Урок 2

Парадигма ООП: особенности и отличия от других ЯП

Сведения об ООП. Словари. Инкапсуляция, наследование, полиморфизм. Преимущества Python.

Словарь как структура данных

- ***Что такое словарь в Python? Когда и зачем нужно их использовать?***

В жизни много примеров работы с массивами данных, у которых можно выделить именованные параметры: у работников предприятия ими будут имя, фамилия, дата рождения, отдел и стаж. Для описания этих данных подходит словарь.

```
worker1 = {
      "name": "Сергей",
      "surname": "Петров",
      "birth_date": '01.07.1980',
      "department": "Бухгалтерия",
      "experience": "2 года"
}

worker2 = {
      "name": "Андрей",
      "surname": "Лагута",
      "birth_date": '15.06.1970',
      "department": "Цех №2",
      "experience": "5 лет"
}
```

Чтобы обрабатывать и изменять данные для каждого работника, пишем функции:

```
def worker_1_handling():
    name_val = worker1['name']
    surname_val = worker1['surname']
    birth_date_val = worker1['birth_date']
    department_val = worker1['department']
    experience_val = worker1['experience']
    print(f"Работник: {name_val} {surname_val}, дата рождения:\
    {birth_date_val}, отдел: {department_val}, опыт работы: {experience_val}")
worker_1_handling()
```

В приведенном примере используем f-строки — один из способов форматирования, появившийся в Python 3.6. F-строки обеспечивают подстановку в строку значений переменных из текущей области видимости. При этом в самой строке достаточно указания названия переменной в фигурных скобках.

Если в программе 20 разных структур и для каждой — по 10 функций обработки (немного для реальной программы), получится 200 разных функций, в которых можно запутаться.

Гораздо удобнее сгруппировать структуры данных с помощью классов.

Введение в ООП

- ***Объясните концепцию ООП. Когда лучше ее применять?***

Доска, стол, парта — все это привычные для нас объекты. У них есть присущие им параметры (свойства): цвет, размер, материал, вес. Рассматривая ООП-подход в программировании, будем проводить аналогии с объектами реального мира.

Чтобы писать с помощью ООП, нужно понять суть этого подхода и преимущества, которые он дает.

Можно ли жить без ООП?

Многие языки программирования не поддерживают эту концепцию. Все программы, написанные при помощи ООП, можно сделать и без него. Зачем тогда тратить столько времени на его изучение?

ООП — это инструмент, с помощью которого гораздо легче писать большие программы.

Python — объектно-ориентированный язык, в нем все является объектом. Но только поняв принцип ООП, вы сможете писать красивые, функциональные и легко масштабируемые программы.

Классы

- ***Что такое класс и чем он отличается от объекта?***

Табуретка — это объект. При ее изготовлении пользуются руководствами (чертежами). И если сказать «табуретка», не указывая на какую-то конкретную, то все поймут, о чем речь, так как знают особенности этого предмета.

Класс является описанием объектов определенного типа (таким чертежом). Это абстракция без материального воплощения, но с ее помощью можно систематизировать объекты. Ведь можно создать множество объектов, которые будут принадлежать одному классу. Может быть и класс без объектов, реализованных на его основе.

Класс — это пользовательский тип, состоящий из методов и атрибутов.

Если рассматривать класс с точки зрения ООП, он представляет собой коллекцию данных. Использование классов дает нам все преимущества абстрактного подхода в программировании.

Объект определенного класса создается путем вызова имени класса как функции с параметрами. Объект определенного класса состоит из атрибутов и методов.

Атрибут — это переменная класса, **метод** — это функция. Метод отличается от функции тем, что у него есть первый параметр — **self**. Он является ссылкой на тот объект, для которого был вызван метод. Метод через **self** всегда имеет доступ к атрибутам и другим методам своего объекта.

Основными свойствами ООП являются инкапсуляция, наследование и полиморфизм.

Примеры

```
# class - шаблон для создания объектов
# Классы содержат атрибуты - данные, и методы - функции для обработки данных
class Student:
    # функция-конструктор - запускается автоматически при создании объекта
    # (экземпляра класса)
    def __init__(self, name, surname, birth_date, school, class_room):
        self.name = name
        self.surname = surname
        self.birth_date = birth_date
        self.school = school
        self.class_room = class_room

    # метод
    def get_full_name(self):
        return print(self.name + ' ' + self.surname)
```

Можно создать сколько угодно объектов, используя класс как шаблон.

```
student1 = Student("Петр", "Сидоров", '10.01.1995', "8 гимназия", "8 Б")
```

Начальные аргументы передаются в конструктор класса. Класс является шаблоном, который описывает структуру и поведение объекта.

Разделитель в виде точки (.) используется для обращения к атрибутам объекта.

```
# Выводим текущий класс первого ученика
print(student1.class_room)
```

Методы класса — это обычные функции, которые получают в качестве первого аргумента ссылку на экземпляр класса.

```
# Вызываем метод для получения полного отображаемого имени студента
print(student1.get_full_name())
```

По сути, вызов **student1.get_full_name()** равносилен вызову **get_full_name(student1)**, только в случае с методами ссылка на экземпляр передается автоматически.

```
# Можно изменить значение любого атрибута, присвоив ему новое значение
student1.name = 'Вася'
print(student1.name)
```

Основные свойства ООП

Инкапсуляция

- ***Что такое инкапсуляция?***

Инкапсуляция — это ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонентов доступными только внутри класса. Это могут быть служебные переменные и методы, которые необходимо использовать для работы класса, но нежелательно держать доступными для пользователей или сторонних лиц.

Главная цель инкапсуляции — скрыть внутреннюю логику реализации класса от пользователей. Для доступа к данным, которые хранятся в классе, и работы с ними используются публичные методы.

В Python инкапсуляция работает лишь на уровне соглашения между программистами о том, какие данные являются общедоступными, а какие — скрытыми от конечного пользователя. Для инкапсулирования переменных используется символ подчеркивания (_). Но практического смысла он не имеет, поскольку даже при его наличии можно получить доступ к переменной класса или его методу. Этот символ необходим прежде всего как указатель для команды разработчиков, что данную переменную или метод не следует использовать вне класса.

Рассмотрим пример — класс, соответствующий товару в заказе. Клиент одновременно может заказать не более 10 единиц:

```
class Item:
    def __init__(self, name, quant):
        self._name = name
        self._quant = quant

item = Item("Epson", 3)
print(item._name, item._quant)
```

Результат:

```
Epson 3
```

Мы без проблем получили доступ к наименованию товара и его количеству, хотя соответствующие переменные объявлены как приватные. Чтобы ограничить доступ к ним, необходимо воспользоваться двойным подчеркиванием. Такой подход даст защиту: получить содержимое переменных мы уже не сможем.

```
class Item:
    def __init__(self, name, quant):
        self.__name = name
        self.__quant = quant

item = Item("Epson", 3)
print(item._name, item._quant)
```

Результат:

```
AttributeError: 'Item' object has no attribute '_name'
```

Получить доступ к «закрытым» переменным все же можно, поскольку на практике бывает такая потребность. Для этого создаются специальные методы: геттеры (получение данных) и сеттеры (изменение данных). Модифицируем рассмотренный выше пример:

```
class Item:
    def __init__(self, name, quant):
        self.__name = name
        self.__quant = quant

    def get_quant(self):
        return self.__quant

    def get_name(self):
        return self.__name

item = Item("Epson", 3)
```

Попробуем здесь получить содержимое переменных:

```
print(item.get_name(), item.get_quant())
```

Результат:

```
Epson 3
```

Сделаем так, что переменная **self.__quant** будет иметь строго определенное значение, которое необходимо изменить:

```
class Item:
    def __init__(self, name):
        self.__name = name
        self.__quant = 3

    def set_quant(self, quant):
        if quant in range(1, 10):
            self.__quant = quant
        else:
            print("Превышено допустимое количество товара в заказе")

    def get_quant(self):
        return self.__quant

    def get_name(self):
        return self.__name

item = Item("Epson")
```

Изменим содержимое переменной:

```
item.set_quant(5)
print(item.get_name(), item.get_quant())
```

Результат:

```
Epson 5
```

Наследование

- ***Что такое наследование?***

Наследование — способ создавать специализированные классы на основе базовых. Это позволяет не писать код повторно. Наследование является важнейшей составляющей ООП, и в данном случае под ним подразумевается наличие классов и подклассов — их также называют родительскими и дочерними классами. Последние наследуют атрибуты родителей, а также могут переопределять атрибуты и добавлять свои. Связь родительского и дочернего класса осуществляется через дочерний: классы-родители перечисляются после имени дочернего в скобках.

```
Epson 5
```

Создадим на основе предыдущего примера дочерний класс **ItemReport** на базе класса-родителя **Item**:

```
class Item:
    def __init__(self, name, quant):
        self.__name = name
        self.__quant = quant

    def get_quant(self):
        return self.__quant

    @property
    def get_name(self):
        return self.__name


class ItemReport(Item):

    def get_info(self, date):
        print(self.get_name, " - товар заказан: ", date)

item = ItemReport("Epson", 3)
item.get_info("25.09.2017")
```

Результат:

```
Epson  - товар заказан:  25.09.2017
```

Переменная **self.__name** объявлена как приватная и доступа к ней нет. Чтобы получить ее значение, используем метод **get_name**. Для него определяем специальный декоратор **@property**, который выполняет ряд полезных задач — в частности, конвертацию метода класса в атрибуты только для чтения, а также преобразование геттеров и сеттеров в атрибуты.

Полиморфизм

- ***Что такое полиморфизм?***

Полиформизм — это возможность использовать один интерфейс для различных базовых элементов (типов данных и классов). При этом функции могут оперировать объектами разных классов. Значит конкретный объект, относящийся к определенному классу, может быть использован так же, как если бы он был другим объектом иного класса.

Полиформизм применяется, когда классы и подклассы обладают общими методами. Это позволяет функциям оперировать объектами любого из этих полиморфных классов, не определяя различий между ними. Если в нескольких классах находятся методы, имеющие одинаковые имена, но реализуемые по-разному, их называют полиморфными.

Полиморфные классы

Рассмотрим полиморфизм классов на примере создания двух различных классов и соответствующих им объектов. Добавим в этих классах несколько одноименных методов с разной функциональностью. Говорят, эти классы обладают общим интерфейсом.

```
class FirstClass():

    def first_method(self):
        print("Первый метод первого класса")

    def second_method(self):
        print("Второй метод первого класса")

    def third_method(self):
        print("Третий метод первого класса")


class SecondClass():

    def first_method(self):
        print("Первый метод второго класса")

    def second_method(self):
        print("Второй метод второго класса")

    def third_method(self):
        print("Третий метод второго класса")
```

На основе данных классов создадим объекты:

```
fc = FirstClass()
fc.first_method()

sc = SecondClass()
sc.first_method()
```

Результат:

```
Первый метод первого класса
Первый метод второго класса
```

Полиморфизм в методах классов

Создадим цикл **for** для перебора кортежей объектов (экземпляров созданных классов). Затем вызовем методы, не указывая, к какому классу относится каждый из них. Достаточно только указания метода:

```
for obj in (fc, sc):
    obj.first_method()
    obj.second_method()
    obj.third_method()
```

Результат:

```
Первый метод первого класса
Второй метод первого класса
Третий метод первого класса
Первый метод второго класса
Второй метод второго класса
Третий метод второго класса
```

Цикл **for** сначала выполнял операции с объектом класса **FirstClass**, затем — с объектом класса **SecondClass**. В примере Python оперирует данными методами, не зная точно, к какому классу принадлежит каждый из них.

Полиморфизм в функциях

Создадим функцию для работы с объектами и назовем ее **obj_handler()**. Она будет принимать объект **obj** (а вообще сможет принять любой существующий объект).

```
def obj_handler(obj):
```

Выполним вызов метода **first_method()**. Как вы помните, данный метод определен в каждом нашем классе.

```
def obj_handler(obj):
    obj.first_method()
```

Далее необходимо убедиться, что экземпляры классов **FirstClass** и **SecondClass** созданы:

```
fc = FirstClass()
sc = SecondClass()
```

Теперь вызовем функции **obj_handler** и в качестве аргументов укажем ссылки на экземпляры классов:

```
obj_handler(fc)
obj_handler(sc)
```

Результат:

```
Первый метод первого класса
Первый метод второго класса
```

Полиморфизм заключается в том, что в разных объектах одна операция может выполнять различные функции. Слово «полиморфизм» имеет греческое происхождение и означает «имеющий многие формы». Простым примером полиморфизма может служить функция **count()**, выполняющая одинаковое действие для различных типов объектов: **'abc'.count('a')** и **[1, 2, 'a'].count('a')**. Оператор + полиморфичен при сложении чисел и строк.

С помощью полиморфизма мы можем работать с объектами, не задумываясь, к какому типу они принадлежат.

```
>>> 2 + 4
 6
>>> '2' + '4'
 '24'
>>> (2, 4)+(2, 4)
 (2, 4, 2, 4)
```

Результат операции зависит от реализации метода, соответствующего применяемой ей.

Мы можем не знать, с какими объектами работать. Главное — знать, что данный тип объекта поддерживает эту операцию. Это очень удобно.

Отличия Python от других языков программирования

Java

- ***Чем Python отличается от Java?***

Программы, написанные на Python, медленнее Java-программ. Но существенным плюсом Python является то, что на разработку уходит в два раза меньше времени.

Еще Python-программы в 3–4 раза короче, чем аналогичные на Java, так как в Python на высоком уровне встроена разница в типах данных и динамической типизации.

Python-программист не тратит время на объявление типов аргументов или переменных. Доступно множество полиморфных словарей и списков, а также синтаксическая поддержка, встроенная в язык. Эти возможности применяются практически в каждой Python-программе.

Но из-за встроенной динамической типизации Python-программы выполняются медленнее, чем написанные на Java. К примеру, вычисляя выражение **a+b**, программа изначально определяет тип объектов **a** и **b**. После она запрашивает операцию сложения, которая может быть перегружена пользователем.

Перегрузка операторов — это механизм переопределения операторов языка с помощью специальных методов в классах. Имена таких методов начинаются с символа двойного подчеркивания и заканчиваются им же.

Операторами в данном случае являются не только знаки +, –, *, /, реализующие возможность сложения и вычитания, но и специфические конструкции, которые отвечают за создание объекта, вызов его как функции, доступ к элементу объекта по индексу, вывод объекта и так далее.

Примеры методов перегрузки объекта:

- **__init__()**. Данный метод является конструктором объектов класса, инициируется при создании объекта.
- **__str__()**. Метод отвечает за преобразование объекта к строковому формату.

- **__add__()**. Метод, отвечающий за перегрузку оператора сложения. Его вызов осуществляется при участии объекта в операции сложения в качестве операнда с левой стороны.

В Python реализованы и другие методы перегрузки операторов. В целом, такой механизм в пользовательских классах применяется достаточно редко, за исключением метода-конструктора класса.

Пример перегрузки в Java:

```
class MyClass{
    int x,y;
    MyClass()
    {}
    MyClass(int x1,int y1){
    x=x1;
    y=y1;
    }
    MyClass Add(MyClass C1,MyClass C2){
        MyClass CSum=new MyClass();
        CSum.x=C1.x+C2.x;
        CSum.y=C1.y+C2.y;
        return CSum;
    }
}

class ComplexMyClass{
    public static void main(String[] a){
        MyClass C1=new MyClass(3,4);
        MyClass C2=new MyClass(5,6);
        MyClass C3=new MyClass();
        C3=C3.Add(C1,C2);
        System.out.println("(" + C3.x +", " + C3.y + ")");
    }
}
```

Результат:

```
(8, 10)
```

Аналогичный пример перегрузки, но в Python:

```
class MyClass:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # Возвращает строковое представление объекта
    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

    # Сложение соответствующих параметров экземпляров классов
    def __add__(self, other):
        return MyClass(self.x + other.x, self.y + other.y)

x = MyClass(3, 4)
y = MyClass(5, 6)
print(x + y)
```

Результат:

```
(8, 10)
```

Плюс Java — эффективное сложение чисел: и простых, и с плавающей запятой. Но это требует задавать тип переменных, и язык не позволяет пользователю написать перегрузку этого метода.

Python лучше подходит как интегрирующий язык, а Java можно характеризовать как низкоуровневый. Вместе они — превосходная комбинация. Компоненты могут разрабатываться на Java, а Python можно использовать для прототипирования этих компонентов, пока их модель не будет разработана на Java.

JavaScript

- ***Чем Python отличается от JavaScript?***

Python во многом совпадает с Javascript — особенно в том, что касается объектов. Разработка на Python, как и на JavaScript, предполагает применение простых функций и переменных. Но Python позволяет повторно использовать код и писать более крупные программы.

Словари Python похожи на ассоциативные массивы JavaScript.

В Python отступы являются частью логики языка. В JS все блоки кода определяются фигурными скобками. Практически любой код, написанный на JavaScript, можно запустить в браузере. А для Python нужно устанавливать интерпретатор.

Perl

- ***Чем Python отличается от Perl?***

Изначально эти языки появились из скриптов под Unix, и у них много общего, но совершенно разная философия.

Изначально в Perl упор сделан на поддержку задач с ориентацией на приложение. К примеру, в нем есть встроенные регулярные выражения, а также функционал для генерации отчетов и сканирования данных. Python поддерживает общие методологии, такие как объектно-ориентированное программирование и структурирование данных.

В Python отсутствуют префиксы у переменных, что делает код более компактным. Есть элегантная система обозначений, посредством которой можно писать удобный для чтения и поддержки код. Python имеет механизм декораторов, который отсутствует в Perl.

C++

- ***Чем Python отличается от C++?***

Все, что сказано выше о Java, относится и к C++. С одной поправкой — код на Python короче разработок на C++ не в 3–5 раз, как при сравнении с Java, а в 5–10. Python великолепен в качестве языка интегрирования, когда используется для сборки компонентов, написанных на C++.

Программа, написанная на C++, не запускается с помощью интерпретатора, ее нужно компилировать. Но в основном современные операционные системы и драйвера написаны на C или на C++, что обеспечивает максимальное быстродействие. C++ работает практически на всех современных процессорах, но на чистом языке нельзя создать программу — разработчики устройств поддерживают индивидуальные для каждого устройства зависимости.

Практическое задание

1. Проверить механизм наследования в Python. Для этого создать два класса. Первый — родительский (**ItemDiscount**), должен содержать статическую информацию о товаре: название и цену. Второй — дочерний (**ItemDiscountReport**), должен содержать функцию (**get_parent_data**), отвечающую за отображение информации о товаре в одной строке. Проверить работу программы, создав экземпляр (объект) родительского класса.

2. Инкапсулировать оба параметра (название и цену) товара родительского класса. Убедиться, что при сохранении текущей логики работы программы будет сгенерирована ошибка выполнения. Усовершенствовать родительский класс таким образом, чтобы получить доступ к защищенным переменным. Результат выполнения заданий 1 и 2 должен быть идентичным.

3. Реализовать возможность переустановки значения цены товара. Необходимо, чтобы и родительский, и дочерний классы получили новое значение цены. Следует проверить это, вызвав соответствующий метод родительского класса и функцию дочернего (функция, отвечающая за отображение информации о товаре в одной строке).

4. Реализовать расчет цены товара со скидкой. Величина скидки должна передаваться в качестве аргумента в дочерний класс. Выполнить перегрузку методов конструктора дочернего класса (метод **__init__**, в который должна передаваться переменная — скидка), и перегрузку метода **__str__** дочернего класса. В этом методе должна пересчитываться цена и возвращаться результат — цена товара со скидкой. Чтобы все работало корректно, не забудьте инициализировать дочерний и родительский классы (вторая и третья строка после объявления дочернего класса).

5. Проверить на практике возможности полиморфизма. Необходимо разделить дочерний класс **ItemDiscountReport** на два класса. Инициализировать классы необязательно. Внутри каждого поместить функцию **get_info**, которая в первом классе будет отвечать за вывод названия товара, а вторая — его цены. Далее реализовать выполнение каждой из функции тремя способами.

Дополнительные материалы

1. https://habr.com/post/370831/.
2. https://habr.com/company/cit/blog/262887/.
3. http://www.quizful.net/category/python.
4. https://habr.com/post/49671/.
5. http://kip-world.ru/kakie-byvayut-tipy-dannyh-v-python.html.

Используемая литература

1. Учим Python качественно (habr).
2. Самоучитель по Python.
3. Лутц М. Изучаем Python. — М.: Символ-Плюс, 2011 (4-е издание).
4. 15 основных вопросов для Python собеседования.
5. 20 вопросов и ответов из интервью на позицию Python-разработчика.
6. Полиморфизм в Python.