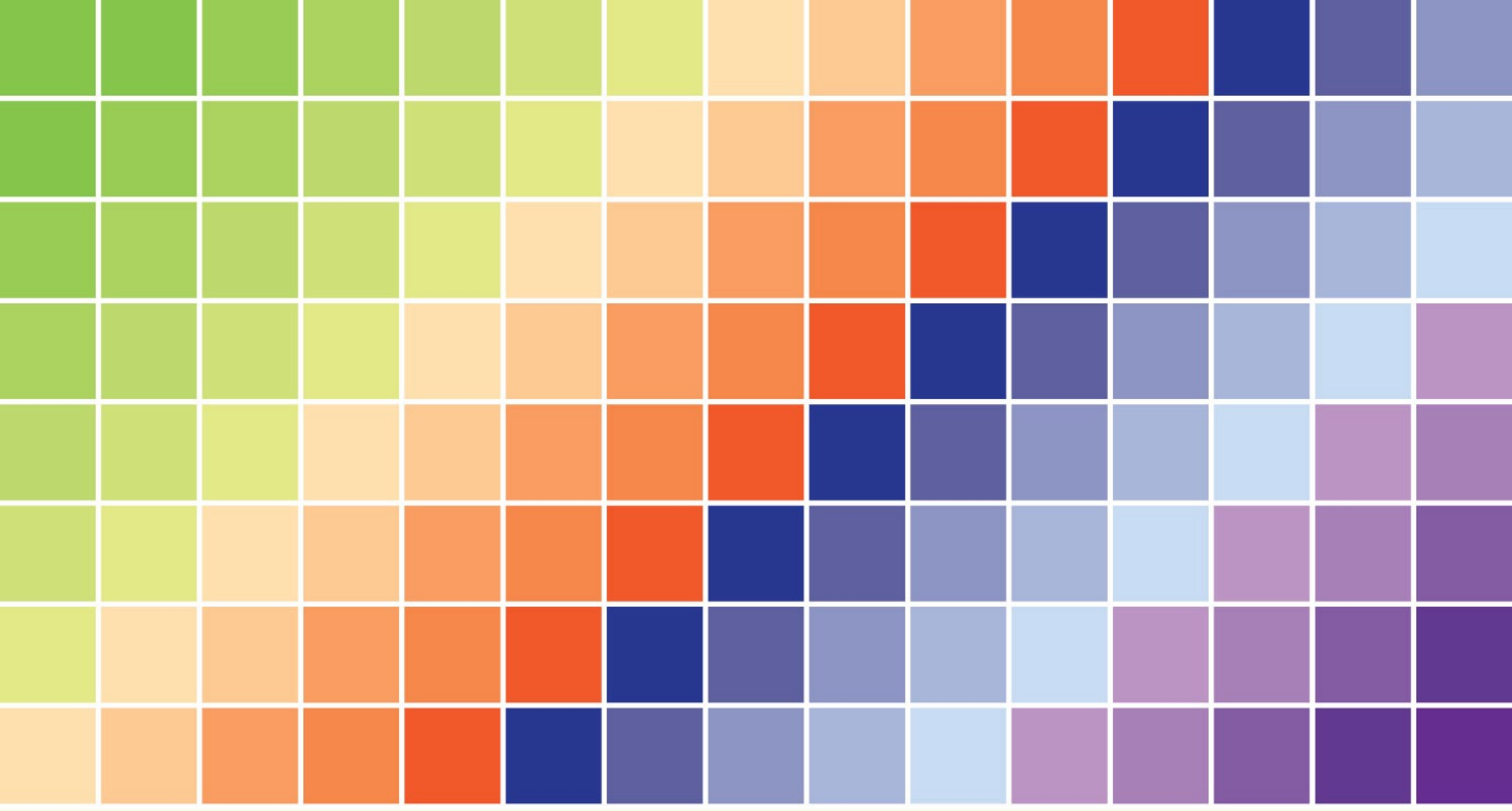

School of Computer Science

ПРАКТИЧЕСКОЕ ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП) – ЗАЧЕМ?

- Введение собственных типов данных:
  - Лучшее приближение к моделированию реального мира
  - Объединение общей функциональности в одну сущность, у которой есть свое состояние и поведение
- Лучший контроль над крупными программными проектами
- Повторное использование кода

ООП-ПОДХОД
ОБЪЕКТЫ И КЛАССЫ

- Все в Python является **объектом**, который наследуется от общего предка object

- Тип объекта называется **классом**

- Для примера введем в Shell команды :
  - type(object), help(object), dir(object)
  - type(str), type(int)
  - num = 1; num.__add__(2); dir(num)
  - dir(“строка”)
  - import sys
    type(sys)
    type(type(sys))

ООП-ПОДХОД
МЕТОДЫ

- Встроенные типы int, str, list,… содержат специальные функции, называемые методами
- Для примера введем в Shell команды:
  - dir(str)
  - help(str)
- Способы вызова метода на примере строки:

```
>>> str.count("teststr", "s")
2
>>> "teststr".count("s")
2
```

ВОПРОС
ФУНКЦИЯ ИЛИ МЕТОД?

- Была выполнена команда import math

- Некоторые из перечисленных ниже команд – вызовы функций, а некоторые – методов

- Укажите что здесь вызовы методов:

1. >>> math.trunc(3.2)
2. >>> len('abc')
3. >>> list.count(['a', 'b', 'a', 'c'], 'a')
4. >>> pow(3, 4)

ВОПРОС
ЭКВИВАЛЕНТНЫЕ ВЫЗОВЫ

- Есть 2 способа вызвать метод у объекта
- Какие пары строк эквивалентны друг другу:

1.
```
>>> '\nabc\t\t'.strip()
>>> str.strip('\nabc\t\t')
```

2.
```
>>> ['a', 'b', 'c'].count('b')
>>> list.count('b', ['a', 'b', 'c'])
```

3.
```
>>> list.append([1, 2, 3], 4)
>>> [1, 2, 3].append(4)
```

4.
```
>>> str.find('Раз уж начал – побеждай', 'а', 2)
>>> 'Раз уж начал – побеждай'.find('а', 2)
```

ООП – ПРАКТИЧЕСКИЙ ПРИМЕР
СОЗДАНИЕ «СВОЕЙ СТРОКИ»

- Создадим класс (тип данных), который ведет себя точно также как и обычная строка

- Тип (класс) WordplayStr – подтип (подкласс) для типа str и наследует все его свойства:

```
>>> class WordplayStr(str):
        """Унаследуем свой тип от строки"""

>>> w1 = WordplayStr()
>>> w2 = WordplayStr("wordplay")
>>> type(w1)
<class '__main__.WordplayStr'>
>>> type(w2)
<class '__main__.WordplayStr'>
>>> w1 == w2
False
```

ВОПРОС
НАСЛЕДОВАНИЕ ТИПОВ

Рассмотрите это описание класса:

```
>>> class MyInt(int):
        """Целочисленый класс, чтобы
           задать вопрос в процессе занятия"""
```

Выберите истинные высказывания:

1. int это подкласс класса MyInt
2. MyInt наследует все свойства типа int
3. Объект типа int это тоже самое, что и объект типа MyInt, но у int больше свойств
4. Любой метод класса int может быть вызван и для объекта класса MyInt

ООП – ПРАКТИЧЕСКИЙ ПРИМЕР
РАЗВИТИЕ «СВОЕЙ СТРОКИ»

- Создадим метод, который возвращает True, если строка типа WordplayStr начинается и заканчивается одной и той же буквой
- Первый параметр для **каждого** метода в класса, который работает с объектом, называют **self**
- **self** ссылается на объект, для которого вызывается метод
- См. пример в скрипте word_play.py

ВОПРОС
ТРЕБОВАНИЯ ДЕФИНИЦИИ МЕТОДА

Рассмотрите это описание класса:

```
>>> class MyInt(int):
        """Целочисленый класс, чтобы
           задать вопрос в процессе занятия"""
```

Один из этих заголовков для описания метода в классе MyInt некорректный.

Какой из них:

1. def is_larger(self, other)
2. def is_not_zero()
3. def is_not_zero(self)
4. def is_between(self, low, hi)

???

ВОПРОС
ПРОТОТИП МЕТОДА В DOCSTRING

Рассмотрите это описание класса:

```
>>> class MyInt(int):
        """Мой целочисленный класс"""
        def is_between(self, low, hi):
            """ <описание прототипа отсутствует>
            Возвращает True если low <= self <= hi
            """
```

Некоторые из описаний типов для docstring метода is_between() некорректные. Какие:

1. def is_between(int, MyInt, MyInt) -> bool
2. def is_between(MyInt, int, int) -> bool
3. def is_between(MyInt, MyInt, MyInt) -> bool
4. def is_between(int, int, int) -> bool

???

РЕАЛИЗАЦИЯ ОБЪЕКТА В PYTHON
СИНТАКСИС

Объявление класса (типа) объекта:

class ClassName [(класс_предок)]:
    <пространство_имен_класса>
    [def __init__(***параметры_создания***):
        <инициализация_объекта>]

Создание объекта нашего типа:

variable = ClassName (
[***параметры_создания_объекта - 1***])

РЕАЛИЗАЦИЯ ОБЪЕКТА В PYTHON
МЕТОД __INIT()__

- Инициализирующий метод __init__ вызывается неявно (по умолчанию) при создании объекта с типом «наш класс»
- Первый параметр метода __init__ обычно называется self
- self ссылается на объект, который инициализируется (создается)
- При создании объекта нашего типа аргумент self передавать не нужно (правило «минус 1 параметр»)

ПРАКТИКА. ШАГ №1
ПРОЕКТИРОВАНИЕ «СВЕРХУ ВНИЗ»

Создадим объект собственного типа данных:

```
if __name__ == "__main__":
    # Создадим объект банкомат, заправленный купюрами
    # 20x100руб, 40x500руб, 30x1000руб и 10x5000руб
    atm = ATMRegister(20, 40, 30, 10)
    print("Сумма денег в банкомате:", atm.get_total())

    # добавим 3 сотни и удалим 2 пятисотки
    atm.increase(3, 100)
    atm.decrease(2, 500)
    print("Сумма денег в банкомате:", atm.get_total())
```

См. скрипт atm_register1.py

СОЗДАНИЕ ОБЪЕКТА КЛАССА
ЭТАПЫ

- Создается объект типа ATMRegister
- Вызывается метод __init()__ этого объекта с неявной передачей ссылки на объект self в качестве первого параметра
- Прочие аргументы передаются в __init()__ из вызова ATMRegister(…)
- Возвращается ссылка на адрес размещения объекта в памяти

ПРАКТИКА. ШАГ №2
ПЕРЕМЕННЫЕ ЭКЗЕМПЛЯРА КЛАССА

- Каждый **объект** класса называется также **экземпляром** (**инстанцией**) этого класса
- Переменные внутри объекта называются его **полями/атрибутами/переменными** экземпляра класса
- Создадим поля класса с использованием точечной нотации **self.** …
- Доработанный скрипт atm_register2.py

ВОПРОС
ТЕРМИНОЛОГИЯ ООП

Некоторые из представленных ниже утверждений правдивы, а некоторые ложны.

Укажите правдивые утверждения:

1. Класс описывает поведение и свойства экземпляра этого класса
2. Класс – это объект экземпляра
3. Объект – это экземпляр класса
4. Объект, инстанция и экземпляр означают одно и то же

ВОПРОС
РАЗНЫЕ ОБЪЕКТЫ ОДНОГО КЛАССА

Вызов ATMRegister(5, 5, 5, 5) выполняется так:

- Создается объект типа ATMRegister
- Вызывается метод __init__ этого объекта
- Возвращается адрес в памяти данного объекта

Рассмотрите эти присваивания:

```
atm1 = ATMRegister(1, 2, 3, 4)   # atm1 → id3
atm2 = ATMRegister(4, 3, 2, 1)   # atm2 → id7
```

Какие из этих высказываний истинные:

1. atm1 ссылается на объект, размещенный в памяти по адресу id3
2. atm1 содержит адрес памяти id3
3. переменные atm1.note100 и atm2.note5000 обе ссылаются на значение 1
4. создание 2-го объекта ATMRegister перетирает значения переменных экземпляра объекта atm1

ПРАКТИКА. ШАГ №3
МЕТОДЫ ЭКЗЕМПЛЯРА КЛАССА

- Реализуем методы get_total(), increase() и decrease() в классе ATMRegister
- get_total() – подсчет общей суммы
- increase() – увеличение счетчиков купюр
- decrease() – уменьшение счетчиков купюр
- См. доработанный скрипт atm_register3.py

ШАГ №4. РЕФАКТОРИНГ
ОПТИМИЗАЦИЯ КОДА КЛАССА

В классе ATMRegister описаны поля для хранения номиналов 100, 500, 1000 и 5000 рублей. В методах increase()/decrease() у нас довольно длинное ветвление, чтобы определить поле для изменения

Какую коллекцию Python лучше использовать вместо этих 4-ех полей, чтобы переписать код методов increase() и decrease() в одну строку?

1. list of int
2. list of tuple (int, int)
3. dict of {int: int}

Оптимизируем код в ATMRegister с использованием выбранной коллекции – см. atm_register4.py

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ ?

School of Computer Science