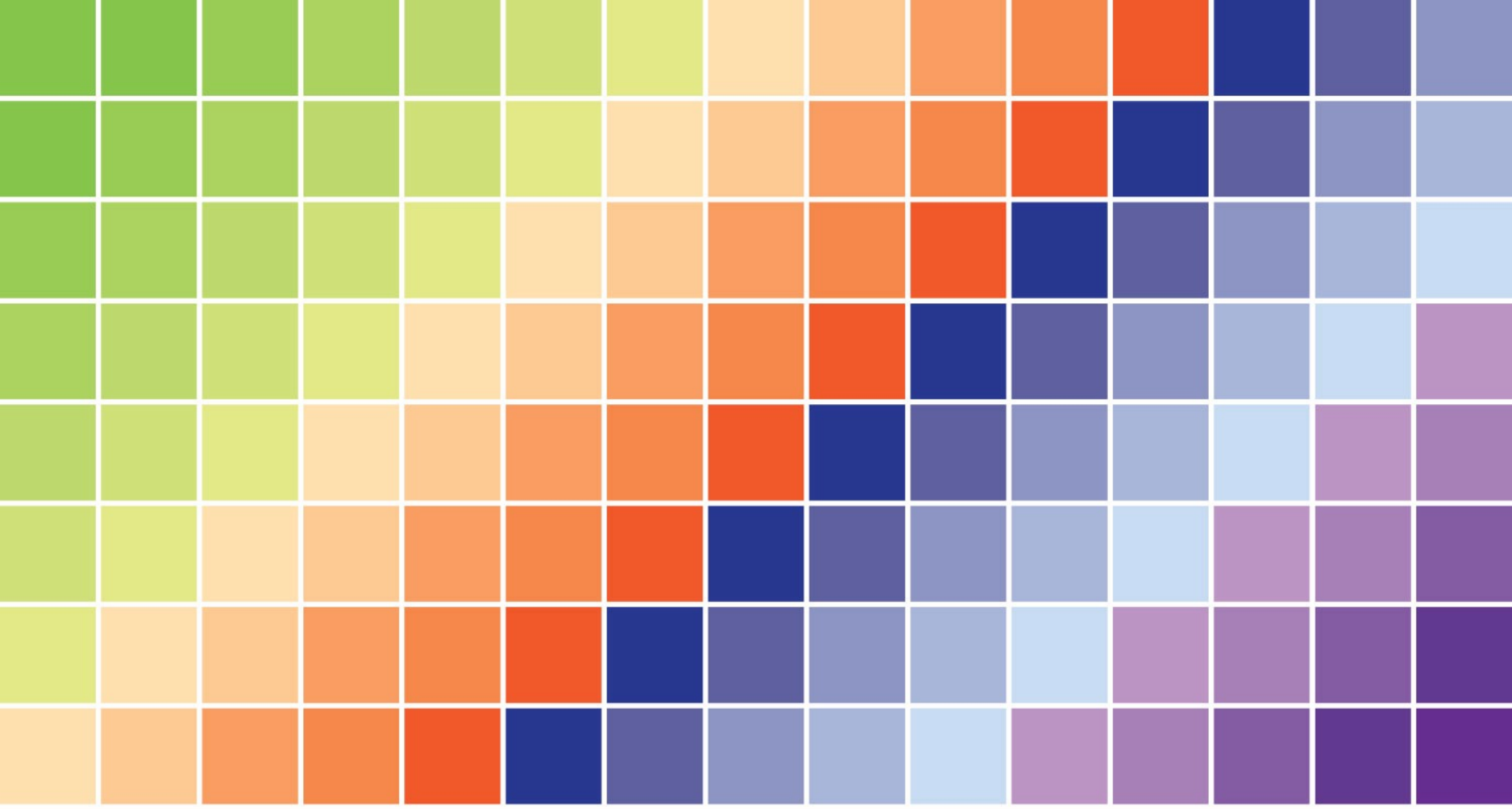

School of Computer Science

ПРИНЦИПЫ ООП. НАСЛЕДОВАНИЕ И АГРЕГАЦИЯ. ПОЛИМОРФИЗМ

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы

ВОПРОС
ЗАГОЛОВОК КЛАССА

- Определенный ниже класс:

```
>>> class password:

        pass
```

1. Имеет имя password и не наследуется от другого пользовательского класса
2. Имеет имя pass и унаследован от класса password
3. Имеет имя password и унаследован от класса pass
4. Не имеет имени и унаследован от pass

ВОПРОС
МЕТОД ИНИЦИАЛИЗАЦИИ

- Когда вызывается метод __init__:
  1. При создании экземпляра
  2. При объявлении класса
  3. При обращении к методу экземпляра
  4. При написании программистом его первой программы “Hello world”

ВОПРОС
SELF В МЕТОДЕ ИНИЦИАЛИЗАЦИИ

- В методе __init__ неявный аргумент self ссылается:

1. На файл __init__.py пакета модулей, в который входит этот класс
2. На последнюю объявленную переменную
3. На последний объявленный метод
4. На только что созданный объект данного класса

ВОПРОС
СВОЙСТВА

– Для чего используются свойства (@property и @xxx.setter):

1. Чтобы создать вычисляемый атрибут
2. Чтобы сделать атрибут приватным
3. Чтобы запретить использование атрибута в клиентском коде
4. Чтобы контролировать допустимые значения атрибута

ВОПРОС
СТРОКИ ДОКУМЕНТАЦИИ

– Классы в Python:

1. Как модули и функции, не могут иметь строки документации
2. По аналогии с модулями и функциями, могут иметь строки документации
3. Обязаны иметь DOC STRING как и модули с функциями
4. В отличие от модулей и функций могут и должны иметь строки документации

ВОПРОС
ОТСТУПЫ ПРИ ОПИСАНИИ КЛАССА

- Первая строка без отступа после определения класса:

1. Означает конец модуля с описанием класса
2. Означает переход к очередному условию вышестоящего оператора if
3. Не отличается от строки с отступом
4. Находится вне блока class

ВОПРОС
ИНКАПСУЛЯЦИЯ

- Инкапсуляция это:

1. Сбор всей востребованной атрибутики в классе с именем Capsule
2. То же самое, что абстракция и ограничение видимости атрибутов
3. Объединение в классе переменных экземпляра и методов, которые с ними работают

ВОПРОС
НЕЯВНЫЙ АРГУМЕНТ МЕТОДА

– Первым аргументом любого метода экземпляра класса, включая __init__, является ссылка на текущий объект, для которого вызывается этот метод. Принято назвать этот аргумент:

1. self
2. this
3. pass
4. result

ВОПРОС
СТАТИЧЕСКИЙ МЕТОД

- Статический метод класса:

1. Статичен, т.к. программируется один раз, а затем код блокируется для изменений
2. Не имеет доступа к экземпляру класса
3. Определяется за пределами объявления класса, но может использоваться в нем
4. Может быть только в системных классах Python, не доступен нам для создания

ВОПРОС
СТАТИЧЕСКИЙ МЕТОД. СИНТАКСИС

– Укажите верные утверждения про @staticmethod:

1. К этому методу можно обращаться от экземпляра класса
2. К этому методу можно обращаться от имени класса
3. Метод не принимает дополнительных аргументов, кроме указанных программистом
4. Метод первым аргументом принимает ссылку на экземпляр класса

ВОПРОС
ПОЛЯ КЛАССА В __INIT__

- В определенном ниже классе:

```
>>> class Sea:

        def __init__(self, depth):

            self.depth = depth
```

- Что такое self.depth:
  1. Класс
  2. Переменная экземпляра
  3. Экземпляр
  4. Другое

ВОПРОС
MAGIC-МЕТОДЫ

- Magic-методы класса имеют по два подчеркивания в начале и в конце своего имени. А вот почему они называются магическими:

1. Код с ними становится волшебным и защищен от сглаза, порчи и багов
2. Секрет фокуса неизвестен, т.е. никто не знает зачем нужны эти методы и когда они вызываются
3. Python вызывает их сам при управлении динамическим распределением памяти
4. Эти методы вызываются автоматически при выполнении определенных команд с экземплярами класса

ТЕМА СЕГОДНЯШНЕГО ЗАНЯТИЯ

- Основные принципы ООП:
  - Инкапсуляция и абстракция данных
  - Наследование
  - Полиморфизм
- Сегодня мы попробуем:
  - Создавать классы наследники
  - Использовать функцию super() для вызова методов базового класса
  - Специализировать методы в дочерних классах
  - Множественное наследование
  - Проверять принадлежность к классу
  - Агрегировать одни классы в другие (композицию классов)

НАСЛЕДОВАНИЕ
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

- Для чего нужно наследование:
  - Изменение поведения класса
  - Расширение функциональности класса

  (с использованием уже имеющегося кода)
- Класс, от которого наследуются – **предок**, **родительский**, **базовый**, **суперкласс**
- Класс-наследник – **потомок**, **дочерний**, **производный**, **подкласс**, **порожденный**
- Класс, создаваемый только для наследования – **абстрактный**

НАСЛЕДОВАНИЕ
ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

- Имеется базовый класс Pet «домашний питомец», в котором реализованы атрибут name, свойства noise и harm
- Создадим 2 класса, унаследованных от Pet – Dog и Cat – со следующей атрибутикой:
  - Атрибуты breed – порода домашнего питомца,
  - guard для собак и walk_wish для котов
  - Метод voice() – для озвучивания животного
- Добавим в базовый класс методы __eq__ и __gt__ для сравнения питомцев
- Доработаем пример в inherit_pet.py

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

- В Python разрешено наследоваться от нескольких классов
- Это используется также для обогащения получаемого класса функциональностью
- Те классы, которые добавляются к основной линии наследования называют классами-примесями

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
ПРАКТИЧЕСКОЕ ЗАДАНИЕ

- JSON (JavaScript Object Notation) – популярный формат данных для хранения и обмена между разнородными программами
- Данные по объекту в формате JSON хранятся в структуре подобной словарю Python
- Работа с этим форматом поддержана в Python в стандартном модуле json
- Необходимо с использованием множественного наследования получить класс ExpDog, который умеет сохранять свое состояние в JSON-строке

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
РЕШЕНИЕ

– Что выведет saved_dog.to_json():

```
>>> import json
>>> class ExportJSON:
        def to_json(self):
            return json.dumps({"name": self.name,
                               "breed": self.breed,
                               "noise": self.noise,
                               "harm": self.harm,
                               "guard": self.guard
                              })


>>> class ExpDog(Dog, ExportJSON):
        pass

>>> saved_dog = ExpDog("Май", "Golden Terrier", guard=0)
>>> saved_dog.to_json()
```

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
ЛИНЕАРИЗАЦИЯ КЛАССА

- Порядок поиска атрибутов и методов объекта по линиям наследования классов:

```
>>> #       object
>>> #         /    \
>>> #        /      \
>>> #     Pet     ExportJSON
>>> #      |        /
>>> #     Dog      /
>>> #       \     /
>>> #      ExpDog
>>>
>>> # Method Resolution Order
>>> ExpDog.__mro__
(<class '__main__.ExpDog'>, <class '__main__.Dog'>,
<class '__main__.Pet'>, <class '__main__.ExportJSON
'>, <class 'object'>)
```

ПРОВЕРКА ПРИНАДЛЕЖНОСТИ К КЛАССУ

Потомком какого класса мы являемся:

```
>>> issubclass(int, object)
True
>>> issubclass(Dog, object)
True
>>> issubclass(Dog, Pet)
True
>>> issubclass(Dog, Cat)
False
>>> issubclass(ExpDog, Pet)
True
```

Экземпляром какого класса мы являемся:

```
>>> isinstance(dog, Dog)
True
>>> isinstance(cat, Cat)
True
>>> isinstance(dog, object)
True
>>> isinstance(dog, Pet)
True
>>> isinstance(dog, Cat)
False
```

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
НЕДОСТАТКИ

- Большое количество классов-примесей порождает более сложный код
- При длинной иерархии наследования становится сложно проследить откуда какие атрибуты поступают в дочерний класс
- Совпадение имен переменных и методов у предков класса и неоднозначности пути наследования в случае более чем двухуровневой иерархии

МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ
УСЛОЖНЕНИЕ ПОДДЕРЖКИ

```
class ExportJSON:
    def to_json(self):
        pass

class ExportXML:
    def to_xml(self):
        pass

class MultiExpDog(Dog, ExportJSON, ExportXML):
    # основной класс для "мульти-экспортной собаки" при подходе с множественным
    # наследованием придется дорабатывать для нового вида экспорта
    pass

# В коде клиентской программы придется вызывать разные методы для разных
# вариантов экспорта, т.е. здесь также требуется дополнительное сопровождение
#>>> dog = MultiExpDog("Фокс", "мопс")
#>>> dog.to_xml()
#>>> dog.to_json()
```

КОМПОЗИЦИЯ КЛАССОВ
АЛЬТЕРНАТИВА МНОЖ. НАСЛЕДОВАНИЮ

- Создаем отдельную иерархию классов для экспорта собак PetExport – ExportDogXML / ExportDogJSON
- Класс ComposedExpDog получает ссылку на объект типа PetExport и хранит ее в атрибуте экземпляра класса
- «Собачьи» объекты вызывают один и тот же метод export() вне зависимости от формата
- Код основного класса не меняется, код клиентской программы тоже стабилен
- Пример в composition.py и comp_shell.txt

ПОЛИМОРФИЗМ
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

- **Полиморфизм** – это разное поведение одноименных методов для разных классов
- В зависимости от типов данных одна и та же операция дает разный результат
- Пример – мы можем сложить 2 числа и 2 строки и получим разный результат
- В сочетании с наследованием, полиморфизм использует возможности модификации метода предка в потомке

- ***Был ли полиморфизм где-то среди из рассмотренных ранее примеров?***

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

- Создать отдельную иерархию классов для экспорта котов PetExport – ExportCatXML / ExportCatJSON
- Создать класс ComposedExpCat, который получает ссылку на объект типа PetExport и хранит ее в атрибуте экземпляра класса
- Разные «Кошачьи» объекты должны вызывать один и тот же метод export() вне зависимости от формата, в котором они будут сохраняться – JSON или XML

ВЗАИМОДЕЙСТВИЕ ОБЪЕКТОВ В ИЕРАРХИИ

ПРОГРАММА «ШКОЛА»

- В скрипте school.py представлена иерархия объектов Member – Teacher/Student
- Доработайте её по этому плану:
  - Учитель должен уметь ***учить*** нескольких учащихся
  - Ученик должен уметь ***брать*** информацию и превращать ее в свои знания
  - Данные представляют собой ***список*** знаний, элементы из которого извлекаются случайным образом

ДОМАШНЕЕ ЗАДАНИЕ
ПРОГРАММА «ШКОЛА V.2.0»

- Предположим, некоторые ученики любят учиться и могут это делать без учителей:
  - Пусть такие ученики научатся чему-то сами
- Добавьте в класс Student метод, позволяющий ученику случайно "забывать" какую-нибудь часть своих знаний
- Ученики, освоившие все предметы, получают средний балл “5”
- Ученики, не освоившие ни одного предмета получают средний балл “2”
- В основном цикле программы выводите достижения своих учеников

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ ?

School of Computer Science