т.е. если мы захотим выяснить, что больше: «Apple» или «Яблоко», – то «Яблоко» окажется бОльшим.
А все потому, что английская буква «A» имеет значение 65 (берется из таблицы кодировки), а русская буква «Я» – 1071 (с помощью функции ord() это можно выяснить).
Такое положение дел не устроило Яну.
Она считает, что строки нужно сравнивать по количеству входящих в них символов.
Для этого девушка создала класс RealString и реализовала озвученный инструментарий. Сравнивать между собой можно как объекты класса, так и обычные строки с экземплярами класса RealString.
К слову, Яне понадобилось только 3 метода внутри класса (включая __init__()) для воплощения задуманного.

Задача 6.

Есть Алфавит, характеристиками которого являются:
1. Язык
2. Список букв

Для Алфавита можно:
1. Напечатать все буквы алфавита
2. Посчитать количество букв

Так же есть Английский алфавит, который обладает следующими свойствами:
1. Язык
2. Список букв
3. Количество букв

Для Английского алфавита можно:
1. Посчитать количество букв
2. Определить, относится ли буква к английскому алфавиту
3. Получить пример текста на английском языке

Задача 6.

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

Задача 6.

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.

Домашнее задание

1. Создайте класс Tree – дерево. Оно должно иметь год посадки, текущий возраст дерева и метод, который увеличивает возраст дерева на 1 год.

2. Наследуясь от класса Tree, создайте класс Apple_tree – яблоня. Она должна иметь максимальный возраст яблони и метод info, который покажет всю информацию о яблоне.
А также список с информацией о количестве выросших яблок за прошедшие года, каждый новый элемент списка – новое число выросших яблок за один год.
Переопределите метод, увеличивающий возраст, чтобы он после каждого прошедшего года записывал яблоки в список урожайности.

АПЛАНА
ГРУППА КОМПАНИЙ

Входит в ГК Аплана

АКАДЕМИЯ АЙТИ

Основана в 1995 г.

E-learning и очное обучение

Направления обучения:

- Информационные технологии
- Информационная безопасность
- ИТ-менеджмент и управление проектами
- Разработка и тестирование ПО
- Гос. и муниципальное управление

Филиалы:

Санкт-Петербург, Казань, Уфа, Челябинск, Хабаровск, Красноярск, Тюмень, Нижний Новгород, Краснодар, Волгоград, Ростов-на-Дону

Головной офис в Москве

Разработка программного обеспечения и информационных систем

Ежегодные награды Microsoft, Huawei, Cisco и другие

Программы по импортозамещению

Ресурсы более 400 высококлассных экспертов и преподавателей

Сеть региональных учебных центров по всей России

Крупные заказчики

СБЕР

РЖД

Билайн

TATNEFT

НОРНИКЕЛЬ

100+
сотрудников

АКАДЕМИЯ АЙТИ

Спасибо за внимание!

Центральный офис:
Москва, Варшавское шоссе 47, корп. 4, 7 этаж
Тел: +7 (495) 150-96-00
academy@it.ru
academyit.ru