| Региональный конкурс методических разработок в системе СПО «Профессионально-ориентированное содержание урока общеобразовательной дисциплины»<br><br>Номинация: Математика и информатика | Конкурсный №<br><br>27 |
|---|---|

**МЕТОДИЧЕСКАЯ РАЗРАБОТКА УРОКА**

**по общеобразовательной дисциплине:**

**ОУП.05 «Информатика»**

**ОП.04 «Основы алгоритмизации и программирования»**

**на тему: «Разработка и управление коллекциями объектов: классы и массивы»**

**для обучающихся по специальности:**

**09.02.07 «Информационные системы и программирование»**

**Разработчик:** преподаватель информатики ГАПОУ «Самарский государственный колледж» Моргунова Анастасия Викторовна

Урок посвящен изучению основ разработки и управления коллекциями объектов в программировании. Студенты узнают о понятиях классов и массивов, их назначении и принципах работы.

В ходе урока будут рассмотрены ключевые аспекты создания собственных классов, их методы и свойства. Разработка урока может быть полезна всем, кто изучает программирование и хочет углубить свои знания в области работы с объектами и коллекциями данных.

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

В современном информационном обществе, где компьютеры и программное обеспечение занимают важное место в повседневной жизни, освоение программирования становится ключевым элементом образовательного процесса. Развитие информационных технологий предоставляет уникальные возможности для совершенствования образования, и в этом контексте изучение структур данных, таких как классы и массивы, играет фундаментальную роль.

Выбор темы «Разработка и управление коллекциями объектов: классы и массивы» обусловлен не только актуальностью этих концепций в современном программировании, но и их важностью для формирования базовых навыков будущих специалистов в области информационных технологий. Эта тема напрямую связана с основами объектно-ориентированного программирования и предоставляет студентам практические навыки по созданию и управлению структурированными данными. Понимание принципов работы с классами и массивами позволяет разработчикам создавать более эффективные и гибкие программы, а также упрощает процесс разработки и отладки кода. Кроме того, изучение данной темы способствует развитию алгоритмического мышления и формированию базовых навыков, необходимых для дальнейшего изучения программирования.

В данной работе будет рассмотрено не только теоретическое основание классов и массивов, но и их применение на практике.

Урок «Разработка и управление коллекциями объектов: классы и массивы» занимает важное место в образовательном процессе, так как он является частью изучения основ программирования и помогает студентам лучше понять механизмы работы с данными. Этот урок может быть интегрирован в различные курсы по программированию, начиная от базовых и заканчивая более продвинутыми. Кроме того, его можно использовать как самостоятельную единицу для обучения разработке и управлению коллекциями объектов, что позволит студентам получить более глубокое понимание этой темы.

В результате проведения занятия обучающийся должен освоить основной вид деятельности: Разработка модулей программного обеспечения для компьютерных систем и соответствующие ему общие и профессиональные компетенции:

Перечень общих компетенций (код и наименование):

ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности;

Перечень профессиональных компетенций (код и наименование):

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

Таким образом, данная тема не только отвечает на актуальные вызовы современного информационного общества, но и способствует развитию компетенций, необходимых для успешной карьеры в области программирования и информационных технологий.

## ТЕХНОЛОГИЧЕСКАЯ КАРТА

Тема урока «Разработка и управление коллекциями объектов: классы и массивы» предполагает изучение принципов разработки и управления коллекциями объектов, а также использование классов и массивов для хранения и обработки данных.

Методическое обоснование этой темы заключается в подготовке специалистов, способных разрабатывать эффективные и надежные программы, используя различные структуры данных и алгоритмы для решения задач. В процессе обучения студенты должны изучить следующие аспекты:

1) Понимание принципов объектно-ориентированного программирования, включая инкапсуляцию, наследование, полиморфизм и абстракцию.

2) Изучение принципов и методов разработки коллекций объектов, включая принципы проектирования коллекций, выбор структуры данных для конкретной задачи, а также методы доступа и модификации данных.

3) Использование массивов как одной из основных структур данных для хранения коллекций объектов. Изучение методов работы с массивами, таких как создание, заполнение, сортировка, поиск и удаление элементов.

4) Изучение классов как основного инструмента для создания и управления объектами, включая понимание принципов работы конструкторов и деструкторов, а также принципов перегрузки операций.

5) Применение принципов разработки универсальных алгоритмов и структур данных, которые могут быть использованы для различных задач и проблем.

Методические рекомендации для проведения занятия включают следующие пункты:

1. Использование практических примеров для демонстрации принципов работы с классами и массивами.

2. Применение интерактивных методов обучения, таких как работа в группах, выполнение практических заданий, использование онлайн-тренажеров и симуляторов.

3. Разработка методических материалов, включающих примеры кода, учебные пособия, видеоматериалы и другие ресурсы для поддержки процесса обучения.

4. Регулярное обновление методических материалов и программ обучения в соответствии с новыми технологиями и стандартами

**Описание основных этапов занятия**

| Этапы занятия, Продолжительность в мин. | Деятельность преподавателя | Деятельность студентов | Планируемые образовательные результаты | Типы оценочных мероприятий | Дидактические материалы, МТО |
|---|---|---|---|---|---|
| **1. Организационный этап занятия** | | | | | |
| Вхождение в тему и создание условий для осознанного восприятия нового материала, 10 мин. | - Организует начало занятия, выясняет отсутствующих; проверяет готовность группы к занятию.<br>- Представление темы занятия.<br>- Объявление целей и ожидаемых результатов занятия.<br>- Объяснение структуры занятия и плана занятия. | - Приветствие преподавателя.<br>- Докладывают о готовности к занятию; называют отсутствующих на занятии, причины отсутствия, сосредотачиваются, настраиваются на занятие. | - Готовность к труду, осознание ценности мастерства, трудолюбие.<br>- Определять цели деятельности, задавать параметры и критерии их достижения. | Оценка студентов осуществляется на данном этапе через участие в обсуждении и готовности к занятию. | Презентация к занятию на тему: «Разработка и управление коллекциями объектов: классы и массивы».<br>- Проектор, интерактивная доска.<br>- ПК с установленным IDE и редактором кода для Python. |
| **2. Основной этап занятия** | | | | | |
| Освоение нового материала, 45 мин. | - Познакомить студентов с основными понятиями об иерархии классов в ООП.<br>- Демонстрация кода на языке программирования Python, иллюстрирующего концепции классов и объектов.<br>- Обсуждение свойств и | - Делают запись основных понятий об иерархии классов в ООП в тетради.<br>- Участие в обсуждении теоретического материала. | - Привить интерес к различным сферам профессиональной деятельности.<br>- Устанавливать существенный признак или основания для сравнения, классификации и обобщения. | Оценка активности студентов во время обсуждения. | - Примеры кода на языке программирования Python.<br>- Проектор, интерактивная доска.<br>- ПК с установленным IDE и редактором кода для Python. |

| | | | | | |
|---|---|---|---|---|---|
| | методов классов. | | | | |
| Применение изученного материала, 50 мин. | - Демонстрация различных методов и подходов к решению задач с использованием классов ООП.<br>- Обсуждение и решение задач на тему: «Разработка и управление коллекциями объектов: классы и массивы» на примере структурированных типов данных и массивов на языке программирования Python.<br>- Разбор возможных ошибок и проблем при работе с классами ООП и их устранение. | - Обсуждение этапов решения задач.<br>- Выполнение практического задания, связанной с созданием классов и объектов, совместно с преподавателем. | - Использовать средства информационно-коммуникационных технологий в решении когнитивных, коммуникативных и организационных задач с соблюдением требований эргономики, техники безопасности, гигиены, ресурсосбережения, правовых и этических норм, норм информационной безопасности.<br>- Уметь читать и понимать программы, реализующие несложные алгоритмы обработки числовых и текстовых данных (в том числе массивов и символьных строк) на выбранном для | - Оценка активности студентов во время обсуждения.<br>- Оценка выполнения заданий в соответствии с критериями оценивания. | - Задачи на языке программирования Python.<br>- Проектор, интерактивная доска.<br>- ПК с установленным IDE и редактором кода для Python. |

| | | | | | |
|---|---|---|---|---|---|
| | | | изучения универсальном языке программирования высокого уровня.<br>- Уметь реализовать этапы решения задач на компьютере.<br>- Умение реализовывать на выбранном для изучения языке программирования высокого уровня типовые алгоритмы обработки чисел, числовых последовательностей и массивов. | | |
| Закрепление изученного материала, 50 мин. | - Организация самостоятельного выполнения практической работы обучающимися.<br>- Контроль выполнения практических заданий студентами. | - Самостоятельное выполнение практической работы на персональном компьютере.<br>- Тестирование и оптимизация программного кода на языке программирования Python. | - Использовать средства информационно-коммуникационных технологий в решении когнитивных, коммуникативных и организационных задач с соблюдением требований эргономики, техники безопасности, гигиены, ресурсосбережения, правовых и | - Оценка выполнения практической работы в соответствии с критериями оценивания. | - Задачи на языке программирования Python.<br>- Проектор, интерактивная доска.<br>- ПК с установленным IDE и редактором кода для Python. |

| | | | | | |
|---|---|---|---|---|---|
| | | | этических норм, норм информационной безопасности.<br>- Уметь читать и понимать программы, реализующие несложные алгоритмы обработки числовых и текстовых данных (в том числе массивов и символьных строк) на выбранном для изучения универсальном языке программирования высокого уровня.<br>- Уметь реализовать этапы решения задач на компьютере.<br>- Умение реализовывать на выбранном для изучения языке программирования высокого уровня типовые алгоритмы обработки чисел, числовых последовательностей и массивов. | | |

| **3. Заключительный этап занятия** | | | | | |
|---|---|---|---|---|---|
| Диагностика, 10 мин. | - Предлагает обучающимся оценить результаты своей деятельности. Раздача чек-листов для выявления уровня освоения темы каждому студенту. | - Оценивают свою деятельность с использованием чек-листов. | - Вносить коррективы в деятельность, оценивать соответствие результатов целям, оценивать риски последствий деятельности. | - Оценка уровня освоения темы обучающимися. | Раздаточный материал. |
| Подведение итогов, домашнее задание, 15 мин. | - Подведение итогов занятия. Выставление оценок.<br>- Выдает домашнее задание, объясняет алгоритм выполнения задания и указывает источники для дополнительного изучения. | - Обсуждение домашнего задания.<br>- Запись комментариев по домашнему заданию в тетрадь. | - Уметь организовывать личное информационное пространство с использованием различных средств цифровых технологий. | - Оценка участия в обсуждении. | - Домашние задание на создание классов и использование массивов на языке программирования Python.<br>- Источники для самостоятельного изучения (учебники и документация Python). |

## ЗАКЛЮЧЕНИЕ

В заключение данной разработки урока по теме «Разработка и управление коллекциями объектов: классы и массивы» можно подвести следующие итоги:

1. Этот урок является важным этапом обучения, направленным на формирование у студентов глубокого понимания концепций объектно-ориентированного программирования и практических навыков работы с классами и массивами.

2. В ходе урока студенты ознакомились с основными принципами создания и управления коллекциями объектов, что предоставляет им важные инструменты для более эффективного решения задач в программировании. Они изучили преимущества и особенности использования классов и массивов, а также научились применять их в практических сценариях.

3. Проблемные вопросы, поставленные педагогом, были рассмотрены и разъяснены в ходе урока. Студенты смогли понять, как правильно проектировать классы, как эффективно использовать массивы для хранения данных, и как управлять коллекциями объектов в контексте реальных программных проектов.

4. Этот урок не только предоставил студентам необходимые теоретические знания, но и способствовал их применению на практике через разработку соответствующих задач и упражнений. Студенты, таким образом, получили ценный опыт, который будет полезен им в будущей профессиональной деятельности.

В целом, разработанный урок оказался полноценным инструментом для формирования у студентов компетентности в области разработки и управления коллекциями объектов, и его успешная реализация содействует укреплению фундаментальных знаний, необходимых для успешного вхождения в мир программирования.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Гуриков С.Р. Основы алгоритмизации и программирования на Python: Учебное пособие / С.Р. Гуриков. - Москва: ИНФРА-М, 2023. - 343 с. - (Среднее профессиональное образование). - ISBN 978-5-16-016906-4. - Текст: электронный. - URL: https://znanium.com/catalog/product/1927269

2) Жуков Р.А. Язык программирования Python. Практикум: учебное пособие / Р.А. Жуков. - Москва: ИНФРА-М, 2023. - 216 с. Доп. материалы [Электронный ресурс]. - (Среднее профессиональное образование). - ISBN 978-5-16-015638-5. - Текст: электронный. - URL: https://znanium.com/catalog/product/1916202

3) Щерба А.В. Программирование на Python: Первые шаги: практическое руководство / А.В. Щерба. - Москва: Лаборатория знаний, 2022. - 253 с. - (Школа юного программиста). - ISBN 978-5-93208-578-3. - Текст: электронный. - URL: https://znanium.com/catalog/product/1988404

## ПРИЛОЖЕНИЕ № 1

### Теоретический материал к практическому занятию на тему: «Разработка и управление коллекциями объектов: классы и массивы»

Цель: знакомство с базовыми понятиями объектно-ориентированного программирования и разбор этапов решения типовых задач.

Дидактические материалы, МТО: проектор, интерактивная доска, презентация, ПК с установленным IDE и редактором кода для Python.

Время на изучение материала: 45 минут.

Объектно-ориентированное программирование (ООП) – один из подходов к реализации программного кода для проецирования сущностей реального мира. Считается, что введение классов и объектов упрощает понимание кода человеком.

Поговорим про основные принципы объектно-ориентированного программирования: абстракцию, инкапсуляцию, наследование и полиморфизм. Научимся создавать классы и объекты классов на языке программирования Python. Рассмотрим, чем отличаются понятия поля, свойства, методы и атрибуты класса.

Вы наверняка слышали, что существуют два главных подхода к написанию программ:

- Процедурное программирование.

- Объектно-ориентированное программирование (оно же ООП).

Оба подхода объединены общей целью - сделать процесс программирования максимально эффективным. Это значит, что, благодаря им разработка программного обеспечения становится более простой для понимания, легко масштабируемой и содержащей минимальное количество ошибок.

По сути, любая программа представляет собой совокупность данных и операций по их обработке. Но что важнее, сами данные или операции над ними? В языках, в основе работы которых лежит принцип процедурного программирования (Basic, C, Pascal, Go), главным является код для обработки данных. При этом сами данные имеют второстепенное значение.

В основе ООП лежит простая идея, в соответствии с которой главное в программе - это данные. Именно они определяют, какие методы будут использоваться для их обработки. Т.е. данные первичны, код для обработки этих данных - вторичен.

Мир, в котором мы живем, состоит из объектов. Это деревья, солнце, горы, дома, машины, бытовая техника. Каждый из этих объектов имеет свой набор характеристик и предназначение. Несложно догадаться, что именно объектная картина реального мира легла в основу ООП. Разберем несколько ключевых понятий:

Класс - в объектно-ориентированном программировании, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.

Объект - некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определенные свойства (поля) и операции над ними (методы). Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Термины «экземпляр класса» и «объект» взаимозаменяемы.

Класс описывает множество объектов, имеющих общую структуру и обладающих одинаковым поведением. Класс - это шаблон кода, по которому создаются объекты. Т.е. сам по себе класс ничего не делает, но с его помощью можно создать объект и уже его использовать в работе.

Данные внутри класса делятся на свойства и методы. Свойства класса (они же поля) - это характеристики объекта класса.

Методы класса - это функции, с помощью которых можно оперировать данными класса.

Объект - это конкретный представитель класса.

Как вы думаете, объект класса и экземпляр класса - это одно и то же?

Рассмотрим простой пример.

Перед нами класс «Автомобиль». Если мыслить абстрактно, то он представляет собой набор чертежей и инструкций, с помощью которых можно собрать машину. При этом каждая машина, которую мы будем собирать, должна обладать рядом характеристик, которые соответствуют нашему классу. Как мы уже выяснили, данные характеристики - называются свойствами класса. А какие они могут быть в нашем примере? (цвет, объем двигателя, мощность, тип коробки передач)

Так же наш автомобиль может выполнять какие-то действия, характерные для всего класса. Эти действия, как мы теперь знаем, есть методы класса. А какие действия может выполнять автомобиль? (ехать, остановиться, заправиться, поставить на сигнализацию, включить дворники).

Начинка класса готова, теперь можно переходить к созданию объектов.

Все объекты создаются по одному шаблону, то есть на выходе обязательно будут машины, никаких велосипедов и мотоциклов. Они будут выкрашены в какой-то цвет, ехать они будут за счет наличия в них двигателя, скорость будет регулироваться с помощью коробки передач. Также объекты данного класса будут обладать одинаковыми методами -

все машины этого класса будут ездить, периодически им будет нужна заправка, а от угона они будут защищены установкой сигнализации.

Значения свойств будут различаться. Одна машина красная, другая - зеленая. У одной объем двигателя 1968 см³ и коробка-робот, а у другой - 1395 см³ и ездить владельцу придется на механике.

Теперь давайте посмотрим, как реализуется ООП в рамках языка программирования Python. Синтаксис для создания класса выглядит следующим образом:

```
class <название_класса>:
    <тело_класса>
```

А вот так компактно смотрится пример объявления класса с минимально возможным функционалом:

```
class Car:
    pass
```

Как мы видим, для задания класса используется инструкция class, далее следует имя класса Car и двоеточие. После них идет тело класса, которое в нашем случае представлено оператором pass. Данный оператор сам по себе ничего не делает. Чтобы создать объект класса, нужно воспользоваться следующим синтаксисом:

```
<имя_объекта> = <имя_класса>()
```

Вывод: Объекты класса на выходе похожие и одновременно разные. Различаются, как правило, свойства. Методы остаются одинаковыми.

Python имеет множество встроенных типов, например, int, str и так далее, которые мы можем использовать в программе. Но также Python позволяет определять собственные типы с помощью классов. Класс представляет некоторую сущность. Конкретным воплощением класса является объект.

Можно провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, рост и т.д. Человек может выполнять некоторые действия - ходить, бегать, думать и т.д. То есть это представление, которое включает набор характеристик и действий, можно назвать классом. Конкретное воплощение этого шаблона может отличаться, например, одни люди имеют одно имя, другие - другое имя. И реально существующий человек будет представлять объект этого класса.

Класс определяется с помощью ключевого слова class:

```
1 class название_класса:
2     атрибуты_класса
3     методы_класса
```

Внутри класса определяются его атрибуты, которые хранят различные характеристики класса, и методы - функции класса.

Создадим простейший класс:

```
class Person:
    pass
```

В данном случае определен класс Person, который условно представляет человека. Здесь в классе не определяется никаких методов или атрибутов. Однако в нем должно быть что-то определено, то в качестве заменителя функционала класса применяется оператор pass. Этот оператор используется, когда синтаксически необходимо определить некоторый код, однако что бы его не писать, вместо конкретного кода вставляем оператор pass.

После создания класса можно определить объекты этого класса. Например:

```
class Person:
    pass

tom = Person()      # определение объекта tom
bob = Person()      # определение объекта bob
```

После определения класса Person создаются два объекта класса Person - tom и bob. Для создания объекта применяется специальная функция - конструктор, которая называется по имени класса и которая возвращает объект класса. То есть в данном случае вызов Person() представляет вызов конструктора.

**Методы классов**

Методы класса фактически представляют функции, которые определенны внутри класса и которые отражают его поведение. Например, определим класс Person с одним методом:

```
class Person:       # определение класса Person
    def say_hello(self):
        print("Hello")

tom = Person()
tom.say_hello()    # Hello
```

Здесь определен метод say_hello(), который условно выполняет приветствие - выводит строку на консоль. При определении методов любого класса следует учитывать, что все они должны принимать в качестве первого параметра ссылку на текущий объект, который согласно условностям называется self. Через эту ссылку внутри класса мы можем обратиться к функциональности текущего объекта. Но при самом вызове метода этот параметр не учитывается.

Используя имя объекта, мы можем обратиться к его методам. Для обращения к методам применяется нотация точки - после имени объекта ставится точка и после нее идет вызов метода:

```
объект.метод([параметры метода])
```

Например, обращение к методу say_hello() для вывода приветствия на консоль:

```
tom.say_hello()     # Hello
```

Что в итоге данная программа выведет на консоль? (строку «Hello»)

Если метод должен принимать другие параметры, то они определяются после параметра self, и при вызове подобного метода для них необходимо передать значения:

```
class Person:       # определение класса Person
    def say(self, message):     # метод
        print(message)


tom = Person()
tom.say("Hello METANIT.COM")    # Hello METANIT.COM
```

Какой метод здесь определен? (метод say). Он принимает два параметра: self и message. И для второго параметра - message при вызове метода необходимо передать значение.

**self**

Через ключевое слово self можно обращаться внутри класса к функциональности текущего объекта:

```
self.атрибут    # обращение к атрибуту
self.метод      # обращение к методу
```

Например, определим два метода в классе Person:

```
class Person:

    def say(self, message):
        print(message)

    def say_hello(self):
        self.say("Hello work")  # обращаемся к выше определенн


tom = Person()
tom.say_hello()     # Hello work
```

Здесь в одном методе - say_hello() вызывается другой метод - say():

```
self.say("Hello work")
```

Поскольку метод say() принимает кроме self еще параметры (параметр message), то при вызове метода для этого параметра передается значение.

Причем при вызове метода объекта нам обязательно необходимо использовать слово self. А что будет, если мы его не используем? (столкнемся с ошибкой)

```
def say_hello(self):
    say("Hello work")  # ! Ошибка
```

**Конструкторы**

Для создания объекта класса используется конструктор. Так, выше когда мы создавали объекты класса Person, мы использовали конструктор по умолчанию, который не принимает параметров и который неявно имеют все классы:

```
tom = Person()
```

Однако мы можем явным образом определить в классах конструктор с помощью специального метода, который называется __init__() (по два прочерка с каждой стороны). К примеру, изменим класс Person, добавив в него конструктор:

```
class Person:
    # конструктор
    def __init__(self):
        print("Создание объекта Person")

    def say_hello(self):
        print("Hello")


tom = Person()      # Создание объекта Person
tom.say_hello()     # Hello
```

Итак, здесь в коде класса Person определен конструктор и метод say_hello(). В качестве первого параметра конструктор, как и методы, также принимает ссылку на текущий объект - self. Обычно конструкторы применяются для определения действий, которые должны производиться при создании объекта.

Будет производится вызов конструктора __init__() из класса Person, который выведет на консоль строку «Создание объекта Person».

**Атрибуты объекта**

Атрибуты хранят состояние объекта. Для определения и установки атрибутов внутри класса можно применять слово self. Например, определим следующий класс Person:

```
class Person:

    def __init__(self, name):
        self.name = name    # имя человека
        self.age = 1        # возраст человека


tom = Person("Tom")

# обращение к атрибутам
# получение значений
print(tom.name)     # Tom
print(tom.age)      # 1
# изменение значения
tom.age = 37
print(tom.age)      # 37
```

Теперь конструктор класса Person принимает еще один параметр - name. Через этот параметр в конструктор будет передаваться имя создаваемого человека.

Внутри конструктора устанавливаются два атрибута - name и age (условно имя и возраст человека):

```
def __init__(self, name):
    self.name = name
    self.age = 1
```

Атрибуту self.name присваивает значение переменной name. Какое значение получит атрибут age?  (значение 1).

Если мы определили в классе конструктор __init__, мы уже не сможем вызвать конструктор по умолчанию. Теперь нам надо вызывать наш определённый конструктор __init__, в который необходимо передать значение для параметра name:

```
tom = Person("Tom")
```

Далее по имени объекта мы можем обращаться к атрибутам объекта - получать и изменять их значения:

```
print(tom.name)     # получение значения атрибута name
tom.age = 37        # изменение значения атрибута age
```

В принципе нам необязательно определять атрибуты внутри класса - Python позволяет сделать это динамически вне кода:

```python
class Person:

    def __init__(self, name):
        self.name = name    # имя человека
        self.age = 1        # возраст человека


tom = Person("Tom")

tom.company = "Microsoft"
print(tom.company)  # Microsoft
```

Здесь динамически устанавливается атрибут company, который хранит место работы человека. И после установки мы также можем получить его значение. В то же время подобное определение чревато ошибками. Например, если мы попытаемся обратиться к атрибуту до его определения, то программа сгенерирует ошибку:

```python
tom = Person("Tom")
print(tom.company)  # ! Ошибка - AttributeError: Person object
```

Для обращения к атрибутам объекта внутри класса в его методах также применяется слово self:

```python
class Person:

    def __init__(self, name):
        self.name = name    # имя человека
        self.age = 1        # возраст человека

    def display_info(self):
        print(f"Name: {self.name}  Age: {self.age}")


tom = Person("Tom")
tom.display_info()      # Name: Tom  Age: 1
```

Здесь определяется метод display_info(), который выводит информацию на консоль. И для обращения в методе к атрибутам объекта применяется слово self: self.name и self.age.

Если создать два объекта класса Person: tom и bob, то мы получим следующий консольный вывод:

```
Name: Tom  Age: 37
Name: Bob  Age: 41
```

ООП - это способ, позволяющий программисту создавать собственные типы переменных под названием класс. Так вот теперь вы сможете создавать собственные типы данных.

Класс может объединять в себе много переменных. Например, мы создали класс «человек», чтобы объединить для каждого человека в вашей группе данные о его росте, весе, возрасте и т.д.

## ПРИЛОЖЕНИЕ № 2

### Практическая работа по теме:

### «Разработка и управление коллекциями объектов: классы и массивы»

Цель работы: закрепление знаний, умений, навыков работы с целочисленными и строковыми типами данных, массивами и классами.

Дидактические материалы, МТО: задачи на языке программирования Python, проектор, интерактивная доска, ПК с установленным IDE и редактором кода для Python.

На выполнение практической работы отводится - 50 минут.

**Создать коллекцию объектов на языке программирования Python согласно следующей структуре:**

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 40 $м^2$.

**Задание**

Практическую работу следует выполнять в следующем порядке:

**Класс Human**

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.

**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__()

3. Создайте метод final_price(), который принимает в качестве параметра.

**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 40 м².

**Тесты**

1. Вызовите справочный метод default_info() для класса Human().

2. Создайте объект класса Human.

3. Выведите справочную информацию о созданном объекте (вызовите метод info()).

4. Создайте объект класса SmallHouse.

5. Попробуйте купить созданный дом, убедитесь в получении предупреждения.

6. Поправьте финансовое положение объекта - вызовите метод earn_money().

7. Снова попробуйте купить дом.

8. Посмотрите, как изменилось состояние объекта класса Human.

**ПРИЛОЖЕНИЕ № 3**

**Чек-лист (диагностика)**

**ЧЕК-ЛИСТ**
**Оценка результатов работы на занятии**

ФИО_____

Группа _____ Дата «_____» _____ 2023

Тема занятия: Разработка и управление коллекциями объектов: классы и массивы.

| Оцени свою работу на занятии | Я справился легко и свободно | Я справился, но с ошибками и затруднениями | Я не справился |
|---|---|---|---|
| Дать определение понятий объектно-ориентированного программирования – «классы» и «объекты» | | | |
| Назвать отличия свойств от методов объекта | | | |
| Воспроизводить алгоритм создания классов | | | |
| Создать методы класса | | | |
| Работа с возвращаемыми данными от функций | | | |
| Создание абстрактного класса | | | |
| Работа с унаследованными свойствами и функциями от производного класса | | | |
| Переопределение виртуальный свойств и функций | | | |
| Добавлять статические свойства объектов | | | |