АКАДЕМИЯ АЙТИ

День 22

academyit.ru

Классы и объекты.
Введение в объектно-ориентированное программирование (ООП).
Класс и экземпляр класса.
Данные экземпляра, методы экземпляра и свойства экземпляра.
Создание собственного класса.
Инкапсуляция.
Атрибуты класса.
Чтение и изменение атрибута.

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

Проверка домашнего задания

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.

ООП: повторение

Инкапсуляция

- В ООП значения атрибутов обычно скрываются от других объектов и доступны только с использованием методов
- В этом состоит принцип **инкапсуляции**
- В Python для скрытия атрибута или метода перед его именем ставят два подчеркивания __

Полиморфизм

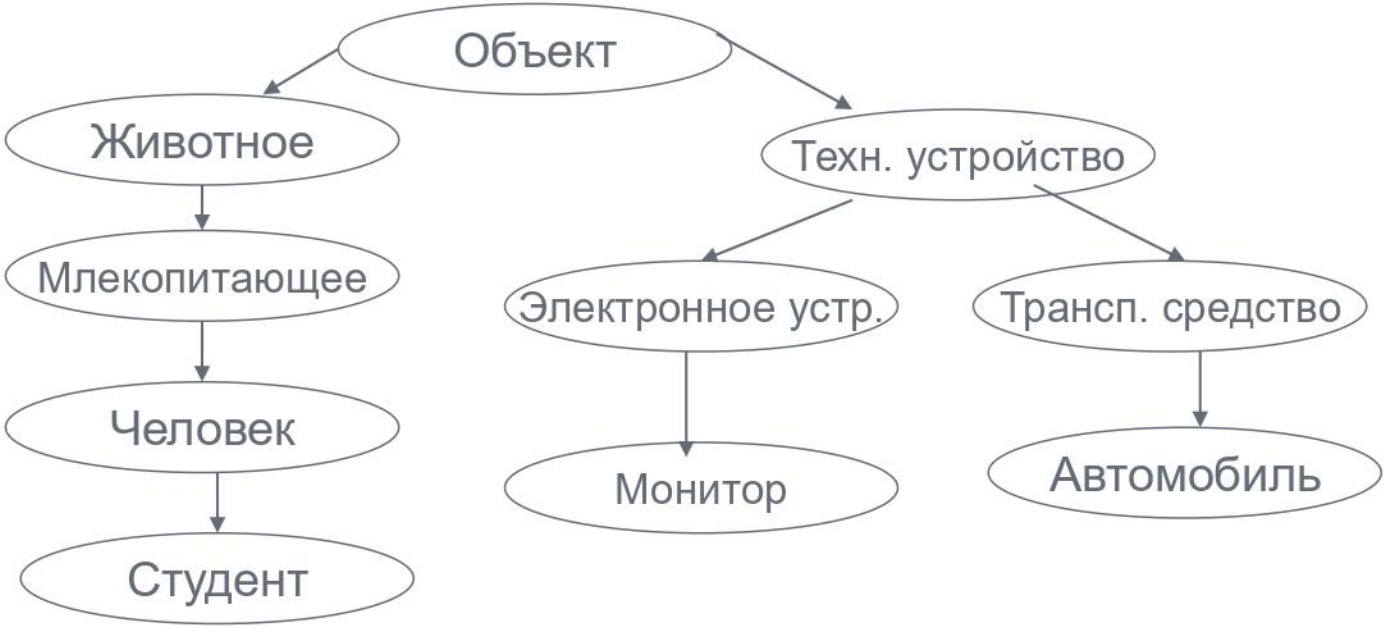
- Объектно-ориентированный подход позволяет писать код, который будет правильно работать с экземплярами различных классов, которые возможно даже еще не созданы.
- Достаточно, чтобы у разных классов были одинаковые по названию и смыслу методы, которые могут при этом работать по-разному.
- Это называется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

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.



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 7.

Есть Помидор со следующими характеристиками:

1. Индекс
2. Стадия зрелости(стадии: Отсутствует, Цветение, Зеленый, Красный)

Помидор может:

1. Расти (переходить на следующую стадию созревания)
2. Предоставлять информацию о своей зрелости

Есть Куст с помидорами, который:

1. Содержит список томатов, которые на ней растут

И может:

1. Расти вместе с томатами
2. Предоставлять информацию о зрелости всех томатов
3. Предоставлять урожай

И также есть Садовник, который имеет:

1. Имя
2. Растение, за которым он ухаживает

И может:

1. Ухаживать за растением
2. Собирать с него урожай

Задача 7.

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)

Задача 7.

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

Задача 7.

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.

Задача 8.

Есть Человек, характеристиками которого являются:
1. Имя
2. Возраст
3. Наличие денег
4. Наличие собственного жилья

Человек может:
1. Предоставить информацию о себе
2. Заработать деньги
3. Купить дом

Также же есть Дом, к свойствам которого относятся:
1. Площадь
2. Стоимость

Для Дома можно:
1. Применить скидку на покупку

Также есть Небольшой Типовой Дом, обязательной площадью 40м2.

Задача 8.

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 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.
7. Реализуйте метод earn_money(), увеличивающий значение свойства money.
8. 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

Задача 8.

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

ООП

Свойства объектов

- Свойства (property) – это динамические атрибуты
- При обращении к свойствам они выглядят как атрибуты: для чтения используется синтаксис <имя объекта>.<имя свойства>, для установки значения мы присваиваем значение

  <имя объекта>.<имя свойства> = <значение>
- На самом деле при обращении к свойству вызывается некоторый метод чтения (getter), возвращающий значение, а при присвоении значения свойству – другой метод, сохраняющий это значение, который называется метод записи (setter)

Описание свойств в Python

- Метод чтения помечается декоратором @property

- Метод записи помечается декоратором

@<имя метода чтения>.setter

- Альтернативный вариант описания:

<имя> = property(

fget=<метод чтения>,

fset=<метод записи>,

doc=<документация>)

Пример

```
class Person:
    def __init__(self, name='', surname=''):
        self.name = name
        self.surname = surname

    @property
    def full_name(self):
        return self.name + ' ' + self.surname

    @full_name.setter
    def full_name(self, new):
        if len(new.split(' ')) != 2:
            raise ValueError('Bad fullname')
        self.name, self.surname = new.split(' ')

    def set_age(self, new):
        if isinstance(new, int) and 0 <= new <= 120:
            self._age = new
        else:
            raise ValueError('Bad age')

    age = property(
        fget=lambda self: self._age,
        fset=set_age,
        doc='age - возраст')
```

Классы данных (Python 3.7+)

- Часто в программах используются классы, которые сохраняют некоторые атрибуты

```
class RegularBook:
    def __init__(self, title, author):
            self.title = title
            self.author = author
```

- Такие классы можно описать как классы данных, указав только имена атрибутов и аннотации типов

```
from dataclasses import dataclass
@dataclass
class Book:
    title: str
    author: str
```

Классы данных (Python 3.7+)

- В результате автоматически будут созданы методы __init__, __str__, __eq__
- Можно указать значения по умолчанию

```
@dataclass
class Book:
    author: str = 'Пушкин'
```

- Если при создании класса указать в декораторе order = True, то будут созданы методы сравнения
- Автоматический метод __init__ будет вызывать метод __post_init__, если он определен
- Если при объявления поля указать в качестве его типа InitVar, его значение будет передано как параметр метода __post_init__

Классы данных (Python 3.7+)

- В класс данных можно добавлять методы
- Классы данных могут наследоваться с добавлением новых атрибутов
- Классы можно сделать неизменяемыми, для этого, применяя декоратор dataclass, установите для параметра frozen значение True
- Можно получить атрибуты объекта в кортеже или словаре с помощью astuple и asdict из dataclasses
- Функция field позволяет указывать параметры работы с отдельными переменными.

dataclasses.field(*, default=MISSING, default_factory=MISSING, repr=True, hash=None, init=True, compare=True, metadata=None)

Модуль collections

- Модуль collections в Python включает некоторые контейнеры – классы, объекты которых могут сохранять множество значений
- Counter (счетчик) – словарь для подсчета количества неизменяемых объектов, фактически представляет собой множество с повторениями
- deque – двусторонняя очередь
- OrderedDict – словарь, запоминающий порядок добавления ключей (в версии 3.7+ так ведет себя любой словарь)
- namedtuple – кортеж, где каждый элемент имеет имя

Применение Counter

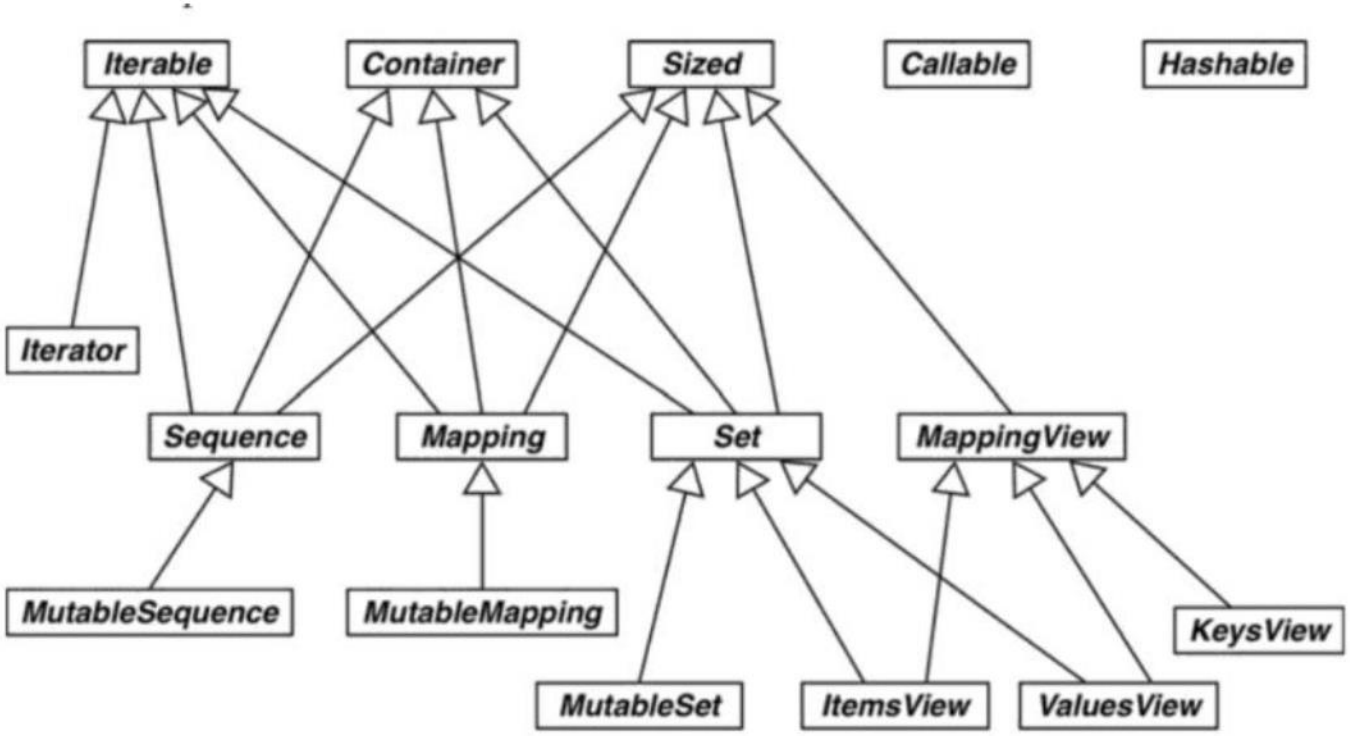
- Подсчет, сколько раз встретилось каждая буква в тексте

```
text = "..."
from collections import Counter
c = Counter()
for letter in text:
    c[letter] += 1
print(c)
print(list(c.elements()))
print(c.most_common(5))

x = Counter(a=3, b=1)
y = Counter(a=1, b=2)
print(x + y)
print(x - y)
print(x & y)
```

Модуль collections.abc

Этот модуль предоставляет абстрактные базовые классы, которые можно использовать для проверки того, предоставляет ли класс определенный интерфейс; например, является ли он последовательностью или множеством



Миксины (примеси)

Примеси добавляют функциональность в классы, которые от них наследуются.

Например, класс, унаследованный от Sequence, требует определения только двух методов __getitem__ , __len__, после чего начинают работать все методы для последовательностей, например index, count, перебор циклом for и т.п.

Пример – множество на основе списка

```
class ListBasedSet(collections.abc.Set):
    def __init__(self, iterable):
        self.elements = []
        for value in iterable:
            if value not in self.elements:
                self.elements.append(value)

    def __iter__(self):
        return iter(self.elements)

    def __contains__(self, value):
        return value in self.elements

    def __len__(self):
        return len(self.elements)

s1 = ListBasedSet('abcdef')
s2 = ListBasedSet('defghi')
print(s1 & s2)
```

Домашнее задание

Стандартный модуль datetime

Содержит 5 классов:

datetime.datetime — для представления одновременно даты и времени.
datetime.date — для представления только даты. Содержит методы, аналогичные методам datetime для работы с датами.
datetime.time — для представления только времени. Содержит методы, аналогичные методам datetime для работы со временем.
datetime.timedelta — для представления разницы во времени, используется для проведения арифметических действий над датами и временем
datetime.tzinfo — для представления информации о временной зоне (часовой пояс).

Полная документация к модулю доступна на сайте **python** (на английском языке).

https://docs.python.org/3.8/library/datetime.html

Задание 1.

Попробуйте создать объекты класса datetime.datetime:

```
import datetime
dt = datetime.datetime(2019, 1, 16)
dt2 = datetime.datetime(2019, 1, 16, minute=11)
dt3 = datetime.datetime.today()
print(dt.replace(year=2013,hour=12))
```

Дополнительно почитайте про этот модуль в интернете.
Попробуйте также сложить дни:

```
now = datetime.datetime.now()
two_days = datetime.timedelta(2)
in_two_days = now + two_days
```

Задание 2.

Почитайте различные статьи про ООП.
Например, эту:

https://python.ivan-shamaev.ru/classes-python-3-methods-oop-examples/

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