МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Индустриальный институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования «Югорский государственный университет»
(ИндИ (филиал) ФГБОУ ВО «ЮГУ»)

Лекции

ОП.04. Основы алгоритмизации и программирования

09.02.07 информационные системы и программирование
(профессионалитет)

I курс

Нефтеюганск
2024

РАССМОТРЕНО
Предметной цикловой
комиссией МиЕНД
Протокол № 5 от 18.01.2024г.
_____Е.С. Игнатенко

УТВЕРЖДЕНО
Заседанием методсовета
Протокол № 4 от 08.02.2024г.
Старший методист
_____Г.Р.Давлетбаева

Лекции по учебной дисциплине ОП.04. Основы алгоритмизации и программирования разработаны на основании рабочей программы учебной дисциплины ОП.04. Основы алгоритмизации и программирования по специальности 09.02.07 среднего профессионального образования

Организация-разработчик: Индустриальный институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Югорский государственный университет»

Разработчик: Игнатенко Е.С. – преподаватель ИндИ (филиала) ФГБОУ ВО «ЮГУ»

Содержание

Пояснительная записка

Лекции по учебной дисциплине ОП.04. Основы алгоритмизации и программирования предназначены для освоения программы подготовки специалистов среднего звена (ППССЗ) на базе основного общего образования при подготовке специалистов технического профиля с получением среднего общего образования и реализуется на 1 курсе очной формы обучения.

Лекции по учебной дисциплине «Основы алгоритмизации и программирования» включают следующие разделы:

- Общие сведения об информации и информационных технологиях
- Знакомство и работа с офисным по

Лекции учебной дисциплине ОП.04. Основы алгоритмизации и программирования учитывают специфику осваиваемых специальностей СПО, предполагают углубленное изучение отдельных тем, различных видов самостоятельной работы, направленных на подготовку обучающихся к профессиональной деятельности с использованием икт.

В результате освоения разделов ОП.04. Основы алгоритмизации и программирования обучающийся должен:

Уметь:

- Выполнять оптимизацию и рефакторинг программного кода.
- Работать с системой контроля версий.
- Использовать выбранную систему контроля версий.
- Использовать методы для получения кода с заданной функциональностью и степенью качества.
- Анализировать проектную и техническую документацию.
- Организовывать постобработку данных.
- Приемы работы в системах контроля версий.
- Выявлять ошибки в системных компонентах на основе спецификаций.

Знать:

- Способы оптимизации и приемы рефакторинга.
- Инструментальные средства анализа алгоритма.
- Методы организации рефакторинга и оптимизации кода.
- Принципы работы с системой контроля версий.
- Модели процесса разработки программного обеспечения.
- Основные принципы процесса разработки программного обеспечения.
- Основные подходы к интегрированию программных модулей.
- Основы верификации и аттестации программного обеспечения.
- Стандарты качества программной документации.
- Основы организации инспектирования и верификации.
- Встроенные и основные специализированные инструменты анализа качества программных продуктов.
- Методы организации работы в команде разработчиков.

В результате освоения учебной дисциплины обучающийся должен овладеть **общими и профессиональными компетенциями**, включающими в себя способность:

ОК 1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам

ОК 2. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности

ОК 4. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 5. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста.

ОК 9. Использовать информационные технологии в профессиональной деятельности

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языках.

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.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования.

Раздел 1. Введение в программирование

Тема 1.1. Основы алгоритмизации

План:

1. Алгоритмы и способы их описания
2. Правила построения блок-схем
3. Этапы решения задач на ЭВМ

Алгоритмы и способы их описания

Алгоритм – строгая система правил или инструкций для исполнителя, определяющая некоторую последовательность действий, которая после конечного числа шагов приводит к достижению искомого результата.

Алгоритм - это последовательность действий, приводящих к требуемому результату.

Таким образом, при разработке алгоритма решения задачи математическая формулировка преобразуется в процедуру решения, представляющую собой последовательность арифметических действий и логических связей между ними. При этом алгоритм обладает следующими свойствами:

1) Дискретность - процесс преобразования данных, т.е. на каждом шаге алгоритма выполняется очередная одна операция;

2) Результативность - алгоритм должен давать некоторый результат;

3) Конечность - алгоритм должен давать результат за конечное число шагов;

4) Определенность - все предписания алгоритма должны быть однозначны, понятны пользователю;

5) Массовость - алгоритм должен давать решения для целой группы задач из некоторого класса, отличающихся исходными данными;

Действия в алгоритме выполняются в порядке их записи. Нельзя менять местами никакие два действия алгоритма, а так же нельзя не закончив одного действия переходить к следующему.

Для записи алгоритмов используются специальные языки:

1. Естественный язык (словесная запись). Запись алгоритма происходит с помощью словесных слов:

если условие то действие1 иначе действие2

2. Формулы.

3. Псевдокод.

4. Структурограммы. Используется структурированная словесная запись:

Рисунок 1

5. Синтаксические диаграммы.

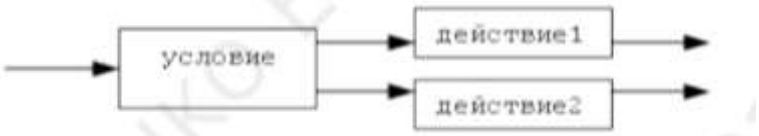


Рисунок 2

6. Графический (язык блок-схем).

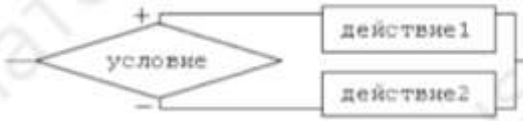
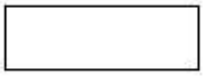
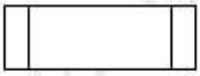


Рисунок 3

Таблица 1 – Составление блок-схем

Название	Рисунок	Выполняемая функция
1. Блок вычислений		Выполняет вычислительное действие или группу действий
2. Логический блок		Выбор направления выполнения алгоритма в зависимости от условия
3. Блоки ввода/вывода		Ввод или вывод данных вне зависимости от физического носителя
		Вывод данных на печатающее устройство
4. Начало/конец		Начало или конец программы, вход или выход в подпрограмму
5. Предопределенный процесс		Вычисления по стандартной или пользовательской подпрограмме
6. Блок модификации		Выполнение действий, изменяющих пункты алгоритма
7. Соединитель		Указание связи между прерванными линиями в пределах одной страницы
8.Межстраничный соединитель		Указание связи между частями схемы, расположенной на разных страницах

Правила построения блок-схем:

1. Блок-схема выстраивается в одном направлении либо сверху вниз, либо слева направо
2. Все повороты соединительных линий выполняются под углом 90 градусов

Общими правилами при проектировании визуальных алгоритмов (блок-схем) являются следующие:

1. В начале алгоритма должны быть блоки ввода значений входных данных.
2. После ввода значений входных данных могут следовать блоки обработки и блоки условия.
3. В конце алгоритма должны располагаться блоки вывода значений выходных данных.
4. В алгоритме должен быть только один блок начала и один блок окончания.

Этапы решения задач на ЭВМ

1. Постановка задачи.

Прежде чем понять задачу, следует уточнить ее основные характеристики, сформулировать цель решения задачи, подробно описать ее содержание, провести анализ характера и сущности всех известных и неизвестных данных, определить условия, при которых задача может быть решена. При постановке задачи выясняется конечная цель и вырабатывается общий подход к решению задачи. Выясняется, сколько решений имеет задача и имеет ли их вообще. Изучаются общие свойства рассматриваемого физического явления или объекта, анализируются возможности данной системы программирования. Исходные данные должны быть полными, т.е. содержать данные, необходимые и достаточные для решения задачи. Если данные неполные, необходимо приложить дополнительные усилия для сбора дополнительных сведений; эта ситуация может также возникнуть на последующих этапах при выборе метода решения.

Различают исходные данные трех видов: постоянные, условно-постоянные и переменные.

Постоянные исходные данные - это данные, которые сохраняют свои значения в процессе решения задачи (математические константы, координаты неподвижных объектов) и не зависят от внешних факторов.

Условно-постоянные данные - это данные, которые могут иногда изменять свои значения; причем эти изменения не зависят от процесса решения задачи, а определяются внешними факторами (величина подоходного налога, курс валют, количество дней в году).

Переменные данные - это данные, которые изменяют свои значения в процессе решения задачи.

На этом этапе важно не только классифицировать данные по отношению к процессу решения, но определить их наименование, тип, структуру и ограничения, накладываемые на значения. Желательно также определить допустимые и недопустимые операции по отношению к различным типам исходных данных

2. Формализация (математическая постановка).

После проведения анализа постановки задачи, выявления данных, их структуры и отношений между ними можно приступить к построению формальной модели. Это наиболее важный этап в процессе решения задачи.

Модель - это упрощенное представление о реальном объекте, процессе или явлении.

Моделирование - построение моделей для исследования и изучения моделируемого объекта, процесса, явления с целью получения новой информации при решении конкретных задач.

Для описания модели предметной области решаемой задачи необходимо выбрать некоторую формальную систему. Обычно, исходя из постановки задачи, можно сразу определить один или несколько видов моделей, подходящих для описания и моделирования решения вашей задачи: математические, геометрические, структурные, логические и др. Наиболее распространенными и хорошо изученными являются математические модели, описывающие зависимости между данными числового типа. Например, в качестве математической модели звезды можно использовать систему уравнений, описывающих процессы, происходящие в недрах звезды. Математической моделью другого рода являются математические соотношения, позволяющие рассчитать оптимальный план работы предприятия. К основным достоинствам математических моделей, безусловно, относятся хорошо изученные и широко применяемые математические методы решения большого класса задач, что значительно облегчает формирование основной идеи и выбор методов решения задачи.

Приступая к разработке модели, следует попытаться решить задачу для конкретных входных данных, затем обобщить полученное решение на основе его анализа для любых значений входных данных.

На этом этапе все объекты задачи описываются на языке математики, выбирается форма хранения данных, составляются все необходимые формулы.

Если задача четко поставлена, то для нее несложно разработать математическую модель. Выбор модели существенно влияет на остальные этапы в процессе решения. Математическая формулировка и последующий выбор метода решения являются основой для определения последовательности действий, приводящих к получению искомого результата. В зависимости от содержания задачи построение ее модели может быть областью исследований таких дисциплин, как исследование операций, методы оптимизации, математическая статистика, численный анализ, теория информации и др.

3. Выбор (или разработка) метода решения.

Выбор существующего или разработка нового метода решения (очень важен и, в то же время, личностный этап).

4. Разработка алгоритма.

На этом этапе метод решения записывается применительно к данной задаче на одном из алгоритмических языков.

Наиболее распространенными методами разработки алгоритмов являются: метод частных целей, метод подъема и эвристический алгоритм.

Для разработки алгоритма методом частных целей необходимо определить варианты возможностей решения задачи:

- Можно ли решить хотя бы часть задачи, игнорируя некоторые условия?
- Можно ли решить задачу для частных случаев?
- Есть ли что-то, что недостаточно понятно?
- Встречалась ли похожая задача?
- Можно ли видоизменить ее для решения данной задачи?

Для метода подъема необходимо вначале сделать предположение, что имеется некоторое начальное состояние решения задачи. Затем следует определить, насколько возможно движение дальше – от начального к наилучшему решению.

При разработке эвристического алгоритма необходимо помнить, что такой алгоритм обычно помогает найти хорошее, но не обязательно оптимальное решение.

Общий подход заключается в перечислении всех требований к точному решению с указанием, для каких из них возможен компромисс, а какие непременно должны быть выполнены.

5. Составление программы.

Решение задачи переводится на язык, понятный машине.

6. Отладка программы.

7. Вычисление и обработка результатов.

Тема 1.2. Понятие системы программирования

План:

1. Понятие о языках программирования
2. Основные этапы исторического развития языков программирования
3. Области применения языков программирования
4. Понятие системы и среды программирования
5. Классификация систем программирования
6. Интегрированная среда программирования

Понятие о языках программирования

Программу можно представить как набор последовательных команд, то есть алгоритм, для объекта, то есть исполнителя, который должен их выполнить для достижения определенной цели.

Так можно условно запрограммировать человека, составив для него к примеру инструкцию "как приготовить оладьи", а он начнет четко ей следовать. При этом инструкция, она же программа, для человека будет написана на так называемом естественном языке, например, русском или английском.

Все же программируют не людей, а вычислительные машины, используя при этом специальные языки. Необходимость в особых языках связана с тем, что машины не в состоянии "понимать" наши, то есть человеческие естественные для нас языки. Инструкции для машин пишут на языках программирования, которые характеризуются формальностью, то есть синтаксической однозначностью (например, в них нельзя менять местами определенные слова) и ограниченностью (имеют строго определенный набор слов и символов).

Язык программирования — формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими при различных обстоятельствах.

Основные этапы исторического развития языков программирования

Первые программы писались на машинном языке, так как для ЭВМ того времени еще не существовало развитого программного обеспечения, а машинный язык – это единственный способ взаимодействия с аппаратным обеспечением компьютера, так называемым "железом".

Каждую команду машинного языка непосредственно выполняет то или иное электронное устройство. Данные и команды записывали в цифровом виде, например, в шестнадцатеричной или двоичной системах счисления. Человеку воспринимать

программу на таком языке сложно. Кроме того, даже небольшая программа состояла из множества строк кода. Ситуация осложнялась еще и тем, что каждая вычислительная машина понимает лишь свой машинный язык.

Людям, в отличие от машин, более понятны слова, чем наборы цифр. Стремление человека оперировать словами, а не цифрами привело к появлению ассемблеров. Это языки, в которых вместо численного обозначения команд и областей памяти используются словесно-буквенные.

При этом появляется проблема: машина не в состоянии понимать слова. Необходим какой-нибудь переводчик на ее родной машинный язык. Поэтому, начиная со времен ассемблеров, под каждый язык программирования создаются трансляторы – специальные программы, преобразующие программный код с языка программирования в машинный код. Ассемблеры на сегодняшний день продолжают использоваться. В системном программировании с их помощью создаются низкоуровневые интерфейсы операционных систем, компоненты драйверов.

После ассемблеров наступил расцвет языков так называемого высокого уровня. Для них потребовалось разрабатывать более сложные трансляторы, так как языки высокого уровня куда больше удобны для человека, чем для вычислительной машины.

В отличие от ассемблеров, которые остаются привязанными к своим типам машин, языки высоко уровня обладают переносимостью. Это значит, что, написав один раз программу, программист без последующего редактирования может выполнить ее на любом компьютере, если на нем установлен соответствующий транслятор. Программа-транслятор для данной ЭВМ при трансляции исходного кода сама адаптирует его под эту ЭВМ.

Следующим значимым шагом было появление объектно-ориентированных языков, что в первую очередь связано с усложнением разрабатываемых программ. С помощью таких языков программист как бы управляет виртуальными объектами. Мыслить в рамках объектов-сущностей, описывать их взаимодействие, обобщать объекты в классы и устанавливать между ними наследственные связи, – все это делает программу по-своему похожей на реальный мир, на то, как его воспринимает человек.

На сегодняшний день в большинстве случаев реализация крупных проектов осуществляется с помощью объектно-ориентированных возможностей языков. Хотя существуют и другие современные парадигмы программирования, поддерживаемые другими или теми же языками.

Области применения языков программирования

В настоящее время языки программирования применяются в самых различных областях человеческой деятельности, таких как:

- научные вычисления (языки C++, FORTRAN, Java);
- системное программирование (языки C++, Java);
- обработка информации (языки C++, COBOL, Java);
- искусственный интеллект (LISP, Prolog);
- издательская деятельность (Postscript, TeX);
- удаленная обработка информации (Perl, PHP, Java, C++);
- описание документов (HTML, XML).

С течением времени одни языки развивались, приобретали новые черты и остались востребованы, другие утратили свою актуальность и сегодня представляют в

лучшем случае чисто теоретический интерес. В значительной степени это связано с такими факторами, как:

- наличие среды программирования, поддерживающей разработку приложений на конкретном языке программирования;
- удобство сопровождения и тестирования программ;
- стоимость разработки с применением конкретного языка программирования;
- четкость и ортогональность конструкций языка;
- применение объектно-ориентированного подхода.

Понятие системы и среды программирования

Системы программирования - это комплекс инструментальных программных средств, предназначенный для разработки программ на одном или последующем этапе развития.

Система программирования — это система для разработки новых программ на конкретном языке программирования.

Специалисты с помощью сервисных возможностей систем программирования могут разрабатывать собственные компьютерные программы. При этом компьютерная программа состоит из совокупности указаний автоматизированной вычислительной системы, в результате выполнения которой получается требуемый результат.

Наиболее полное определение системы программирования и ее составляющих представлено в документе ГОСТ 19781-90. Согласно ему: «Система программирования — система, образуемая языком программирования, компиляторами или интерпретаторами программ, представленных на этом языке, соответствующей документацией, а также вспомогательными средствами для подготовки программ к форме, пригодной для выполнения».

Системы программирования позволяют программистам заниматься разработкой компьютерных программ. Данная задача значительно облегчается совершенствованием систем программирования, в которых постоянно расширяются пользовательские возможности, создается удобная среда для работы и оптимизируется процесс разработки программ.

Среда программирования – это совокупность программ, обеспечивающих технологический цикл разработки программ: анализ, спецификация, проектирование, кодирование (редактирование, компиляция, компоновка), тестирование, отладка.

Базовые компоненты среды:

1. Редактор – средство создания и изменения исходных файлов с текстом программы.
2. Компилятор – транслирует исходный файл в объектный файл, содержащий команды в машинном коде для конкретного компьютера.
3. Компоновщик (редактор связей) – собирает объектные файлы программы и формирует исполняемый файл (разрешая внешние ссылки между объектными файлами).
4. Отладчик – средство управления выполнением исполняемого файла на уровне отдельных операторов программы для диагностики ошибок.
5. Библиотекарь – средство ведения совокупностей объектных файлов (библиотек).
6. Профилировщик – средство измерения времени выполнения программных компонент для последующей оптимизации критических компонентов.

7. Загрузчик – копирует исполняемый файл с диска в память и осуществляет его запуск.

Классификация систем программирования

По набору входных обязательных языковых систем программирования одно- и многоязычные. Отличительная черта многоязыковых систем состоит в том, что отдельные части программ могут составляться в зависимости от различных различий и со специальной помощью обрабатывающих программ объединять их в готовую для исполнения программу ЭВМ.

По степени вероятности, формализации языка и целевому назначению имеют место системы программирования машинно-ориентированных входных и машинно-независимых систем. Машинно-ориентированные системы программирования имеют повышенный входной язык, набор операций и тяжелые состояния сильно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.).

Интегрированная среда программирования.

Microsoft Visual Studio

Visual Studio — интегрированная среда разработки C++, которая позволяет разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms. Она также подходит для создания веб-сайтов, веб-приложений и веб-служб для всех поддерживаемых платформ: Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone, .NET Compact Framework и Silverlight.

Достоинства:

1. Есть бесплатная версия Visual Studio Community.
2. Встроенный интерфейс командной строки.
3. API для подключения дополнительных инструментов отладки.
4. Полный набор инструментов разработчика для создания и клонирования Git-репозиториев, управления ветвями и разрешения конфликтов слияния прямо в интегрированной среде разработки C++.
5. Большой набор дополнений для расширения базовой функциональности.

Недостатки:

1. Высокая стоимость платных версий Professional и Enterprise (от 45 долларов в месяц).
2. Высокие требования к «железу».
3. Нет версии для Linux.

Microsoft Visual Studio платная версия, но есть бесплатная версия Community Edition с урезанным функционалом, нам ее достаточно.

Устанавливать нужно только "Классические приложения .NET"

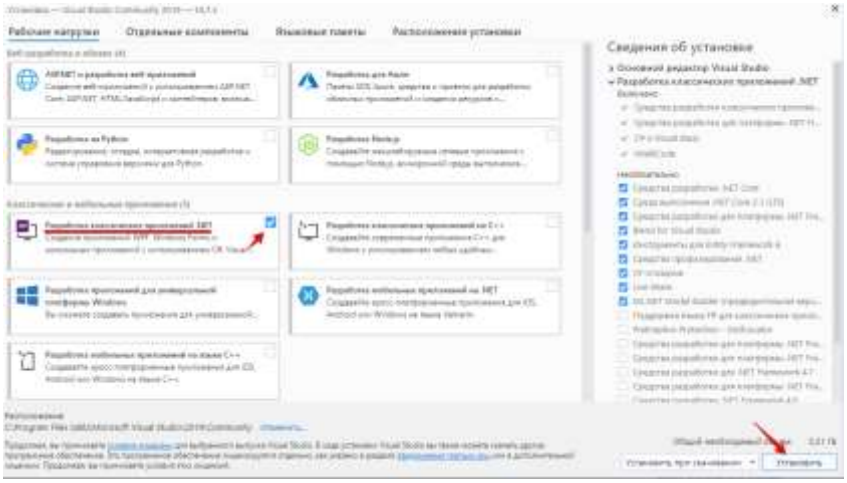


Рисунок 1

IDE Выглядит примерно так:

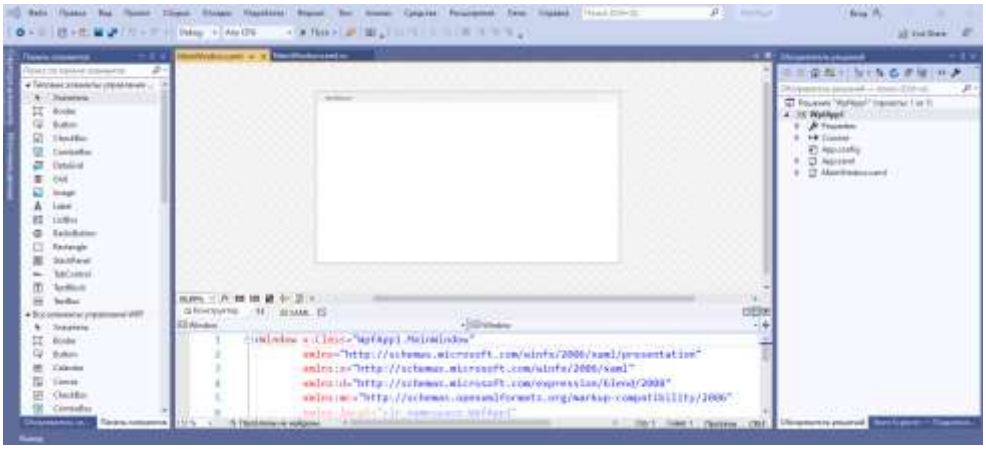


Рисунок 2

Eclipse CDT

Eclipse — свободная интегрированная среда разработки модульных кроссплатформенных приложений, которая приобрела большую популярность среди разработчиков на Java, однако в данной статье мы рассмотрим выпуск Eclipse CDT (C/C++ Development Tooling). Данная среда является отличным выбором, так как Eclipse CDT не только обладает всеми необходимыми инструментами, но и бесплатна и работает под различными операционными системами: Windows/Linux/macOS.

Достоинства:

1. Бесплатное использование.
2. Автозавершение и другие возможности, которые помогают быстрее писать код.
3. Большой набор плагинов для расширения функциональности.
4. Развитое сообщество разработчиков, подробная документация.
5. Встроенное юнит-тестирование, оптимизация тестов.
6. Настраиваемый графический интерфейс.

Недостатки:

1. Медленный запуск, потребление большого объёма памяти.
2. Проблемы с обратной совместимостью.
3. Возможны конфликты плагинов.

Sublime Text

Sublime Text — проприетарный текстовый редактор, написанный на C++ и Python. Разработан в 2008 году. У этого текстового редактора кода достаточно большое сообщество, поэтому нет проблем с поиском дополнений и обучающих материалов.

Достоинства:

1. Можно пользоваться бесплатно.
2. Не предъявляет высоких требований к «железу».
3. Встроенная поддержка нескольких десятков языков программирования.
4. Автозавершение и подсветка синтаксиса в текстовом редакторе кода.
5. Большой выбор плагинов, написанных на Python.

Недостатки:

1. Не такой хороший отладчик, как в Visual Studio и других IDE.
2. Нет автосохранения файлов.
3. Постоянно предлагает купить платную версию.

NetBeans

NetBeans — еще одна бесплатная интегрированная среда разработки. Позволяет создавать приложения на языках программирования Java, Python, PHP, JavaScript, C, C++ и ряде других. У этой интегрированной среды программирования C++ есть дистрибутива для платформ Microsoft Windows, Linux, FreeBSD, macOS , OpenSolaris и Solaris, а для всех остальных платформ есть возможность собрать NetBeans самостоятельно из исходников.

Достоинства:

1. Бесплатная интегрированная среда разработки C++ .
2. Кроссплатформенная поддержка.
3. Большой выбор плагинов.
4. Автозавершение кода, инструменты для рефакторинга.
5. Развитое сообщество разработчиков.

Недостатки:

1. Медленный запуск.
2. Проблемы с собственным кэшем при сборке готовых программ.
3. Для установки требуется JDK.

Qt Creator

Qt Creator — интегрированная среда разработки C++, доступная на Windows, Linux и macOS. Предоставляет бесплатную версию, которой можно пользоваться в течение 1 месяца. Предлагает полный набор инструментов разработчика, предназначенных для создания и развёртывания приложений.

Достоинства:

1. Поддерживает отладку, профилирование, автозавершение кода и рефакторинг.
2. Возможность компиляции проектов для разных ОС.

Недостатки:

1. Большой вес приложений.
2. Не всегда работает автозавершение кода.
3. Дорогая платная версия.
4. Нужна регистрация для скачивания бесплатной версии.

CLion

CLion — кроссплатформенная среда программирования на C++ от компании JetBrains. Включает в себя современные стандарты C ++, libc ++ и Boost. Поддерживает также другие языки программирования — Kotlin, Python, Rust и т.д. — «из коробки» или с помощью плагинов.

Достоинства:

1. Удобные механизмы отладки приложений.
2. Автозавершение кода.
3. Поддержка VIM.

Недостатки:

1. Нет бесплатной версии — только демо на 30 дней.
2. Нет встроенного компилятора.
3. Возникают проблемы с установкой компилятора.

CodeLite

CodeLite распространяется бесплатно и работает во множестве операционных систем: Windows 7/8/8.1/10, Debian, Ubuntu, Fedora, OpenSUSE, ArchLinux и macOS. Интерфейс прост и интуитивно понятен, что делает его весьма хорошим выбором для новичков. Также следует отметить, что последние версии этой интегрированной среды разработки C++ поддерживают проекты на PHP и Node.js.

Достоинства:

1. Мощный инструмента автозавершения кода, основанный на собственном синтаксическом анализаторе.
2. Плагины для работы с Git и SVN.
3. Встроенный отладчик.

Недостатки:

1. Сложный интерфейс.

Code::Blocks

Завершает нашу подборку бесплатная среда разработки Code::Blocks. Она позволит писать не только на C/C++, но и обеспечит поддержкой таких языков программирования, как Fortran и D (с некоторыми ограничениями). У набора инструментов разработчика есть возможность для расширения за счёт установки плагинов. У этой среды программирования на C++ есть версии под Windows, macOS и Linux, однако существует возможность установить её на любую Unix-подобную систему при помощи сборки исходников.

Достоинства:

1. Бесплатная среда разработки C++.
2. Автозавершение кода.
3. Встроенный отладчик.
4. Большой выбор плагинов для расширения функциональности.

Недостатки:

1. Не подходит для разработки больших проектов.

Вопросы и задания для самоконтроля по разделу 1

1. Дайте определение алгоритма.
2. Можно ли считать алгоритмом: (a) правила правописания; (b) законы физики; (c) математические формулы; (d) статьи уголовного кодекса. Ответы обоснуйте.
3. В повседневной жизни существует множество синонимов понятия «алгоритм». Что из перечисленного ниже нельзя назвать алгоритмом? Ответ обоснуйте.
4. Какими способами можно описать алгоритм решения задачи?
5. Какую форму записи алгоритма называют блок-схемой? Приведите пример.
6. В чем суть таких свойств алгоритма как «результативность» и «массовость»? Обладает ли требованиям массовости и результативности следующая последовательность действий при вычислении значения функции y = (a +b)/c:
   Шаг 1. Ввести значения переменных a, b и c.
   Шаг 2. Вычислить значение функции y = (a +b)/c.
   Шаг 3. Напечатать значение результата y.
   Шаг 4. Прекратить вычисления.
7. Какой алгоритм называют линейным? Приведите пример.
8. Какой алгоритм называется циклическим? Приведите пример.
9. Дайте понятие о языках программирования
10. Перечислите основные этапы исторического развития языков программирования
11. Опишите области применения языков программирования
12. Дайте понятие системы и среды программирования
13. Классификация систем программирования
14. Опишите следующие программы: Microsoft Visual Studio, Eclipse CDT, Sublime Text, NetBeans, Qt Creator, CLion, CodeLite, Code::Blocks

Решение задач

Задача 1.

Даны координаты вершин треугольника ABC. Найти его площадь. Составьте блок-схему алгоритма решения поставленной задачи.

Задача 2.

Вычислить значение функции y=ax+b

Задача 3.

Дана величина A, выражающая объем информации в байтах. Перевести A в более крупные единицы измерения информации. Составьте блок-схему алгоритма решения поставленной задачи.

Задача 4.

Дан алгоритм в виде блок-схемы.

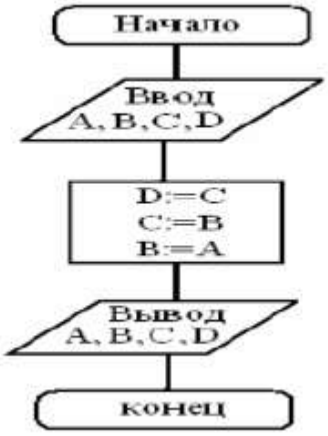


Рисунок 1

Найти A, B, C, D, если изначально:

а) A=0, B=0, C=5, D=10;

б) A=0, B=5, C=0, D=10;

в) A=10, B=20, C=6, D=4;

г) A=10, B=10, C=4, D=0.

Результат работы алгоритма определяется с помощью трассировочных таблиц (а, б, в, г):

а) A=0, B=0, C=5, D=10.

Таблица 1

Шаг		1
Исходные значения	A	0
	B	0
	C	5
	D	10
Результат выполнения	A	0
	B	0
	C	0
	D	5
Вывод значений		0, 0, 0, 5

Выполнить расчеты для остальных значений, составив аналогичные таблицы

Задача 5.

Вычислить путь, пройденный лодкой, если ее скорость в стоячей воде v км/ч, скорость течения реки v1 км/ч, время движения по озеру t1 ч, а против течения реки – t2 ч. Составьте блок-схему алгоритма решения поставленной задачи.

Задача 6.

Вычислите значение функции Y при X=2, используя блок-схему алгоритма. Схема алгоритма: X = 2, Z = 8 * X, Z= корень из Z, Z= Z – 1, Y = 3 * X, Y = Y / Z

Задача 7.

Даны длины двух катетов прямоугольного треугольника. Определить периметр этого треугольника.

Задача 8.

Даны два действительных числа a и b. Вычислить их сумму, разность, произведение и частное.

Задача 9.

Вычислить высоту правильного треугольника h при заданной стороне правильного треугольника a.

Задача 10.

Дан размер файла в байтах. Используя операцию деления нацело, найти количество полных килобайтов, которые занимает данный файл.

(В C++ деление: / - целая часть, % - остаток от деления)

Задача 11.

Вычислить значение выражения по формуле (каждая формула отдельный алгоритм)

$$R = 3t^2 + 3l^5 + 4.9$$

$$K = \ln(p^2 + y^3) + e^p$$

$$G = n(y + 3.5) + \sqrt{y}$$

$$S = \sqrt{cos4y^2 + 7.151}$$

$$N = 3y^2 + \sqrt{y + 1}$$

$$N = 3y^2 + \sqrt{y^3 + 1}$$

Задача 12.

Дана блок-схема алгоритма

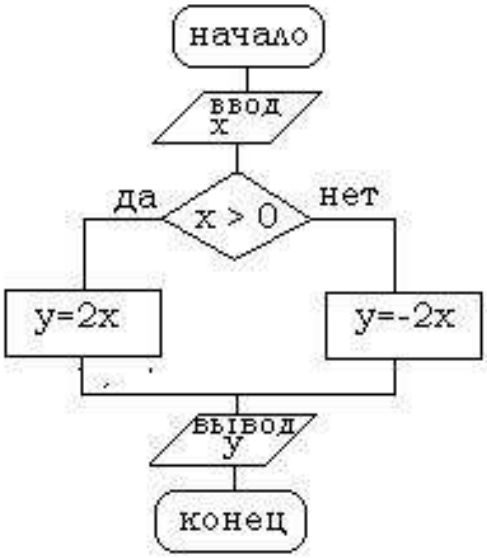


Рисунок 2

Определить результат выполнения алгоритма при определённых значениях исходных данных

Например, при x=-6 или x=0 или x=7

1. Ввод: x=
   Проверка условия x>0 □
   Вывод: y=
2. Ввод: x=
   Проверка условия x>0
   Вывод: y=
3. Ввод: x=
   Проверка условия x>0
   Вывод: y=

Задача 13.

Дана блок-схема алгоритма

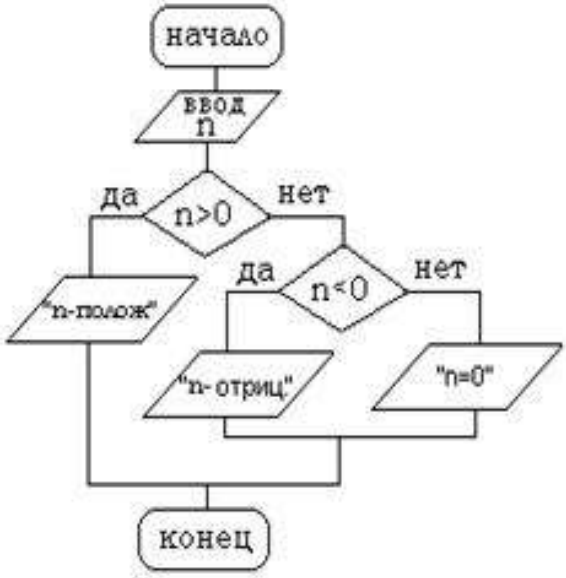


Рисунок 3

Определить результат выполнения алгоритма при определённых значениях исходных данных

Например, при n=15 или n=0 или n=-7

Задача 14.

Сравните два числа, большее из них уменьшите в 2 раза, а меньшее уменьшите на 2.

Задача 15.

Даны три числа a, b, c. Составить алгоритм, определяющий большее из трех чисел.

Задача 16.

Вычислить значение функции $y = \begin{cases} \sqrt{x}, & x \geq 0 \\ x^2, & x < 0 \end{cases}$, если: а) x=0; б) x=1; в) x= – 5.

Задача 17. Реализован некоторый алгоритм в виде блок- схемы. Что получится на выходе блок-схемы, если: а) x=0, y=1; б) x=2, y=4; в) x=6,y=0?

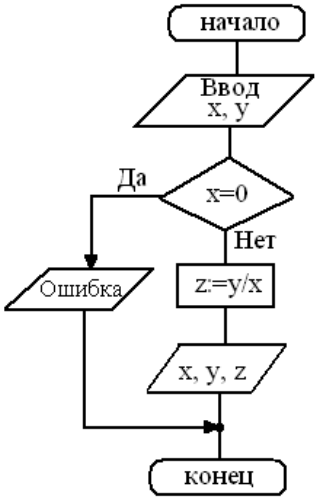


Рисунок 4

Задача 18.

Дана блок-схема алгоритма

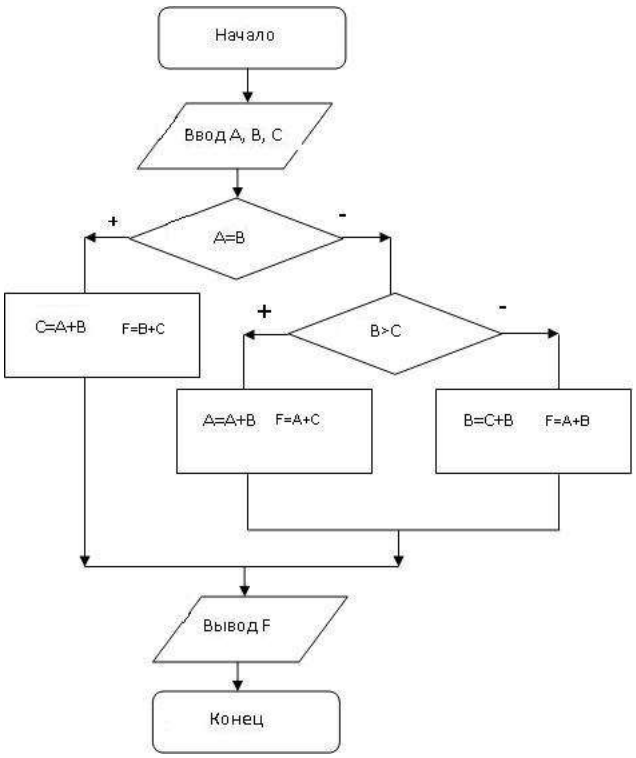


Рисунок 5

Определить результат выполнения алгоритма при определённых значениях исходных данных

При а) A=7; B=8; C=9; б) A=6; B=6; C=-10; с) A=6; B=10; C=-10

Задача 19.

Вычислить значение выражений. Если x >0

$$\frac{2x-1}{3x}$$

Задача 20.

Вычислить значение выражений. Если b >0, d>0

$$\frac{a}{b}-\frac{c}{d}$$

Задача 21.

Дана блок-схема алгоритма

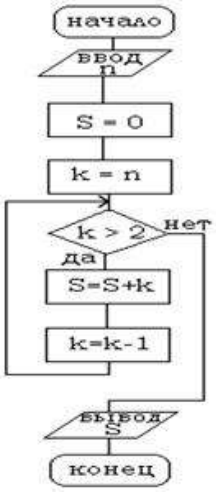


Рисунок 6

Определить результат выполнения алгоритма при определённых значениях исходных данных

При а) n=4; б) n=1

Задача 22.

По блок-схеме сформулируйте условие задачи.

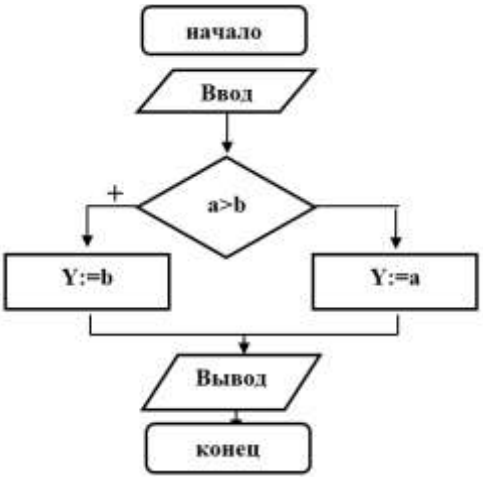


Рисунок 7

Задача 23.

Ввести с клавиатуры два числа. Определить, что больше, сумма квадратов или квадрат суммы этих чисел. Ответ вывести в виде сообщения.

Задача 24.

Ввести с клавиатуры значение трех сторон треугольника, и определить является ли он прямоугольным. Ответ вывести в виде сообщения.

Задача 25.

Возвести числа A в целую степень N

Задача 26.

Вычислить факториал заданного целого числа. Факториал числа N вычисляется по следующей формуле: N!=1*2*3*…**N

Раздел 2. Реализация алгоритмов и программирование на языке Python

Тема 2.1. Введение в программирование на Python

План:

1. Структура программы
2. Ввод данных
3. Вывод данных
4. PyCharm
5. Python Shell
6. Создание проекта в Python Shell
7. Синтаксис Python на примере функции print()
8. Типы данных

Python – это объектно-ориентированный, интерпретируемый, переносимый язык сверхвысокого уровня. Программирование на Python позволяет получать быстро и качественно необходимые программные модули.

Язык программирования Python был задуман нидерландским программистом Гвидо ван Россумом. Разработка начата в 1989 году.

Среда разработки по умолчанию работает в интерактивном режиме, то есть команды в ней запускаются на выполнение сразу после ввода.

Любая Python-программа состоит из последовательности допустимых символов, записанных в определенном порядке и по определенным правилам.

Структура программы

Модуль – это ряд связанных между собой операций.

Инструкции – это указания компьютеру, определяющие, какие операции выполнит компьютер над данными.

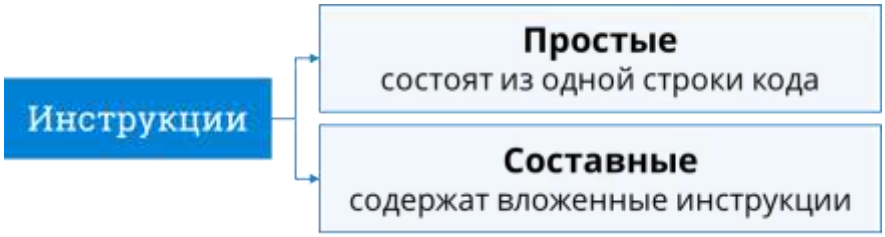


Рисунок 1

Рисунок 2

Ввод данных

Функция input возвращает значение, введённое пользователем с клавиатуры.

Запись функции: <имя переменной> = input()

Пример использования:

```
a = input()
print(a)
```

В скобках функции можно указать сообщение - комментарий к вводимым данным:

```
a = input ("Введите количество: ")
```

Команда input() по умолчанию воспринимает входные данные как строку символов. Поэтому, чтобы ввести целочисленное значение, следует указать тип данных int():

```
a = int (input())
```

Для ввода вещественных чисел применяется команда

```
a=float(input())
```

Вывод данных

Инструкция print выводит данные из оперативной памяти компьютера на экран.

Запись функции: print (<список данных>)

Пример использования:

```
a = 1
b = 2
print(a)
print(a + b)
print('сумма = ', a + b)
```

Дружественность пользовательского интерфейса

Ввод и вывод данных в программе должны сопровождаться различными поясняющими сообщениями и подсказками, так чтобы пользователю было понятно, что ему нужно сделать и какие данные он получит в результате.

При использовании инструкции print между перечисленными значениями выводятся разделители. По умолчанию это пробелы.

```
x=2
y=5
print ( x, "+", y, "=", x+y, sep = " " )
```

Результат отобразится с пробелами между элементами: 2 + 5 = 7

Функция format формирует символьную строку заданного формата.

Функции преобразования в числовые типы:

- в целые числа – int (<данные>);
- в вещественные числа – float (<данные>).

Обзор программ

IDE (или интегрированная среда разработки) — это программа, предназначенная для разработки программного обеспечения. Как следует из названия, IDE объединяет несколько инструментов, специально предназначенных для разработки. Эти инструменты обычно включают редактор, предназначенный для работы с кодом (например, подсветка синтаксиса и автодополнение); инструменты сборки, выполнения и отладки; и определённую форму системы управления версиями.

PyCharm

Одной из лучших полнофункциональных IDE, предназначенных именно для Python, является PyCharm. Существует как бесплатный open-source (Community), так и

платный (Professional) варианты IDE. PyCharm доступен на Windows, Mac OS X и Linux.

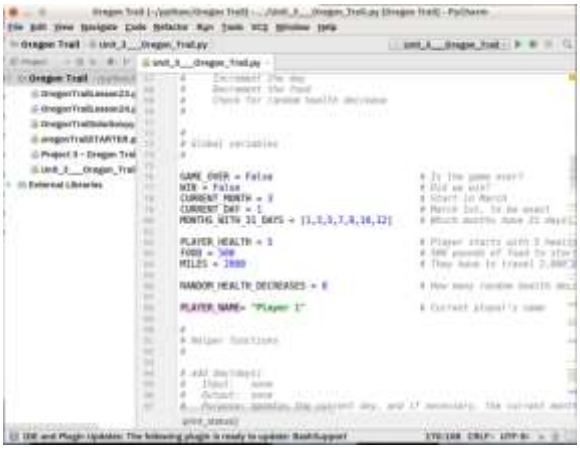


Рисунок 3

Python Shell

Окно Python Shell (Оболочка Python) обеспечивает интерактивный режим работы Python. Окно Python Shell обеспечивает доступ к интерактивному режиму Python. Оно откроется при запуске IDLE.

Как и в прямом интерактивном режиме, вы набираете выражение Python после приглашения >>> и затем нажимаете клавишу Enter, направляя команду в интерпретатор Python. В отличие от обычного интерактивного режима в середине многострочного составного выражения не появляется промежуточного приглашения (...).

Если возникла ситуация, когда все как будто зависло и не появляется нового приглашения, вероятно интерпретатор ждет от вас ввода чего-нибудь особенного (он занят анализом). Нажатие клавиш Ctrl-c вызовет клавиатурное прерывание и вернет вас к приглашению. Этот прием можно использовать для прерывания любой выполняющейся команды.

Подсветка синтаксиса

По мере набирания кода он подсвечивается в соответствии с синтаксическими типами Python.

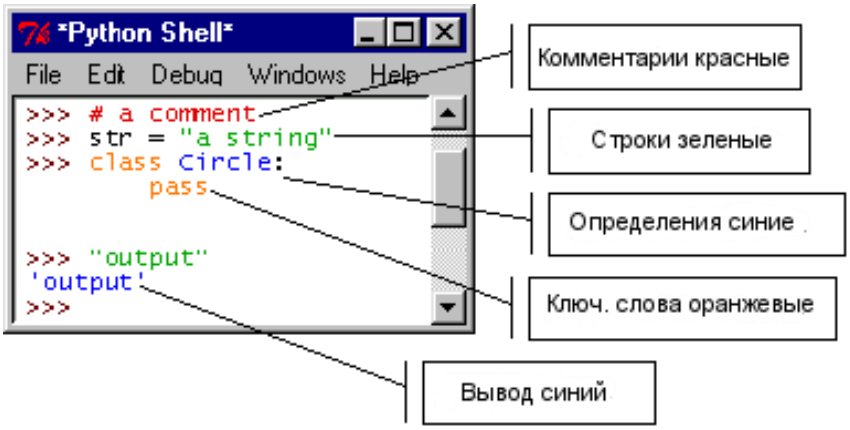


Рисунок 4

Отступы

Предусмотрена автоматическая поддержка отступов:

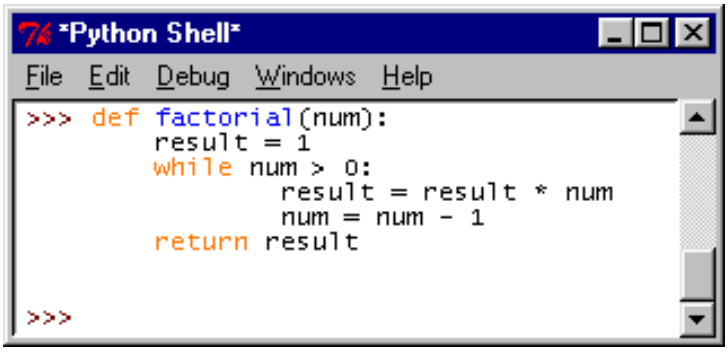


Рисунок 5

Создание проекта в Python Shell

1. Создадим текстовый файл с нужным нам именем и расширением ру
2. Откроем текстовый редактор при помощи IDLE Python. Для щелкаем по файлу ПКМ выбираем Edit width IDLE (он появится если вы правильно установили оболочку) для этого выбираем редактор в соответствии с вашим разрядом ОС. Здесь пишется код на питоне, для копирования используем комбинации клавиш, работает только английская раскладка.

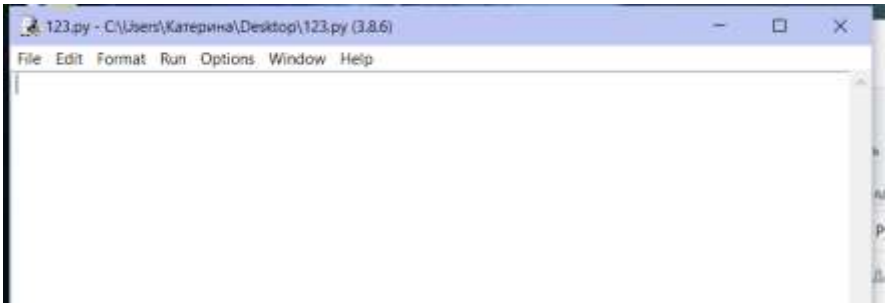


Рисунок 6

3. Для запуска нажимаем Run – Run Module или F5
4. Открывается командная оболочка Python. >>> - приглашение к написанию кода

Пример

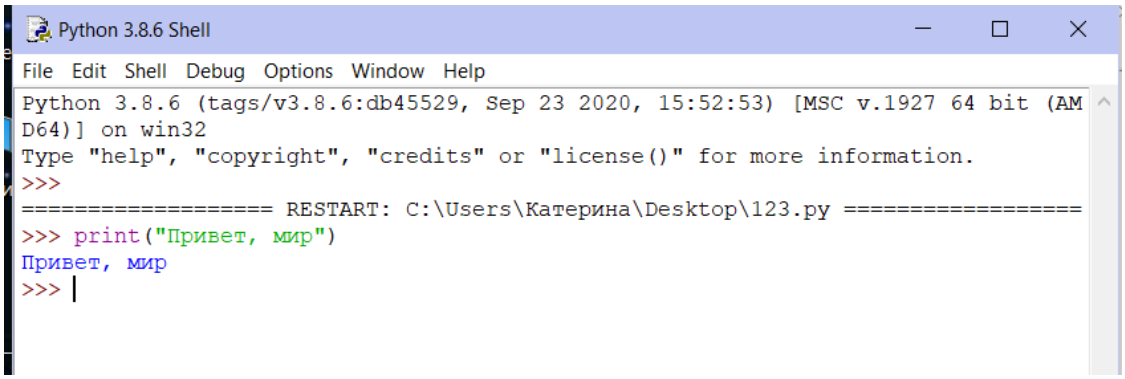


Рисунок 7

Синтаксис Python на примере функции print()

>>> (print ('Привет!')
SyntaxError: unexpected indent
>>>print ('Привет') print('!)'
SyntaxError: invalid syntax
>>>print ('Привет! '). /
SyntaxError: invalid syntax

Комментарии в Python обозначаются предваряющим их символом # и продолжаются до конца строки (т.е. в Python все комментарии являются однострочными), при этом не допускается использование перед символом # кавычек

Знаки пунктуации

В алфавит Python входит достаточное количество знаков пунктуации, которые используются для различных целей. Например, знаки "+" или " *" могут использоваться для сложения и умножения, а знак запятой "," - для разделения параметров функций.

Идентификаторы

Идентификаторы в Python это имена используемые для обозначения переменной, функции, класса, модуля или другого объекта.

Ключевые слова

Некоторые слова имеют в Python специальное назначение и представляют собой управляющие конструкции языка.

Типы данных в языке Python

Рисунок 8

Типы данных

1. None (неопределенное значение переменной)
2. Логические переменные (Boolean Type)
3. Числа (Numeric Type)
1. int – целое число
2. float – число с плавающей точкой
3. complex – комплексное число
4. Списки (Sequence Type)
1. list – список
2. tuple – кортеж
3. range – диапазон
5. Строки (Text Sequence Type)
1. str

Арифметические операторы

Python реализует семь основных бинарных арифметических операторов, два из которых могут использоваться как унарные операторы. Они приведены в следующей таблице:

Таблица 1

Оператор	**Название**	**Описание**
a + b	Сложение	Сумма a и b
a — b	Вычитание	Разность a и b
a * b	Умножение	Произведение a и b
a / b	Деление	Частное a и b

a // b	Целочисленное деление	Деление a на b без остатка (дробная часть отбрасывается)
a % b	Взятие модуля	Целый остаток от деления a на b
a ** b	Возведение в степень	a, возведенное в степень b
-a	Отрицание	Отрицательное значение a
+a	Унарный плюс	a без изменений (используется редко)

Эти операторы можно использовать и комбинировать интуитивно понятным образом, используя стандартные круглые скобки для группировки операций. К примеру, это может выглядеть так:

```
# сложение, вычитание, умножение
(4 + 8) * (6.5 - 3)
# 42.0
```

Целочисленное деление — это обычное деление, только с усечённой дробной частью:

```
# Деление
print(11 / 2)
# 5.5
# Целочисленное деление
print(11 // 2)
# 5
```

Оператор целочисленного деления был добавлен в Python 3. Если вы работаете в Python 2, вы должны знать, что стандартный оператор деления (/) действует как оператор целочисленного деления для целых чисел и как обычный оператор деления для чисел с плавающей запятой.

Наконец, упомянем ещё один арифметический оператор, который был добавлен в Python 3.5. Это оператор a @ b, предназначенный для указания матричного произведения a и b для использования в различных пакетах линейной алгебры.

Битовые операторы

В дополнение к стандартным числовым операциям в Python есть операторы для выполнения побитовых логических операций над целыми числами. Они используются гораздо реже стандартных арифметических операций, но знать их полезно. Шесть побитовых операторов сведены в следующую таблицу:

Таблица 2

Оператор	**Название**	**Описание**
a & b	Логическое И	Биты, определенные как в a, так и в b
a | b	Логическое ИЛИ	Биты, определенные в a или b или в обоих
a ^ b	Исключающее ИЛИ	Равен 1, если только a или только b равно 1
a << b	Побитовый сдвиг влево	Сдвинуть биты a влево на b единиц
a >> b	Побитовый сдвиг вправо	Сдвинуть биты a вправо на b единиц
~a	Логическое НЕ	Отрицание a

Эти побитовые операторы имеют смысл только с точки зрения двоичного представления чисел. Это можно увидеть, используя встроенную функцию bin:

```
bin(10)
# '0b1010'
```

Результат имеет префикс 0b, что указывает на двоичное представление. Остальные цифры означают, что число 10 выражается как сумма $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$. Точно так же мы можем написать:

```
bin(4)
# '0b100'
```

Теперь, используя логическое ИЛИ, мы можем найти число, которое объединяет биты 4 и 10:

```
4 | 10
# 14
bin(4 | 10)
# '0b1110'
```

Эти побитовые операторы не так полезны, как стандартные арифметические операторы, но стоит увидеть их хотя бы раз, чтобы понять, какой класс операций они выполняют. В частности, у пользователей других языков иногда возникает соблазн использовать исключающее ИЛИ (т. е. a ^ b), на самом деле имея в виду возведение в степень (т. е. a ** b).

Операторы присваивания

Мы видели, что переменным можно присваивать значение с помощью оператора =. Например, следующим образом:

```
a = 24
print(a)
# 24
```

Мы можем использовать эти переменные в выражениях с любым из упомянутых ранее операторов. Например, чтобы добавить 2 к a, мы пишем:

```
a + 2
# 26
```

Мы можем захотеть обновить переменную a новым значением. В этом случае мы могли бы объединить сложение и присваивание и написать a = a + 2. Поскольку этот тип комбинированной операции и присваивания очень распространен, Python включает встроенные операторы обновления для всех арифметических операций:

```
a += 2 # эквивалентно a = a + 2
print(a)
# 26
```

Расширенный оператор присваивания есть для каждого из бинарных операторов, перечисленных ранее. Они выглядят следующим образом:

```
a += b a -= b a *= b a /= b
a //= b a %= b a **= b a &= b
a |= b a ^= b a <<= b a >>= b
```

Каждый из этих операторов эквивалентен применению соответствующего оператора с последующим присваиванием. То есть для любого оператора ■ выражение a ■= b эквивалентно a = a ■ b с небольшой оговоркой.

Для изменяемых объектов, таких как списки, массивы или датафреймы, эти расширенные операции присваивания на самом деле немного отличаются от своих более подробных аналогов. Они изменяют содержимое исходного объекта, а не создают новый объект для хранения результата. Но это тонкости, в целом же укороченная версия работает так же, как и полная, но экономит кучу вашего времени.

Операторы сравнения

Другой тип операций, который может быть очень полезным, — это сравнение различных переменных. Для этого в Python реализованы стандартные операторы сравнения, которые возвращают логические значения True или False. Операции сравнения представлены в следующей таблице:

Таблица 3

Оператор	**Описание**
a == b	a равняется b
a < b	a строго меньше чем b
a <= b	a меньше либо равно b
a != b	a не равняется b
a > b	a строго больше чем b
a >= b	a больше либо равно b

Эти операторы сравнения можно комбинировать с арифметическими и побитовыми операторами, чтобы осуществлять самые разнообразные операции над числами. Например, мы можем проверить, является ли число нечетным, проверив, что остаток от деления на 2 возвращает 1:

```
# 25 - нечетное число
25 % 2 == 1
# True
# 66 - нечетное число
66 % 2 == 1
# False
```

Мы можем объединить несколько сравнений, чтобы проверить более сложные отношения:

```
# проверяем, находится ли a между 15 и 30
a = 25
15 < a < 30
# True
```

И, чтобы у вас немного закружилась голова, взгляните на это сравнение:

```
-1 == ~0
# True
```

Напомним, что ~ — это оператор инвертирования битов, и, очевидно, когда вы инвертируете все биты нуля, вы получите -1. Если вам интересно, почему это так, посмотрите схему кодирования целых чисел с дополнением до двух, которую Python использует для кодирования целых чисел со знаком, и подумайте, что происходит, когда вы начинаете переворачивать все биты целых чисел, закодированных таким образом.

От редакции Pythonist: об операторе != можно почитать в статье «Оператор неравенства != в Python».

Логические операторы

При работе с логическими значениями Python предоставляет операторы для объединения значений с использованием стандартных понятий «и», «или» и «не». Эти операторы ожидаемо представлены словами and, or и not:

```
x = 4
(x < 6) and (x > 2)
```

```
# True
(x > 10) or (x % 2 == 0)
# True
not (x < 6)
# False
```

Поклонники булевой алгебры могут заметить, что оператор исключающего ИЛИ не включен. Он, конечно, может быть построен несколькими способами путем составления других операторов. Или же вы можете использовать хитрый трюк:

```
# (x > 1) xor (x < 10)
(x > 1) != (x < 10)
# False
```

Иногда в языке возникает путаница: когда использовать логические операторы (and, or, not), а когда побитовые (&, |, ~). Ответ кроется в их названиях: логические операторы следует использовать, когда вы хотите вычислить логические значения (т.е. истинность или ложность) утверждений. Побитовые операции следует использовать, когда вы хотите работать с отдельными битами или компонентами рассматриваемых объектов.

Операторы принадлежности и идентичности

Помимо and, or и not, Python также имеет операторы для проверки принадлежности и идентичности. Они следующие:

Таблица 4

Оператор	Описание
a is b	Возвращает True, если a и b — идентичные объекты
a is not b	Возвращает True, если a и b — не идентичные объекты
a in b	Возвращает True, если a содержится в b
a not in b	Возвращает True, если a не содержится в b

Операторы is и is not проверяют идентичность объекта. Идентичность объекта отличается от равенства, как мы видим здесь:

```
a = [1, 2, 3]
b = [1, 2, 3]
a == b
# True
a is b
# False
a is not b
# True
```

Как выглядят одинаковые объекты? Вот пример:

```
a = [1, 2, 3]
b = a
a is b
# True
```

Разница между этими двумя случаями в том, что в первом a и b указывают на разные объекты, а во втором они указывают на один и тот же объект. Оператор is проверяет, указывают ли две переменные на один и тот же контейнер (объект), а не на то, что содержит контейнер.

Новички часто испытывают искушение использовать is, хотя на самом деле имеют в виду ==. Подробнее о разнице между == и is можно почитать в статье «Чем == отличается от is?«.

Операторы принадлежности проверяют принадлежность к составным объектам. Так, например, мы можем написать:

```
1 in [1, 2, 3]
# True
2 not in [1, 2, 3]
# False
```

Эти операции членства являются примером простоты Python по сравнению с языками более низкого уровня, такими как C. В языке C членство обычно определяется путем ручного построения цикла по списку и проверки равенства каждого значения. В Python вы просто печатаете то, что хотите узнать, в манере, напоминающей простой текст.

Тема 2.2. Условия и циклы Python

План:

1. Разветвляющиеся алгоритмы
2. Логическое высказывание
3. Логические переменные
4. Инструкция ветвления
5. Сложные условия. Каскадные ветвления
6. Преобразование типов
7. Циклы
8. Цикл с предусловием
9. Бесконечный цикл
10. Цикл с параметром

Разветвляющиеся алгоритмы — это алгоритмы, содержащие конструкции ветвления.

Ветвление — это алгоритмическая конструкция, в которой, в зависимости от некоторого условия, происходит исполнение одной из двух последовательностей действий (ветвей).

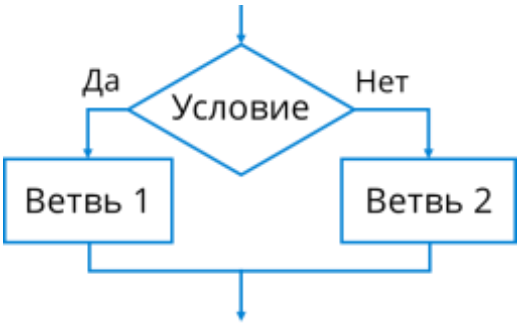


Рисунок 1

Логическое высказывание — это утверждение, истинность которого можно определить однозначно.

Логические переменные

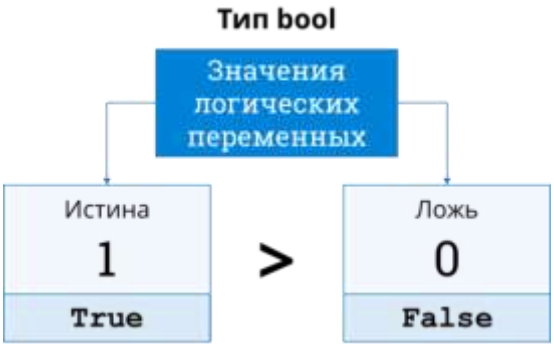


Рисунок 2

Инструкция ветвления

Запись инструкции ветвления:

if
  <инструкция 1>
  <инструкция 2>
  …
  <инструкция n>
else
  <инструкция n + 1>
  <инструкция n + 2>
  …
  <инструкция m>

Пример инструкции ветвления

Нужно присвоить переменной M наибольшее из значений, содержащихся в переменных a и b, или любое из значений, если они равны.

Рисунок 3

Логические операции

Рисунок 4

Сложные условия. Каскадные ветвления

Запись каскадного ветвления:

If <условие 1>:
  <инструкции 1>
elif <условие 2>:
  <инструкции 3>
elif <условие 3>:
  <инструкции 4>
else:
  <инструкции 2>

Пример.

```
>>> t = 13
>>> if t < 12:
        print("Надень шапку")
elif t < 20:
        print("Можно без шапки")
else:
        print("Нужно без шапки")

Можно без шапки
```

Рисунок 5

Преобразование типов

```
>>> t = "12"
>>> t
'12'   был тип данных str
>>> t = int(t)
>>> t
12     стал тип данных int
```

Рисунок 6

Пример. Напишите в IDLE программу, которая спрашивает вашу любимую игру, а затем печатает результат

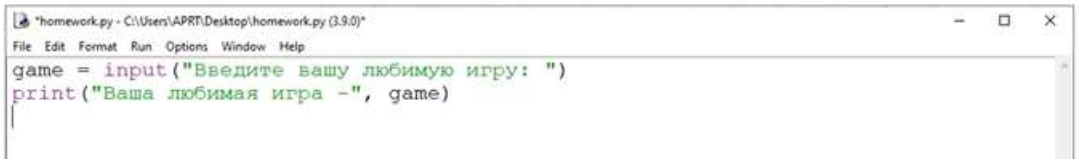


Рисунок 7

Дополним программу выводом количество часов для игры

```
game = input("Введите вашу любимую игру: ")
print("Ваша любимая игра -", game)

if game == "CS:GO":
    print("Вы наиграли в неё 420 часов")
elif game == "Mass Effect 2":
    print("Вы наиграли в неё 70 часов")
else:
    print("Странно, вы в неё не играли")
```

Рисунок 8

Напишите программу для сравнения возраста

```
age1 = int(input("Возраст Ивана: "))
age2 = int(input("Возраст Артёма: "))

if age1 > age2:
    print("Иван старше Артёма")
elif age1 == age2:
    print("Иван и Артёму поровну лет")
else:
    print("Артём старше Ивана")


Возраст Ивана: 12
Возраст Артёма: 14
Артём старше Ивана
```

Рисунок 9

Улучшим программу для сравнения возраста сразу трех человек

```
age1 = int(input("Возраст Ивана: "))
age2 = int(input("Возраст Артема: "))
age3 = int(input("Возраст Александра: "))

if age1 > age2:
        if age1 > age3:
                print("Иван старше всех")
        elif age1 == age3:
                print("Иван и Александр старше Артема")
        else:
                print("Александр старше всех")
elif age1 == age2:
        if age1 > age3:
                print("Иван и Артем старше Александра")
        elif age1 == age3:
                print("Иван, Артем и Александр - ровесники")
        else:
                print("Александр старше всех")
else:
        if age2 > age3:
                print ("Артем старше всех")
        elif age2 == age3:
                print("Аретем и Александр старше Ивана")
        else:
                print ("Александр старше всех")
```

Рисунок 10

Циклы

Цикл — это алгоритмическая конструкция, которая представляет собой последовательность действий, повторяющихся многократно.

Рисунок 11

Цикл с предусловием

Запись цикла с предусловием:

while

<инструкция 1>

<инструкция 2>

…

<инструкция n>

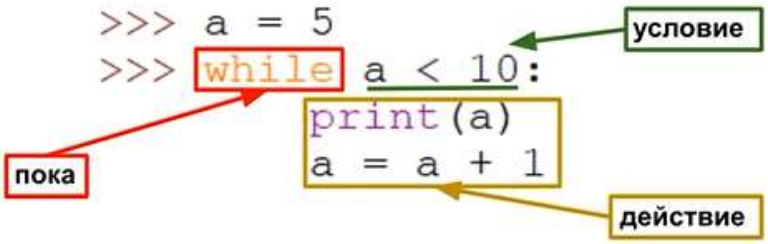


Рисунок 12

В результате будет выполнено 5 итераций

```
>>> a = 10
>>> while a > 0:
        print(a)
        a = a - 1
```

Рисунок 13

В результате будет выполнено 10 итераций

Бесконечный цикл – это цикл условие, которого всегда возвращает True. По сути мы так и можем писать while True

```
>>> a = "Привет"
>>> while True:
        print(a)
```

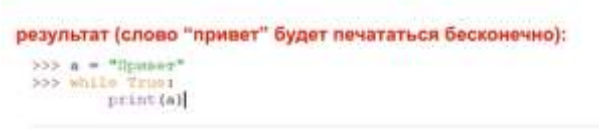


Рисунок 14

Чтобы остановить бесконечный цикл необходимо выполнить Shell – Restart Shell или Ctrl+F6.

Но это является неудобным, т.к. каждый раз придется перезапускать программу, поэтому для таких циклов предусмотрен оператор break. Оператор ставится в нужное место и срабатывание цикла в нем завершается.

Пример. Создадим переменную, а после запустим бесконечный цикл с ее распечатыванием, даем пользователю ее остановить в любой момент с помощью input(). И далее пишем условие: «Если пользователь ввел, нет, то завершить цикл и вывести на экран сообщение об этом. А если пользователь отвечает да, то цикл повторяется бесконечно»

```
>>> a = "Привет"
>>> while True:
        print(a)
        b = input("Продолжать печатать?(да/нет) ")
        if b == "нет":
            print("Цикл остановлен")
            break

Привет
Продолжать печатать?(да/нет) да
Привет
Продолжать печатать?(да/нет) да
Привет
Продолжать печатать?(да/нет) нет
Цикл остановлен
>>>
```

Рисунок 15

Запись цикла с постусловием

В языке Python нет отдельной инструкции для записи цикла с постусловием, поэтому он реализуется через бесконечный цикл с предусловием:

while True:

<инструкция 1>

<инструкция 2>

…

<инструкция n>

if <условие цикла>:

break

Рисунок 16

Пример. Написать программу, генерирующую случайное целое число на промежутке [a; b], предоставляющую пользователю неограниченное число попыток для того, чтобы его угадать. Если пользователю надоело угадывать, то он должен ввести слово «Сдаюсь». Условие a < b

Принцип работы цикла

```
while s != str (n):
    s = input ()
    if s == 'Сдаюсь':
        break
    elif s == str (n):
        continue
    print ('Вы не угадали. Попробуйте ещё раз.')
else:
    print ('Поздравляем! Вы угадали.')
...
```

Рисунок 17

Цикл с параметром

В цикле с параметром параметр принимает все значения из заданного множества и для каждого значения выполняется тело цикла.

Запись цикла с параметром:

for <параметр> in <множества>:

<инструкция 1>

<инструкция 2>

…

<инструкция n>

В теле цикла с параметром нельзя изменять значение параметра.

Тема 2.3. Списки и кортежи Python

План:

1. Списки
2. Встроенная функция append()
3. Команда del
4. Кортежи (tuple)
5. Зачем нужны кортежи в Python?
6. Создание кортежа в Python
7. Индексирование и нарезка кортежей
8. Основные операции с кортежами
9. Встроенные функции

Списки

Строка – это упорядоченный набор символов

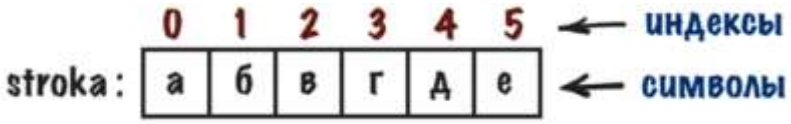


Рисунок 1

Список — это набор элементов, которые хранятся под одним именем. Каждый элемент списка имеет уникальный идентификационный номер (индекс).

Список – это динамическая структура. Это означает, что его длина может изменяться в процессе исполнения программы.

Список – это последовательность элементов, разделенных между собой запятой и помещённых в квадратные скобки.

Тип данных	Описание	Пример
list	Список	[2, 3.6, “Hello”]

например, список одноклассников:

```
classmates = ["Артём", "Иван", "Дарья", "Михаил"]
```

или список чисел:

```
numbers = [25, 3, 9, 137]
```

Рисунок 2

Каждый элемент списка пронумерован, т.е. имеет свой индекс, отсчет начинается с нуля.

```
                [0]       [1]       [2]       [3]
classmates = ["Артём", "Иван", "Дарья", "Михаил"]

              [0]  [1]  [2]  [3]
numbers = [25, 3, 9, 137]
```

Рисунок 3

Чтобы получить элемент списка, нужно написать название списка, а в квадратных скобках номер элемента

```
                [0]       [1]       [2]       [3]
classmates = ["Артём", "Иван", "Дарья", "Михаил"]

>>> classmates[2]
'Дарья'
>>> print(classmates[2])
Дарья

>>> a = classmates[3]
>>> a
'Михаил'
```

Рисунок 4

Встроенная функция append() – добавить. Добавляет элемент к концу списка.

было 4 одноклассника:

```
>>> classmates
['Артём', 'Иван', 'Дарья', 'Михаил']
```

добавился ещё один в конец списка:

```
>>> classmates.append("Мария")
```

теперь одноклассников 5:

```
>>> classmates
['Артём', 'Иван', 'Дарья', 'Михаил', 'Мария']
```

Рисунок 5

Команда del – удалить. Удаляет элемент из списка по его индексу.

было 5 одноклассников:

```
>>> classmates
['Артём', 'Иван', 'Дарья', 'Михаил', 'Мария']
```

вызываем del к последнему однокласснику под индексом 4:

```
>>> del classmates[4]
```

осталось 4 одноклассника:

```
>>> classmates
['Артём', 'Иван', 'Дарья', 'Михаил']
```

Рисунок 6

Также списки можно складывать и умножать

```
>>> classmates = ["Артём", "Иван", "Дарья", "Михаил"]
>>> vegetables = ["potato", "carrot", "cucumber"]
>>> classmates + vegetables
['Артём', 'Иван', 'Дарья', 'Михаил', 'potato', 'carrot', 'cucumber']
```

```
>>> classmates * 5
['Артём', 'Иван', 'Дарья', 'Михаил', 'Артём', 'Иван', 'Дарья', 'Михаил', 'Артём'
, 'Иван', 'Дарья', 'Михаил', 'Артём', 'Иван', 'Дарья', 'Михаил', 'Артём', 'Иван'
, 'Дарья', 'Михаил']
```

Рисунок 7

Кортежи (tuple)

По аналогии со списками кортежи в Python — это стандартный тип, позволяющий хранить значения в виде последовательности. Они полезны в тех случаях, когда необходимо передать данные, не позволяя изменять их. Эти данные могут быть использованы, но в оригинальной структуре изменения не отобразятся.

Кортеж (tuple) – это неизменяемая структура данных, которая по своему подобию очень похожа на список.

Список – это изменяемый тип данных. Т.е. если у нас есть список *a = [1, 2, 3]* и мы хотим заменить второй элемент с 2 на 15, то мы может это сделать, напрямую обратившись к элементу списка.

```
>>> a = [1, 2, 3]
>>> print(a)
[1, 2, 3]
>>> a[1] = 15
>>> print(a)
[1, 15, 3]
```

С кортежем мы не можем производить такие операции, т.к. элементы его изменять нельзя.

```
>>> b = (1, 2, 3)
>>> print(b)
(1, 2, 3)
>>> b[1] = 15
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
```

b[1] = 15

TypeError: 'tuple' object does not support item assignment

Кортежи создают с помощью круглых скобок (). Для создания нужно написать следующее:

cake = ('c','a','k','e') print(type(cake)) <class 'tuple'>

type() — это встроенная функция для проверки типа данных переданного параметра.

Зачем нужны кортежи в Python?

Существует несколько причин, по которым стоит использовать кортежи вместо списков. Одна из них – это обезопасить данные от случайного изменения. Если мы получили откуда-то массив данных, и у нас есть желание поработать с ним, но при этом непосредственно менять данные мы не собираемся, тогда, это как раз тот случай, когда кортежи придутся как нельзя кстати. Используя их в данной задаче, мы дополнительно получаем сразу несколько бонусов – во-первых, это экономия места. Дело в том, что кортежи в памяти занимают меньший объем по сравнению со списками.

```
>>> lst = [10, 20, 30]
>>> tpl = (10, 20, 30)
>>> print(lst.__sizeof__())
32
>>> print(tpl.__sizeof__())
24
```

Во-вторых – прирост производительности, который связан с тем, что кортежи работают быстрее, чем списки (т.е. на операции перебора элементов и т.п. будет тратиться меньше времени). Важно также отметить, что кортежи можно использовать в качестве ключа у словаря.

Создание кортежа в Python

Кортеж в Python можно записать как набор значений, разделенных запятыми(,), заключенных в маленькие скобки(). Скобки необязательны, но их рекомендуется использовать. Кортеж можно определить следующим образом.

```
T1 =(101, "Peter", 22)
T2 =("Apple", "Banana", "Orange")
T3 = 10,20,30,40,50
print(type(T1))
print(type(T2))
print(type(T3))
```

Выход:

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
```

Примечание. Кортеж, созданный без скобок, также известен как упаковка кортежа.

Пустой кортеж можно создать следующим образом.

```
T4 =()
```

Создание кортежа с одним элементом немного отличается. Нам нужно будет поставить запятую после элемента, чтобы объявить кортеж.

```
tup1 =("JavaTpoint")
```

```
print(type(tup1))
#Creating a tuple with single element
tup2 =("JavaTpoint",)
print(type(tup2))
```

Выход:

```
<class 'str'>
<class 'tuple'>
```

Кортеж индексируется так же, как и списки. Доступ к элементам в кортеже можно получить, используя их конкретное значение индекса.

Рассмотрим следующие примеры кортежа:

Пример 1.

```
tuple1 =(10, 20, 30, 40, 50, 60)
print(tuple1)
count = 0
for i in tuple1:
   print("tuple1[%d] = %d"%(count, i))
   count = count+1
```

Выход:

```
(10, 20, 30, 40, 50, 60)
tuple1[0] = 10
tuple1[1] = 20
tuple1[2] = 30
tuple1[3] = 40
tuple1[4] = 50
tuple1[5] = 60
```

Пример 2.

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
   print("tuple1[%d] = %s"%(count, i))
   count = count+1
```

Выход:

```
Enter the tuple elements ...123456('1', '2', '3', '4', '5', '6')
tuple1[0] = 1
tuple1[1] = 2
tuple1[2] = 3
tuple1[3] = 4
tuple1[4] = 5
tuple1[5] = 6
```

Индексирование и нарезка кортежей

Индексация и нарезка кортежа аналогичны спискам. Индексирование в кортеже начинается с 0 и продолжается до длины(tuple) – 1.

Доступ к элементам в кортеже можно получить с помощью оператора index []. Python также позволяет нам использовать оператор двоеточия для доступа к нескольким элементам в кортеже.

Рассмотрим следующее изображение, чтобы подробно разобраться в индексировании и нарезке.

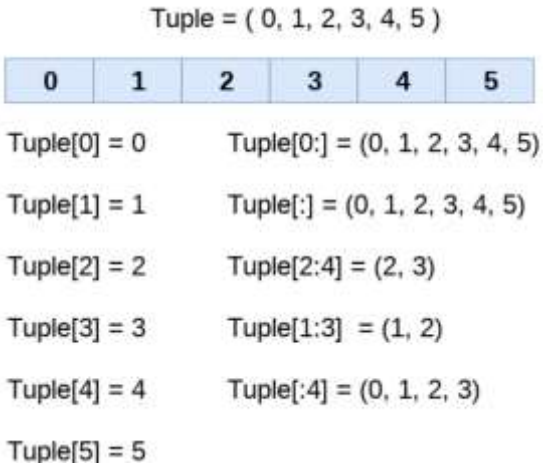


Рисунок 8

Рассмотрим следующий пример:

Выход:

1
2
3
tuple index out of range

В приведенном выше коде кортеж состоит из 7 элементов, которые обозначают от 0 до 6. Мы попытались получить доступ к элементу вне кортежа, который вызвал ошибку IndexError.

```
tuple =(1,2,3,4,5,6,7)
#element 1 to end
print(tuple[1:])
#element 0 to 3 element
print(tuple[:4])
#element 1 to 4 element
print(tuple[1:5])
# element 0 to 6 and take step of 2
print(tuple[0:6:2])
```

Выход:

(2, 3, 4, 5, 6, 7)(1, 2, 3, 4)(1, 2, 3, 4)(1, 3, 5)

Отрицательное индексирование

К элементу кортежа также можно получить доступ, используя отрицательную индексацию. Индекс -1 обозначает крайний правый элемент, а -2 – второй последний элемент и так далее.

Элементы слева направо перемещаются с использованием отрицательной индексации.

Рассмотрим следующий пример:

```
tuple1 =(1, 2, 3, 4, 5)
print(tuple1[-1])
print(tuple1[-4])
print(tuple1[-3:-1])
print(tuple1[:-1])
```

```
print(tuple1[-2:])
```

Выход:

```
5
2(3, 4)(1, 2, 3, 4)(4, 5)
```

Удаление кортежа

В отличие от списков, элементы кортежа нельзя удалить с помощью ключевого слова del, поскольку кортежи неизменяемы. Чтобы удалить весь кортеж, мы можем использовать ключевое слово del с именем кортежа.

Рассмотрим пример:

```
tuple1 =(1, 2, 3, 4, 5, 6)
print(tuple1)
del tuple1[0]
print(tuple1)
del tuple1
print(tuple1)
```

Выход:

```
(1, 2, 3, 4, 5, 6)
Traceback(most recent call last):
  File "tuple.py", line 4, in <module>
    print(tuple1)
NameError: name 'tuple1' is not defined
```

Основные операции с кортежами

Такие операторы, как конкатенация(+), повторение(*), членство(in), работают так же, как они работают со списком. Для получения более подробной информации рассмотрите следующую таблицу.

Допустим, объявлены Tuple t =(1, 2, 3, 4, 5) и Tuple t1 =(6, 7, 8, 9).

Оператор	Описание	Пример
Repetition	Оператор повторения позволяет повторять элементы кортежа несколько раз.	`T1 * 2 =(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)`
Concatenation	Он объединяет кортеж, указанный по обе стороны от оператора.	`T1 + T2 =(1, 2, 3, 4, 5, 6, 7, 8, 9)`
Membership	Он возвращает истину, если в кортеже существует конкретный элемент, в противном случае – ложь.	`print(2 in T1) печатает True.`

Iteration	Цикл for используется для перебора элементов кортежа.	`для i в T1:`
		`печать(i)`
		Выход
		`1`
		`2`
		`3`
		`4`
		`5`

Length	Он используется для получения длины кортежа.	`len(T1) = 5`

Рисунок 9

Встроенные функции

№	Функция	Описание
1	cmp(tuple1, tuple2)	Сравнивает два кортежа и возвращает значение true, если кортеж 1 больше, чем кортеж 2, в противном случае значение false.
2	len(tuple)	Вычисляет длину кортежа.
3	max(tuple)	Возвращает максимальный элемент кортежа
4	min(tuple)	Возвращает минимальный элемент кортежа
5	tuple(seq)	Преобразует указанную последовательность в кортеж.

Рисунок 10

Где использовать кортеж?

Кортежи используются вместо списка в следующих сценариях.

1. Использование кортежа вместо списка дает нам четкое представление о том, что данные кортежа постоянны и не должны изменяться.

2. Кортеж может имитировать словарь без ключей. Рассмотрим следующую вложенную структуру, которую можно использовать как словарь.

[(101, "John", 22),(102, "Mike", 28), (103, "Dustin", 30)]

Тема 2.4. Функции и файлы Python

План:

1. Функции
2. Локальные и глобальные переменные
3. Инструкция возврата

Функции

При программировании вспомогательные алгоритмы оформляются в виде подпрограмм.

В языке Python в роли подпрограмм выступают функции.

Функция — это подпрограмма, которая позволяет производить одни и те же действия над различными данными в различных местах программы.

Описание функции:

def <имя> (<входные параметры>)

    <инструкция 1>
    <инструкция 2>
    …
    <инструкция n>

Пример

Допустим в нескольких местах программы нам нужно вывести одно и тоже текстовое сообщение. Для этого можно использовать функцию print(). Однако если нам понадобится изменить текст выводимого сообщения, то придется искать в коде программы все инструкции print для вывода данного сообщения и изменять их одну за другой. Это может занять некоторое время, к тому же есть возможность пропустить некоторые инструкции, тогда программа будет работать неправильно. Этого можно избежать, если описать функцию вывода на экран поясняющего сообщения.

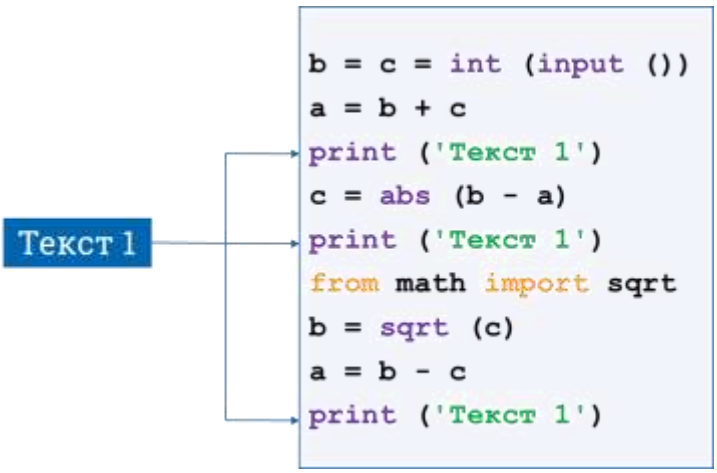


Рисунок 1

Функция описывается в программе до ее первого использования. Т.о. для того чтобы описать функцию, которую мы привели в примере нужно написать слово def после чего имя функции, наша функция не будет содержать параметров, поэтому через пробел укажем пустые скобки. Далее через двоеточие описываем тело функции, которое будет содержать одну инструкцию print, которая выведет на экран заданное текстовое сообщение. Теперь когда это сообщение нужно вывести в программе будет достаточно вызвать описанную функцию записав ее имя и перечислить после него в скобках значения входных параметров функции. Т.к. наша функция не имеет параметров после имени будут следовать пустые скобки. Т.о. при каждом вызове функции будут выполняться инструкции описанные в ее теле, т.е. инструкция print выводящая на экран текстовое сообщение. Следовательно для того чтобы изменить выводимое сообщение, достаточно изменить инструкцию один раз.

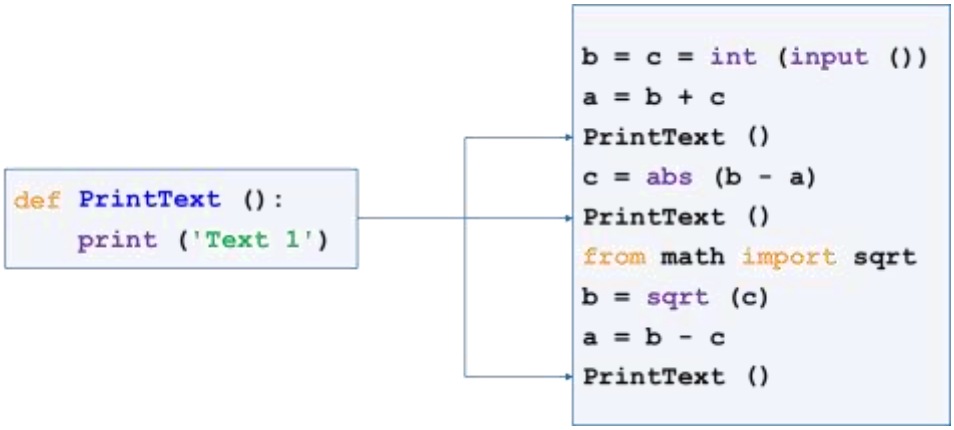


Рисунок 2

При вызове функции количество и порядок следования значений входных параметров должны соответствовать указанным в описании

Пример

def NoName (a, b, c)

<тело функции>

NoName (2.1, 7.4, 8)

Задача. Написать программу перевода целого положительного числа из десятичной системы счисления в двоичную.

Чтобы представить десятичное число в двоичной системе счисления необходимо делить его на два пока оно больше нуля, записывая остатки от деления, после чего нужно вывести эти остатки от деления в обратном порядке.

/* Описание функции с параметром Dec – число в десятичной системе счисления
def DecToBin (Dec):
/* Тело функции
    Bin = “ “ /* Объявляем пустую строку
/* Цикл для перевода числа
    while Dec > 0:
/* Тело цикла будет содержать инструкцию присваивания переменным Dec и Bin соответственно деление переменной Dec на цело на 2, а также соединение остатка от деления Dec на 2, преобразованного в тип str и текущего значения Bin. Т.о. на каждом шаге цикла Dec будет делиться на 2, при этом остаток от деления будет записываться в строку Bin слева от ее текущего значения

        Dec, Bin = Dec // 2, str (Dec % 2) + Bin
print (Bin)

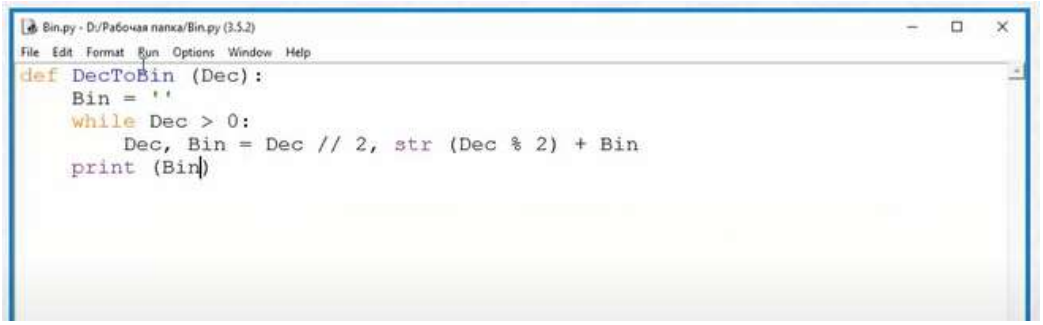


Рисунок 3

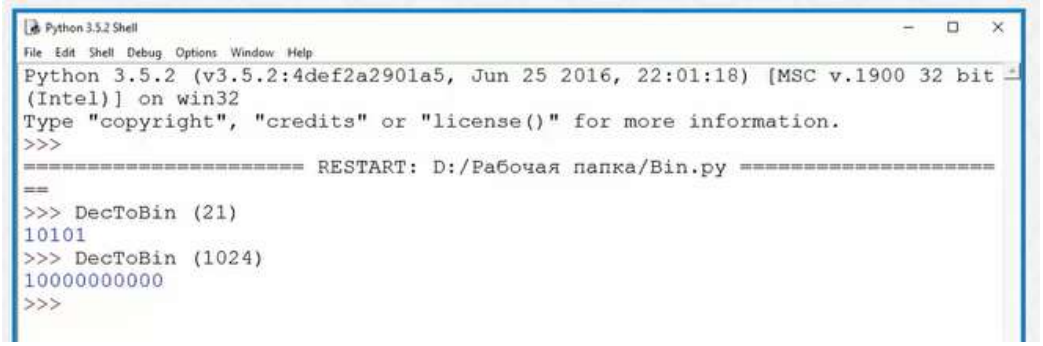


Рисунок 4

Локальные и глобальные переменные

В теле функции могут использоваться дополнительные параметры. К ним нельзя обращаться вне функции. Эти параметры, как и входные параметры функции, называются локальными. Из функции же напротив можно обращаться к переменным, описанным вне ее. Такие переменные называются глобальными параметрами. Для того чтобы обратиться к таким переменным из функции нужно записать служебное слово global, после которого через запятую перечислить глобальные параметры необходимые для работы функции. Далее к ним можно обращаться также как и к локальным параметрам.

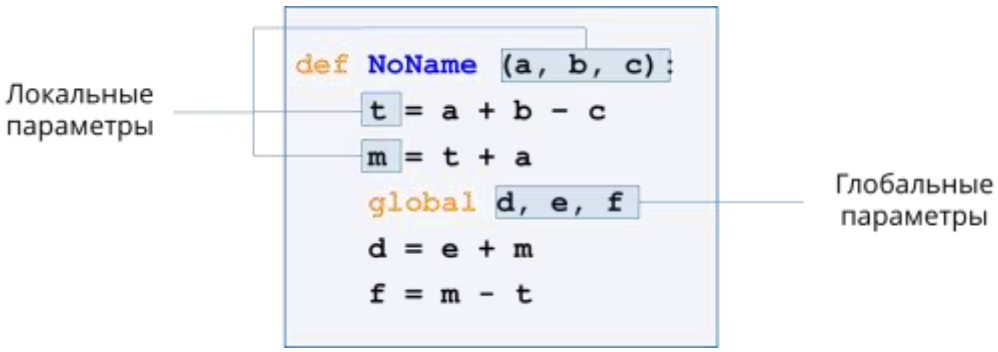


Рисунок 5

Инструкция возврата

При срабатывании инструкции возврата функция завершает свою работу, вернув в основную программу значения параметров, указанных в инструкции.

Инструкция возврата:

return <выходные параметры>

Пример вызова функции с инструкцией возврата:

```
def Noname (a, b, c)
    return a + b, b + c
d, e = NoName (1, 2, 3)
```

Задача. В уже имеющемся модуле:

1. Изменить имеющуюся функцию перевода целого положительного числа из десятичной системы счисления в двоичную таким образом, чтобы она возвращала получившееся число в программу.

2. Написать противоположную ей функцию перевода целого положительного числа из двоичной системы счисления в десятичную.

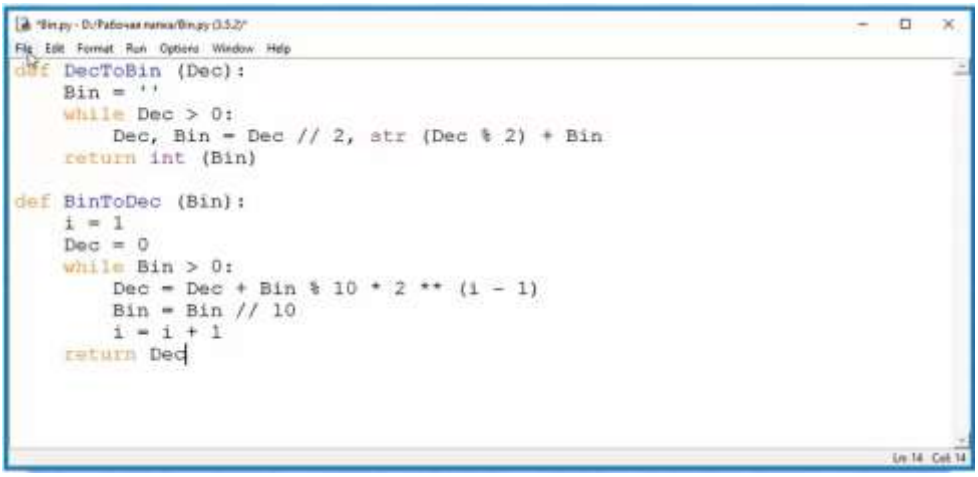


Рисунок 6

или

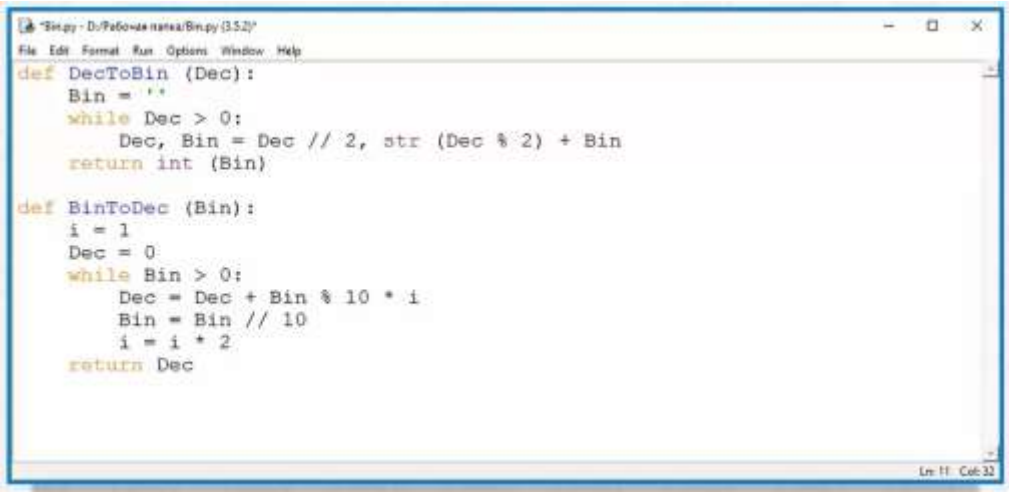


Рисунок 7

Рисунок 8

После сохранения модуля под именем Bin и протестируем его. Для этого создадим модуль с именем test в той же папке, что и модуль Bin. Загрузим в модуль test из модуля Bin обе описанные в нем функции.

Рисунок 9

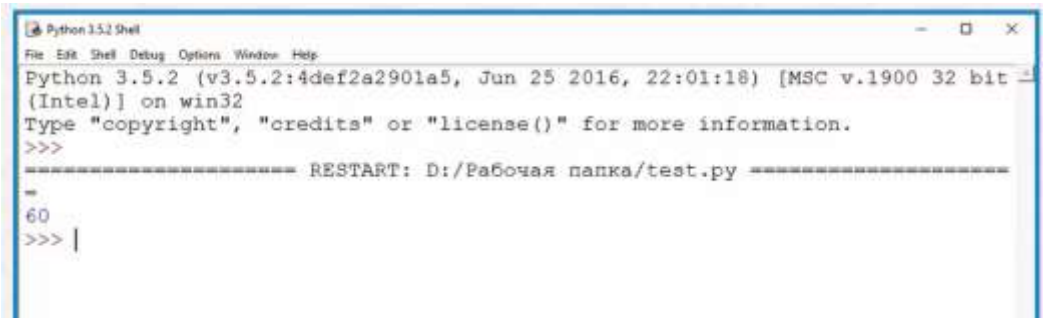


Рисунок 10

Вопросы и задания для самоконтроля по разделу 2

1. Приведите основные вехи развития языка Python: историю возникновения, кем и когда был создан, целевое предназначение, основные версии.
2. Перечислите основные характеристики языка программирования Python, укажите некоторые отличительные особенности. Дзен языка Python.
3. Python-программа: лексическая и семантическая структура, структура файла, основные элементы и блоки, кодировка файла.
4. Выполнение Python-программ: возможные варианты запуска программ и работы с интерпретатором.
5. IDE и их назначение. Примеры известных IDE для Python и их возможности (на любом примере).
6. Дайте определение типа данных и переменной. Приведите классификацию встроенных типов данных в Python. Чем объясняется наличие большого количества разных типов?
7. Оператор присваивания. Управление памятью и сборщик мусора.
8. Скалярные типы данных: числа, логический тип, NoneType. Определение, основные операции.
9. Последовательности: список, кортеж, числовой диапазон. Определение, основные операции. Где целесообразно использовать каждую из структур?
10. Множества: определение, основные операции. Где целесообразно применять множества?
11. Отображения (словарь): определение, основные операции. Где целесообразно применять словари?
12. Общие функции для объектов, приоритет операций. Можно ли повлиять на приоритет выполнения операций?
13. Проверка типов и взаимное преобразование. Для чего может понадобиться проверка типов и преобразование?
14. Разница между поверхностной и глубокой копией. Для каких типов глубокая копия имеет смысл и для чего может понадобиться?
15. Ввод и вывод в терминал: особенности и примеры.
16. Для чего необходимы конструкции ветвления и циклов? Поддерживает ли их Python?
17. Условный оператор if: случаи использования, синтаксис, особенности выполнения, примеры.
18. Цикл while: случаи использования, синтаксис, особенности выполнения, примеры.
19. Цикл for: случаи использования, синтаксис, особенности выполнения, примеры. Итераторы и специальные функции.
20. Прерывание и продолжение цикла: операторы break, continue и др.
21. Вложенные циклы: необходимость, характерные примеры.
22. Возможность комбинации условных и циклических структур. Примеры использования.
23. Коллекционные включения: область применения, примеры.
24. Подпрограмма: определение, правила объявления. Параметры и аргументы, параметры по ссылке и по значению. Понятие вызова функции. Стек вызовов.

25. Определение функции в Python, ключевое слово return. Правила PEP8 для функций. Четыре типа функций в Python.
26. Позиционные и ключевые параметры/аргументы функции. Примеры и особенности вычисления.
27. Упаковка и распаковка аргументов.
28. Понятие области видимости. Четыре области видимости в Python. Ключевые слова global и nonlocal.
29. Понятие рекурсии и возврат нескольких значений из функции. Примеры использования.
30. Строки документации: необходимость и примеры оформления.
31. Анонимные функции: целесообразность и примеры применения.
32. Побочные эффекты в программировании, основные правила при создании функций.
33. Ошибки в программном обеспечении: основные причины, примеры.
34. Определение программной ошибки и классификация: синтаксические, логические (семантические), ошибки времени выполнения и недокументированное поведение.
35. Поиск ошибок. Понятие отладки программы.
36. Подходы к обработке ошибок: LBYL («Семь раз отмерь, один раз отрежь») и EAFP («Легче попросить прощения, чем разрешения»). Понятие и механизм обработки исключений.
37. Понятие исключения в Python, иерархия классов-исключений.
38. Конструкция try: синтаксис, варианты потока выполнения, примеры.
39. Возбуждение исключений (raise) и особенности обработки исключений внутри функций. Утверждения в Python (assert).
40. Рекомендации относительно обработки исключительных ситуаций и утверждений.
41. Файловая система, файлы и каталоги: определения и терминология.
42. Свойства файла. Абсолютный и относительный путь.
43. Операции над файлами. Виды файлов: файлы с последовательным и произвольным доступом.
44. Файловый объект в Python, функция open(). Обработка исключений при работе с файловым объектом.
45. Основные свойства и методы файлового объекта. Примеры чтения и записи файла (одиночное, построчное, обработка файла целиком).
46. Сериализация и десериализация: определение и области применения. Модуль pickle.
47. Популярные форматы текстовых данных. Модуль csv и пакет json. Примеры работы с CSV и JSON.
48. Модульное программирование. Понятие модуля и пакета, их реализация в языке Python.
49. Классификация модулей/пакетов в Python.
50. Подключение и использование модулей/пакетов. Инструкция import.
51. Область поиска модулей. Общая схема импорта.
52. Специальные и дополнительные атрибуты модуля.
53. Кэширование («компиляция») модулей. Особенности *.pyc-файлов.

54. Собственные модули: нюансы PEP8, примеры. Разница между запуском и импортом модуля.
55. Основные функции модуля math, сравнение чисел с заданной точностью. Предназначение и основные функции модуля statistics.
56. Генератор псевдослучайных чисел: проблема случайности в компьютерной технике, характеристики псевдослучайных чисел. Модуль random.
57. Модули и пакеты Python для работы с датой/временем. Поддержка часового пояса и перехода на летнее/зимнее время. Модуль datetime, а также содержащиеся в нем классы (datetime.timedelta, datetime.time, datetime.date, datetime.datetime), их основные атрибуты.
58. Классы datetime.tzinfo, datetime.timezone. Библиотека pytz. Функции strftime() и strptime(), примеры их использования. Модуль locale. Модуль time: назначение, основные понятия и функции.
59. Поддержка функций для работы с платформой и операционной системой в Python. Содержимое, основные классы и функции модулей sys, platform, os, os.path, shutil, subprocess, glob.
60. Понятие регулярных выражений. Флаги, символы и классы символов, квантификаторы, группировка и выбор, проверка границ (привязки). Поддержка регулярных выражений в Python, модуль re.
61. Поддержка функционального программирования в Python. Функции map, filter, functools.reduce, случаи их применения.

Решение задач

Задачи на составление блок - схем

1. Найдите площадь треугольника по формуле Герона. Здесь a, b, c – это длины сторон, S – площадь треугольника, P – периметр. Составьте блок - схему
2. Вычислить значение функции y=ax+b. Составьте блок - схему
3. Даны длины двух катетов прямоугольного треугольника. Определить периметр этого треугольника. Составьте блок - схему
4. . Составить линейный алгоритм вычисление площади квадрата со стороной a. Составьте блок - схему
5. Вычислить значение выражений $\frac{a}{b} - \frac{c}{d}$. Если b >0, d>0. Составьте блок - схему
6. Составить блок-схему для решения задачи: дано число *X*. Увеличить его на 10, если оно положительное, во всех остальных случаях уменьшить его на 10.
7. Ввести число. Если оно больше 20, разделить его на 2, еслименьше или равно 20, то умножить на 5. Составьте блок - схему
8. Вычислить значение функции $Y = \begin{cases} 2X, \text{если } X < 0 \\ 2XZ, \text{если } X = 0 \\ Z, \text{если } \sin X > 0 \end{cases}$ (у вас будет проверка условия три раза). Составьте блок - схему
9. Составить блок-схему и программу вводящие с клавиатуры целые числа и суммирующие их, до тех пор пока не будет введен 0.
10. С клавиатуры вводятся N натуральных чисел. Рассчитать, сумму четных и произведение нечетных чисел. Составьте блок - схему

11. Даны x и y. Рассчитать P по формуле:

$$P = x^y + \sqrt{2xy} + \sqrt[4]{4xy} + \cdots + \sqrt[10]{10xy}$$

. Составьте блок - схему

Задачи на Python

12. В зоопарке живут: 5 обезьян (monkeys), 3 льва (lions) и 2 медведя (bears). Сколько всего животных (animals) в зоопарке?
    Решите задачу с помощью переменных.
    Создайте для каждого животного отдельную переменную по его названию.
    Создайте переменную animals для всех животных и внутри неё сложите переменные.
    Результат выведите через print(), пользуясь переменной animals.
    Пример вывода:
    ▭ животных живёт в зоопарке
13. Создайте переменные с вашим именем (name) и возрастом (age) и выведите эти переменные на экран с помощью print, чтобы получилось одно предложение
    Пример вывода:
    Меня зовут Иван, мне 12 лет.
14. Пират нашёл сундук с кладом в 300 золотых монет. 30% от него он сразу раздал на долги. Также он оставил сундук открытым без присмотра на некоторое время, прилетели 4 вороны и каждая утащила по 10 монет. Сколько монет осталось у пирата?
    Используя переменные, итоговый результат сохраните в переменную coins и выведите её на экран, чтобы получилось:
    Ответ: у пирата осталось ▭ монет
15. Создайте в IDLE переменные с текущей датой:
    day = 25
    month = "января"
    year = 2021
    Выведите эти переменные через print() в Python Shell
    Пример вывода:
    Сегодня 25 января 2021 года
16. Объявите 4 переменные в IDLE, заполните их своими данными.
    Составьте предложение с этими переменными и выведите их все через один print() в Python Shell.
17. Напишите программу с помощью IDLE, переменных и функции print(), напечатайте в Shell изображение одного пингвина размером 5×9 символов.
    Для корректного вывода изображения сначала объеляем в переменных построчно все символы (например: a1 = ' _~_ ', a2 = ' (o o) ' и т.п.), а потом выводим построчно все символы через print (например: print(a1), print(a2) и т.п.)

```
····_~_····¶
···(o·o)···¶
··/··V··\··¶
·/(··_··)\·¶
···^^·^^···¶
```

    Примечание. Точки, которые вы видите на картинке – это количество пробелов (включен компонент «отобразить все знаки»)
18. Придумайте свой рисунок. Напишите программу с помощью IDLE, переменных и функции print(), напечатайте в Shell изображение

19. Создайте новый файл geron.py. Откройте его и мы будем работать в IDLE. Разработайте алгоритм и программу, в которой вычисляется площадь треугольника по трем сторонам. Вычисление проводиться по формуле Герона. $S=\sqrt{p(p-a)(p-b)(p-c)}$.
20. Напишите программу (файл *arithmetic.py*), которая предлагала бы пользователю решить пример 4 * 100 - 54. Потом выводила бы на экран правильный ответ и ответ пользователя.
21. Напишите программу (файл *chislo.py*), которая запросит у пользователя четыре числа. Отдельно сложите первые два и отдельно вторые два. Разделите первую сумму на вторую. Выведите результат на экран так, чтобы ответ содержал две цифры после запятой.
22. Посчитайте сумму трех введенных целых чисел
23. Пользователь вводит радиус круга, выведите площадь круга ($S=\pi R2$)
24. Сумма дробных. Посчитайте сумму трех введенных дробных чисел.
25. Школьники и яблоки. n школьников делят k яблок поровну, неделящийся остаток остается в корзинке. Сколько яблок достанется каждому школьнику? Сколько яблок останется в корзинке?
26. Напишите программу, которая считывает целое число, после чего на экран выводится следующее и предыдущее целое число с пояснительным текстом.
27. Напишите программу, которая считывает строку-разделитель и три строки, а затем выводит указанные строки через разделитель.
28. Напишите программу, которая считывает целое положительное число xx и выводит на экран последовательность чисел x, 2x, 3x, 4x, 5x, разделённых тремя черточками.
29. Напишите программу, которая находит полное число метров по заданному числу сантиметров.
30. Напишите программу, в которой рассчитывается сумма и произведение цифр положительного трёхзначного числа.
31. Напишите программу, в которой рассчитывается сумма и произведение цифр положительного четырехзначного числа.
32. Напишите программу для нахождения цифр трехзначного числа.
33. Напишите программу для нахождения цифр четырёхзначного числа.
34. Напишите программу, которая просит пользователя что-нибудь ввести с клавиатуры. Если он вводит какие-нибудь данные, то на экране должно выводиться сообщение "ОК". Если он не вводит данные, а просто нажимает Enter, то программа ничего не выводит на экран.
35. Напишите программу, которая запрашивает у пользователя число. Если оно больше нуля, то в ответ на экран выводится число 1. Если введенное число не является положительным, то на экран должно выводиться -1.
36. Напишите программу, которая вводит с клавиатуры число, затем прибавляет к этому числу 24, и проверяет полученный результат на условие. Если полученный результат больше 40, то вывести сообщение «Число {..} заданным критериям», иначе вывести сообщение об ошибке. Переменные для ввода и проверки условия, то разные переменные.
37. Найдите максимальное значение, введенное с клавиатуры, из двух чисел

38. Используя цикл while, выведите на экран для числа 2 его степени от 0 до 20. Возведение в степень в Python обозначается как **
39. Найдите значение функции на $y=5x^2-2x+1$ на отрезке [-5;5] с шагом 2
40. Вводится последовательность вещественных чисел. Известно, что последний элемент последовательности равен пяти. Определите, каких из них больше: положительных или отрицательных.
41. Вводится последовательность целых чисел, не равных нулю. Известно, что последний элемент последовательности равен нулю. Найдите среднее арифметическое этих чисел.
42. Дано положительное число N. Вывести все числа от 0 до N с помощью цикла while.
43. Даны два положительных числа K и N (K < N). Найти сумму всех нечетных чисел от K до N с помощью цикла while.
44. Дано положительное число N. Найти факториал числа N. Факториалом числа называется произведение всех чисел от 1 до N. Например, факториал числа 5 равен 5! = 1*2*3*4*5 = 120, 2! = 1*2 = 2, 9! = 1*2*3*4*5*6*7*8*9 = 362880
45. Пользователь вводит число N. Выведите все числа от 0 до N включительно.
46. Пользователь вводит числа K и N. Выведите все числа от K до N включительно.
47. Пользователь вводит числа K и N. Выведите сумму только четных чисел от K до N включительно.
48. Пользователь вводит число N. Найдите сумму чисел: 1 + 1/2 + 1/3 + ... + 1/N
49. Создайте список из 10 четных чисел и выведите его с помощью цикла for
50. Создайте список из 5 элементов. Сделайте срез от второго индекса до четвертого
51. Создайте пустой список и добавьте в него 10 случайных чисел и выведите их. В данной задаче нужно использовать функцию randint.
52. Создайте список из введенной пользователем строки и удалите из него символы 'a', 'e', 'o'
53. Даны два списка, удалите все элементы первого списка из второго
54. Создайте список из случайных чисел и найдите наибольший элемент в нем.
55. Найдите наименьший элемент в списке
56. Найдите сумму элементов списка
57. Найдите среднее арифметическое элементов списка
58. Площадь круга. Напишите функцию, которая получает в качестве аргумента радиус круга и находит его площадь.
59. На три. Напишите функцию, которая возвращает True, если число делится на 3, и False, если - нет.
60. Максимум в списке. Напишите функцию, которая возвращает максимальный элемент из переданного в нее списка.
61. Сколько четных. Напишите функцию, которая возвращает количество четных элементов в списке.
62. Уникальные. Напишите функцию, которая возвращает список с уникальными (неповторяющихся) элементам.

Раздел 3. Объектно-ориентированное программирование

Тема 3.1. Объектно-ориентированное программирование

План:

1. Объектно-ориентированное программирование
2. Структура объектно-ориентированного программирования
3. Процедурное программирование
4. Принципы объектно-ориентированного программирования

Объектно-ориентированное программирование - методология (парадигма) программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. В программе при этом в качестве основных логических конструктивных элементов используются объекты, а не алгоритмы.

Суть понятия объектно-ориентированного программирования в том, что все программы, написанные с применением этой парадигмы, состоят из объектов. Каждый объект — это определённая сущность со своими данными и набором доступных действий.

Например, нужно написать для интернет-магазина каталог товаров. Руководствуясь принципами ООП, в первую очередь нужно создать объекты: карточки товаров. Потом заполнить эти карточки данными: названием товара, свойствами, ценой. И потом прописать доступные действия для объектов: обновление, изменение, взаимодействие.

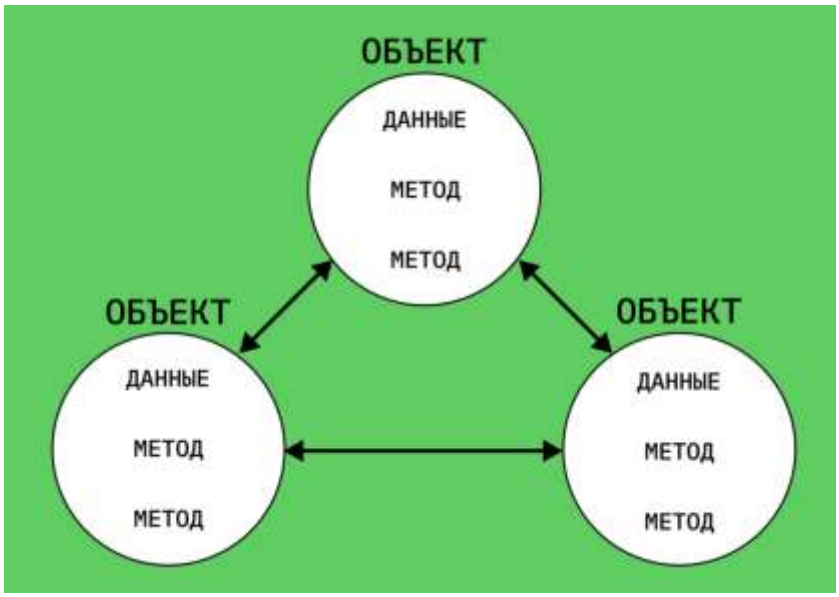


Рисунок 1

Так схематично выглядит программа, написанная по парадигме ООП

Кроме ООП, существуют и другие парадигмы. Из них наиболее распространена функциональная, в которой работают не с объектами, а с функциями. Если использовать функциональную парадигму, чтобы сделать каталог товаров, то начинать нужно не с карточек, а с функций, заполняющих эти карточки. То есть объект будет не отправной точкой, а результатом работы функции.

Обычно написать функцию быстрее, чем создавать объекты и прописывать взаимодействие между ними. Но если объём кода большой, работать с разрозненными функциями сложно.

Структура объектно-ориентированного программирования

В коде, написанном по парадигме ООП, выделяют четыре основных элемента:

Объект.

Часть кода, которая описывает элемент с конкретными характеристиками и функциями. Карточка товара в каталоге интернет-магазина — это объект. Кнопка «заказать» — тоже.

Класс.

Шаблон, на базе которого можно построить объект в программировании. Например, у интернет-магазина может быть класс «Карточка товара», который описывает общую структуру всех карточек. И уже из него создаются конкретные карточки — объекты.

Классы могут наследоваться друг от друга. Например, есть общий класс «Карточка товара» и вложенные классы, или подклассы: «Карточка бытовой техники», «Карточка ноутбука», «Карточка смартфона». Подкласс берёт свойства из родительского класса, например, цену товара, количество штук на складе или производителя. При этом имеет свои свойства, например, диагональ дисплея для «Карточки ноутбука» или количество сим-карт для «Карточки смартфона».

Метод.

Функция внутри объекта или класса, которая позволяет взаимодействовать с ним или другой частью кода. В примере с карточками товара метод может:

- Заполнить карточку конкретного объекта нужной информацией.
- Обновлять количество товара в наличии, сверяясь с БД.
- Сравнивать два товара между собой.
- Предлагать купить похожие товары.

Атрибут.

Характеристики объекта в программировании — например, цена, производитель или объём оперативной памяти. В классе прописывают, что такие атрибуты есть, а в объектах с помощью методов заполняют эти атрибуты данными.

Рисунок 2

Процедурное программирование - есть отражение фон Неймановской архитектуры компьютера. Программа, написанная на процедурном языке, представляет

собой последовательность команд, определяющих алгоритм решения задачи. Основная команда-присвоение, с помощью которой определяется и меняется память компьютера. Программа производит преобразование содержимого памяти, изменяя его от исходного состояния к результирующему.

Объектно-ориентированное программирование (ООП) — это методика разработки программ, в основе которой лежит понятие объект. Объект — это некоторая структура, соответствующая объекту реального мира, его поведению. Задача, решаемая с использованием методики ООП, описывается в терминах объектов и операций над ними, а программа при таком подходе представляет собой набор объектов и связей между ними.

Системы объектно-ориентированного программирования (ООП) дают возможность визуализировать процесс создания графического интерфейса разрабатываемого приложения

Взаимодействие программных объектов между собой и их изменения описываются с помощью программного кода

Создание программного кода в ООП базируется на использовании алгоритмических структур различных типов (линейной, ветвления, цикла), исполнителями которых выступают программные объекты

Класс — разновидность абстрактного типа данных в объектно-ориентированном программировании (ООП), характеризуемый способом своего построения. Другие абстрактные типы данных — метаклассы, интерфейсы, структуры, перечисления, — характеризуются какими-то своими, другими особенностями. Наряду с понятием «объекта» класс является ключевым понятием в ООП.

Программные объекты обладают свойствами, могут использовать методы и реагируют на события

Классы объектов являются «шаблонами», определяющими наборы свойств, методов и событий, по которым создаются объекты

Основными классами объектов являются классы, реализующие графический интерфейс проектов

Объект в программировании — некоторая сущность в виртуальном пространстве, обладающая определѐнным состоянием и поведением, имеющая заданные значения свойств (атрибутов) и операций над ними. 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.

Объект, созданный по шаблону класса объектов, является экземпляром класса и наследует весь набор свойств, методов и событий данного класса

Каждый экземпляр класса имеет уникальное для данного класса имя

Различные экземпляры класса обладают одинаковым набором свойств, однако значения свойств у них могут отличаться

Каждый объект обладает определенным набором свойств, первоначальные значения которых можно установить

Значения свойств объектов можно изменять и в программном коде:

Объект.Свойство := ЗначениеСвойства

Метод в объектно-ориентированном программировании — это функция или процедура, принадлежащая какому-то классу или объекту.

Как и процедура в процедурном программировании, метод состоит из некоторого количества операторов для выполнения какого-то действия и имеет набор входных аргументов.

Различают простые методы и статические методы (методы класса): простые методы имеют доступ к данным объекта (конкретного экземпляра данного класса), статические методы не имеют доступа к данным объекта и для их использования не нужно создавать экземпляры (данного класса).

Методы предоставляют интерфейс, при помощи которого осуществляется доступ к данным объекта некоторого класса, тем самым, обеспечивая инкапсуляцию данных. В зависимости от того, какой уровень доступа предоставляет тот или иной метод, выделяют:

- открытый (public) интерфейс — общий интерфейс для всех пользователей данного класса;
- защищѐнный (protected) интерфейс — внутренний интерфейс для всех наследников данного класса;
- закрытый (private) интерфейс — интерфейс, доступный только изнутри данного класса.

Такое разделение интерфейсов позволяет сохранять неизменным открытый интерфейс, но изменять внутреннюю реализацию.

Обратиться к методу объекта можно также с использованием точечной нотации: Объект.Метод

Событие в объектно-ориентированном программировании — это сообщение, которое возникает в различных точках исполняемого кода при выполнении определѐнных условий.

События предназначены для того, чтобы иметь возможность предусмотреть реакцию программного обеспечения.

Для решения поставленной задачи создаются обработчики событий: как только программа попадает в заданное состояние, происходит событие, посылается сообщение, а обработчик перехватывает это сообщение. В общем случае в обработчик не передаѐтся ничего, либо передаѐтся ссылка на объект, инициировавший (породивший) обрабатываемое событие. В особых случаях в обработчик передаются значения некоторых переменных или ссылки на какие-то другие объекты, чтобы обработка данного события могла учесть контекст возникновения события.

Самое простое событие — это событие, сообщающее о начале или о завершении некоторой процедуры. Событие, по сути, сообщает об изменении состояния некоторого объекта. Наиболее наглядно события представлены в пользовательском интерфейсе, когда каждое действие пользователя порождает цепочку событий, которые, затем обрабатываются в приложении.

Событие может создаваться пользователем (щелчок мышью или нажатие клавиши) или быть результатом воздействия других объектов

В качестве реакции на событие вызывается определенная процедура, которая может изменять свойства объекта или вызывать его методы

Графический интерфейс. Визуальное программирование позволяет делать графический интерфейс разрабатываемых приложений на основе форм и управляющих элементов.

В роли основных объектов при визуальном программировании выступают формы (Forms).

Форма представляет собой окно, на котором размещаются управляющие элементы.

Управляющие элементы — это командные кнопки (CommandButton), переключатели, или «флажки» (Checkbox), поля выбора, или «радиокнопки» (OptionsButton), списки (ListBox), текстовые поля (TextBox) и др.

Все основанные на объектах языки (C#, Java, C++, Smalltalk, Visual Basic и т.п.) должны отвечать трем основным **принципам объектно-ориентированного программирования (ООП):**

- Инкапсуляция
- Наследование
- Полиморфизм

Инкапсуляция — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. В объектно-ориентированном языке данные и код могут быть объединены в совершенно автономный черный ящик. Внутри такого ящика находятся все необходимые данные и код. Когда код и данные связываются вместе подобным образом, создается объект. Иными словами, объект — это элемент, поддерживающий инкапсуляцию.

Наследование — касается способности языка позволять строить новые определения классов на основе определений существующих классов. По сути, наследование позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом)

Т.е. наследование представляет собой процесс, в ходе которого один объект приобретает свойства другого объекта. Это очень важный процесс, поскольку он обеспечивает принцип иерархической классификации.

Полиморфизм - это свойство, которое позволяет одно и тоже имя использовать для решения нескольких технически разных задач.

В общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий.

Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование одного интерфейса для единого класса действий.

Выбор конкретного действия, в зависимости от ситуации, возлагается на компилятор.

Вопросы и задания для самоконтроля по разделу 3

1. Назовите и поясните два основных аспекта объектно-ориентированного программирования.
2. Поля, методы, атрибуты дайте характеристику.
3. Напишите синтаксис создания класса в языке Python.
4. Какой синтаксис используется при обращении к атрибуту класса?
5. Чем методы класса отличаются от обычных функций?
6. Поясните роль параметра self.

7. Какой синтаксис используется при обращении к методу класса?
8. С какой целью создается метод init ? Напишите его синтаксис.
9. Объясните роль статических .методов языка Python. Какие методы объявления статических методов вы знаете?
10. В чем заключается такой принцип ООП, как инкапсуляция?
11. Расскажите о методах создания закрытых атрибутов и способах доступа к ним.
12. С какой целью создаются свойства, и как происходит обращение к ним из клиентского кода?
13. Раскройте особенности одного из основных принципов ООП - наследования. Приведите синтаксис создания производного класса.

Решение задач

1. Создайте класс ФИГУРА с методами вычисления площади и периметра, а также методом, выводящим информацию о фигуре на экран. Создайте дочерние классы ПРЯМОУГОЛЬНИК, КРУГ, ТРЕУГОЛЬНИК со своими методами вычисления площади и периметра. Создайте список п фигур и выведите полную информацию о фигурах на экран.
2. Создайте класс ИЗДАНИЕ с методом, позволяющим вывести на экран информацию об издании, а также определить, является ли данное издание искомым. Создайте дочерние классы КНИГА (название, фамилия автора, год издания, издательство), СТАТЬЯ (название, фамилия автора, название журнала, его номер и год издания), ЭЛЕКТРОННЫЙ РЕСУРС (название, фамилия автора, ссылка, аннотация) со своими методами вывода информации на экран. Создайте список из п изданий, выведите полную информацию из списка, а также организуйте поиск изданий по фамилии автора.
3. Создайте класс ТРЕУГОЛЬНИК, заданный длинами двух сторон и угла между ними, с методами вычисления площади и периметра треугольника, а также методом, выводящим информацию о фигуре на экран. Создайте дочерние классы ПРЯМОУГОЛЬНЫЙ, РАВНОБЕДРЕННЫЙ, РАВНОСТОРОННИМ со своими методами вычисления площади и периметра. Создайте список п треугольников и выведите полную информацию о треугольниках на экран.
4. Создайте класс ТЕЛО с методами вычисления площади поверхности и объема, а также методом, выводящим информацию о фигуре на экран. Создайте дочерние классы ПАРАЛЛЕЛЕПИПЕД, Ш.АР, ПИРАМИДА со своими методами вычисления площади и объема. Создайте список п фигур и выведите полную информацию о фигурах на экран.
5. Создайте класс УРАВНЕНИЕ с методами вычисления корня уравнения и вывода результата на экран. Создайте дочерние классы ЛИНЕЙНОЕ, КВАДРАТНОЕ со своими методами вычисления корней и вывода на экран. Создайте список п уравнений и выведите полную информацию об уравнениях на экран.
6. Создайте класс ВАЛЮТА с методами перевода денежной суммы в рубли и вывода на экран. Создайте дочерние классы ДОЛЛАР, ЕВРО со своими

методами перевода и вывода на экран. Создайте список п валютных денежных сумм и выведите полную информацию о них на экран.

7. Создайте класс ПРОГРЕССИЯ с методами вычисления j-го элемента профессии, ее суммы и методом, выводящим сумму на экран. Создайте дочерние классы: АРИФМЕТИЧЕСКАЯ, ГЕОМЕТРИЧЕСКАЯ со своими методами вычисления. Создайте список п прогрессий и выведите сумму каждой из них экран.

Информационное обеспечение

Основные источники

1. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для среднего профессионального образования / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — Москва : Издательство Юрайт, 2022. — 137 с. — (Профессиональное образование). — ISBN 978-5-534-07321-8. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/493261
2. Чернышев, С. А. Основы программирования на Python : учебное пособие для среднего профессионального образования / С. А. Чернышев. — Москва : Издательство Юрайт, 2022. — 286 с. — (Профессиональное образование). — ISBN 978-5-534-15160-2. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/496897

Дополнительные источники

1. Гуриков, С. Р. Основы алгоритмизации и программирования на Python : учебное пособие / С.Р. Гуриков. — Москва : ИНФРА-М, 2022. — 343 с. — (Среднее профессиональное образование). - ISBN 978-5-16-016906-4. - Текст : электронный. - URL: https://znanium.com/catalog/product/1356004
2. Колдаев, В. Д. Основы алгоритмизации и программирования : учебное пособие / В.Д. Колдаев ; под ред. проф. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2022. — 414 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0733-7. - Текст : электронный. - URL: https://znanium.com/catalog/product/1735805
3. Огнева, М. В. Программирование на языке С++: практический курс : учебное пособие для среднего профессионального образования / М. В. Огнева, Е. В. Кудрина. — Москва : Издательство Юрайт, 2022. — 335 с. — (Профессиональное образование). — ISBN 978-5-534-05780-5. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/493047
4. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для среднего профессионального образования / Д. Ю. Федоров. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2022. — 210 с. — (Профессиональное образование). — ISBN 978-5-534-12829-1. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/492921